

MX



macromedia®

**FLASH™**MX  
2004

Guía de referencia de ActionScript

## Marcas comerciales

Add Life to the Web, Afterburner, Aftershock, Andromedia, Allaire, Animation PowerPack, Aria, Attain, Authorware, Authorware Star, Backstage, Bright Tiger, Clustercats, ColdFusion, Contribute, Design In Motion, Director, Dream Templates, Dreamweaver, Drumbeat 2000, EDJE, EJIPT, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, Generator, HomeSite, JFusion, JRun, Kawa, Know Your Site, Knowledge Objects, Knowledge Stream, Knowledge Track, LikeMinds, Lingo, Live Effects, el logotipo y el diseño de MacRecorder, Macromedia, Macromedia Action!, Macromedia Flash, el logotipo y el diseño de Macromedia M, Macromedia Spectra, el logotipo y el diseño de Macromedia xRes, MacroModel, Made with Macromedia, el logotipo y el diseño de Made with Macromedia, el logotipo y el diseño de MAGIC, Mediamaker, Movie Critic, Open Sesame!, Roundtrip, Roundtrip HTML, Shockwave, Sitespring, SoundEdit, Titlemaker, UltraDev, Web Design 101, what the web can be y Xtra son marcas registradas o marcas comerciales de Macromedia, Inc. y pueden estar registradas en Estados Unidos o en otras jurisdicciones, incluidas las internacionales. Otros nombres de productos, logotipos, diseños, títulos, palabras o frases mencionados en esta publicación pueden ser marcas comerciales, marcas de servicio o nombres registrados de Macromedia, Inc. o de otras entidades y pueden estar registrados en ciertas jurisdicciones, incluidas las internacionales.

## Información de terceros

Esta guía contiene vínculos a sitios Web de terceros que no están bajo el control de Macromedia y, por consiguiente, Macromedia no se hace responsable del contenido de dichos sitios Web. El acceso a uno de los sitios Web de terceros mencionados en esta guía será a cuenta y riesgo del usuario. Macromedia proporciona estos vínculos únicamente como ayuda y su inclusión no implica que Macromedia se haga responsable del contenido de dichos sitios Web.

La tecnología de compresión y descompresión de voz tiene licencia de Nellymoser, Inc. ([www.nellymoser.com](http://www.nellymoser.com)).



La tecnología de compresión y descompresión de vídeo Sorenson™ Spark™ tiene licencia de Sorenson Media, Inc.

Navegador Opera® Copyright © 1995-2002 Opera Software ASA y sus proveedores. Todos los derechos reservados.

## Limitación de garantías de Apple

**APPLE COMPUTER, INC. NO GARANTIZA, DE FORMA EXPRESA NI IMPLÍCITA, LA COMERCIABILIDAD O IDONEIDAD PARA UN FIN DETERMINADO DEL PAQUETE DE SOFTWARE INFORMÁTICO INCLUIDO. LA EXCLUSIÓN DE GARANTÍAS IMPLÍCITAS NO ESTÁ PERMITIDA EN ALGUNOS ESTADOS. LA RESTRICCIÓN ANTERIOR PUEDE NO AFECTARLE. ESTA GARANTÍA LE PROPORCIONA DERECHOS LEGALES ESPECÍFICOS. PUEDE TENER OTROS DERECHOS QUE VARÍAN SEGÚN LA LEGISLACIÓN LOCAL.**

**Copyright © 2003 Macromedia, Inc. Todos los derechos reservados. No se permite la copia, fotocopia, reproducción, traducción ni la conversión en formato electrónico o legible por equipos, ya sea de forma total o parcial de este manual, sin la autorización previa por escrito de Macromedia, Inc. Número de referencia ZFL70M400SP**

## Agradecimientos

Director: Erick Vera

Dirección del proyecto: Stephanie Gowin, Barbara Nelson

Redacción: Jody Bleye, Mary Burger, Kim Diezel, Stephanie Gowin, Dan Harris, Barbara Herbert, Barbara Nelson, Shirley Ong, Tim Statler

Directora de edición: Rosana Francescato

Edición: Linda Adler, Mary Ferguson, Mary Kraemer, Noreen Maher, Antonio Padial, Lisa Stanziano, Anne Szabla

Dirección de la producción: Patrice O'Neill

Producción y diseño multimedia: Adam Barnett, Christopher Basmajian, Aaron Begley, John Francis, Jeff Harmon

Localización: Tim Hussey, Seungmin Lee, Masayo Noda, Simone Pux, Yuko Yagi, Jorge G. Villanueva

Primera edición: septiembre de 2003

Macromedia, Inc.  
600 Townsend St.  
San Francisco, CA 94103

# CONTENIDO

<b>INTRODUCCIÓN:</b> Primeros pasos con ActionScript . . . . .	9
Destinatarios . . . . .	9
Requisitos del sistema . . . . .	9
Utilización de la documentación . . . . .	9
Convenciones tipográficas . . . . .	10
Términos utilizados en este documento . . . . .	10
Recursos adicionales . . . . .	10

## **PARTE I:** Bienvenido a ActionScript

---

<b>CAPÍTULO 1:</b> Novedades de Flash MX 2004 ActionScript . . . . .	15
Elementos de lenguaje nuevos y modificados . . . . .	15
Nuevo modelo de seguridad y archivos SWF heredados . . . . .	17
Transferencia de scripts existentes a Flash Player 7 . . . . .	17
Cambios en el editor de ActionScript . . . . .	23
Cambios en la depuración . . . . .	24
Nuevo modelo de programación orientada a objetos . . . . .	25
<b>CAPÍTULO 2:</b> Conceptos básicos de ActionScript . . . . .	27
Diferencias entre ActionScript y JavaScript . . . . .	27
Compatibilidad de ActionScript con Unicode . . . . .	28
Terminología . . . . .	28
Sintaxis . . . . .	31
Tipos de datos . . . . .	36
Asignación de tipos de datos a elementos . . . . .	39
Variables . . . . .	43
Utilización de operadores para manipular los valores de las expresiones . . . . .	47
Especificación de la ruta de un objeto . . . . .	53
Utilización de funciones incorporadas . . . . .	54
Creación de funciones . . . . .	54

<b>CAPÍTULO 3:</b> Escritura y depuración de scripts . . . . .	57
Control de la ejecución de ActionScript . . . . .	57
Utilización del panel Acciones y la ventana Script . . . . .	60
Utilización del editor de ActionScript . . . . .	63
Depuración de los scripts. . . . .	71
Utilización del panel Salida . . . . .	80
Actualización de Flash Player para realizar pruebas . . . . .	82

## **PARTE II: Gestión de eventos y creación de interacciones**

---

<b>CAPÍTULO 4:</b> Gestión de eventos. . . . .	85
Utilización de métodos de controlador de eventos . . . . .	85
Utilización de detectores de eventos. . . . .	87
Utilización de controladores de eventos de botones y de clips de película. . . . .	88
Creación de clips de película con estados de botón . . . . .	89
Ámbito del controlador de eventos . . . . .	90
Ámbito de la palabra clave “this” . . . . .	91
<b>CAPÍTULO 5:</b> Creación de interacciones con ActionScript . . . . .	93
Eventos e interacciones . . . . .	93
Control de la reproducción de archivos SWF. . . . .	93
Creación de interactividad y efectos visuales . . . . .	96
Análisis de un script de ejemplo . . . . .	110

## **PARTE III: Trabajo con objetos y clases**

---

<b>CAPÍTULO 6:</b> Utilización de clases incorporadas . . . . .	115
Clases e instancias . . . . .	115
Información general sobre las clases incorporadas . . . . .	116
<b>CAPÍTULO 7:</b> Trabajo con clips de película. . . . .	123
Control de clips de película con ActionScript . . . . .	123
Llamada a varios métodos en un solo clip de película. . . . .	124
Carga y descarga de archivos SWF adicionales. . . . .	125
Especificación de una línea de tiempo raíz para archivos SWF cargados. . . . .	126
Cargar archivos JPEG en clips de película . . . . .	127
Modificación de la posición y el aspecto de un clip de película . . . . .	127
Clips de película que se pueden arrastrar . . . . .	128
Creación de clips de película en tiempo de ejecución. . . . .	128
Adición de parámetros a clips de película creados de forma dinámica . . . . .	131
Gestión de las profundidades de los clips de película . . . . .	132
Dibujo de formas con ActionScript . . . . .	133

Utilización de clips de película como máscaras . . . . .	134
Gestión de eventos de clip de película . . . . .	135
Asignación de una clase a un símbolo de clip de película . . . . .	135
Inicialización de las propiedades de clase . . . . .	136
<b>CAPÍTULO 8: Trabajo con texto . . . . .</b>	<b>139</b>
Utilización de la clase TextField . . . . .	140
Creación de campos de texto durante la ejecución . . . . .	141
Utilización de la clase TextFormat . . . . .	141
Aplicación de formato al texto con hojas de estilos en cascada . . . . .	143
Utilización de texto en formato HTML . . . . .	151
Creación de texto desplazable . . . . .	158
<b>CAPÍTULO 9: Creación de clases con ActionScript 2.0 . . . . .</b>	<b>161</b>
Principios de la programación orientada a objetos . . . . .	162
Utilización de clases: un ejemplo sencillo . . . . .	163
Creación y utilización de clases . . . . .	167
Miembros de clase e instancia . . . . .	171
Creación y utilización de interfaces . . . . .	173
Introducción a la ruta de clases . . . . .	175
Utilización de paquetes . . . . .	177
Importación de clases . . . . .	178
Métodos get/set implícitos . . . . .	179
Creación de clases dinámicas . . . . .	180
Cómo se compilan y exportan las clases . . . . .	180
 <b>PARTE IV: Trabajo con datos y medios externos</b>	
 <b>CAPÍTULO 10: Trabajo con datos externos . . . . .</b>	<b>185</b>
Envío y carga de variables hacia y desde una fuente remota . . . . .	185
Envío de mensajes hacia y desde Flash Player . . . . .	193
Funciones de seguridad de Flash Player . . . . .	196
 <b>CAPÍTULO 11: Trabajo con elementos multimedia externos . . . . .</b>	<b>201</b>
Información general sobre la carga de archivos multimedia externos . . . . .	201
Carga de archivos SWF y JPEG externos . . . . .	202
Carga de archivos MP3 externos . . . . .	203
Lectura de etiquetas ID3 en archivos MP3 . . . . .	204
Reproducción dinámica de archivos FLV externos . . . . .	205
Precarga de archivos multimedia externos . . . . .	206

<b>CAPÍTULO 12:</b> Diccionario de ActionScript . . . . .	213
Entrada de muestra para la mayoría de los elementos de ActionScript . . . . .	213
Entrada de muestra para clases. . . . .	214
Contenido del diccionario . . . . .	215
Clase Accessibility . . . . .	274
Clase Arguments . . . . .	279
Clase Array . . . . .	281
Clase Boolean . . . . .	297
Clase Button . . . . .	300
Clase Camera . . . . .	319
Clase Color . . . . .	339
Clase ContextMenu. . . . .	343
Clase ContextMenuItem . . . . .	349
Clase CustomActions . . . . .	355
Clase Date. . . . .	357
Clase Error . . . . .	384
Clase Function . . . . .	395
Clase Key . . . . .	414
Clase LoadVars . . . . .	431
Clase LocalConnection . . . . .	439
Clase Math . . . . .	452
Clase Microphone. . . . .	467
Clase Mouse . . . . .	482
Clase MovieClip . . . . .	487
Clase MovieClipLoader. . . . .	546
Clase NetConnection . . . . .	557
Clase NetStream . . . . .	558
Clase Number . . . . .	571
Clase Object . . . . .	576
Clase PrintJob . . . . .	597
Clase Selection . . . . .	610
Clase SharedObject. . . . .	619
Clase Sound . . . . .	627
Clase Stage . . . . .	642
Clase String. . . . .	651
Clase System . . . . .	663
Objeto System.capabilities. . . . .	668
Objeto System.security . . . . .	677
Clase TextField . . . . .	682
Clase TextField.StyleSheet. . . . .	705
Clase TextFormat . . . . .	718
Objeto TextSnapshot. . . . .	727
Clase Video . . . . .	745
Clase XML . . . . .	752
Clase XMLNode . . . . .	773
Clase XMLSocket . . . . .	774

<b>APÉNDICE A:</b> Mensajes de error. ....	783
<b>APÉNDICE B:</b> Precedencia y asociatividad de los operadores. ....	789
<b>APÉNDICE C:</b> Teclas del teclado y valores de códigos de tecla. ....	791
Letras de la A a la Z y números estándar del 0 al 9. ....	791
Teclas del teclado numérico. ....	792
Teclas de función. ....	793
Otras teclas. ....	794
<b>APÉNDICE D:</b> Escritura de scripts para versiones anteriores de Flash Player ..	797
Utilización de versiones anteriores de Flash Player. ....	797
Utilización de Flash MX 2004 para crear contenido para Flash Player 4. ....	797
<b>APÉNDICE E:</b> Programación orientada a objetos con ActionScript 1. ....	801
ActionScript 1. ....	801
<b>ÍNDICE ALFABÉTICO</b> .....	809





# INTRODUCCIÓN

## Primeros pasos con ActionScript

Macromedia Flash MX 2004 y Flash MX Professional 2004 son las herramientas estándar de edición profesional para la creación de publicaciones Web de gran impacto. ActionScript es el lenguaje que deberá utilizar si desea desarrollar una aplicación en Flash. Para utilizar Flash, no es necesario utilizar ActionScript, pero si desea que los usuarios puedan interactuar, trabajar con objetos que no sean los incorporados en Flash (como por ejemplo, botones y clips de película) o convertir un archivo SWF en una experiencia de usuario más fiable, es posible que desee utilizar este lenguaje.

### Destinatarios

En el manual se presupone que el usuario ya ha instalado Flash MX 2004 or Flash MX Professional 2004 y sabe cómo utilizarlo. Deberá saber cómo colocar objetos en el escenario y cómo manipularlos en el entorno de edición de Flash. Si ha escrito programas anteriormente, ActionScript le resultará familiar. Pero aunque no haya escrito ninguno, no le costará familiarizarse con ActionScript. Es fácil empezar con comandos muy simples y aumentar el grado de complejidad conforme se va avanzando.

### Requisitos del sistema

Los componentes de ActionScript no tienen ningún requisito del sistema adicional a los de Flash MX 2004 or Flash MX Professional 2004. No obstante, en la documentación se presupone que el usuario utiliza la configuración de publicación predeterminada para los archivos Flash: Flash Player 7 y ActionScript 2.0. Si cambia alguno de estos valores, es posible que las explicaciones y los ejemplos de código que se muestran en la documentación no sean válidos.

### Utilización de la documentación

Este documento proporciona información general sobre la sintaxis de ActionScript, sobre cómo utilizar ActionScript al trabajar con distintos tipos de objetos y ofrece, también, detalles sobre la sintaxis y el uso de cada uno de los elementos de lenguaje. Empiece por aprender la terminología y los conceptos básicos utilizados en el resto del documento (véase el [Capítulo 2, “Conceptos básicos de ActionScript”, en la página 27](#)). Después, aprenda los mecanismos para escribir y depurar scripts de Flash (véase el [Capítulo 3, “Escritura y depuración de scripts”, en la página 57](#)).

Antes de escribir sus propios scripts, debe completar las lecciones “Creación de scripts con ActionScript” y “Creación de un formulario con lógica condicional y envío de datos”, que proporcionan una introducción práctica para trabajar con ActionScript. Para encontrar estas lecciones, seleccione Ayuda > Cómo > Inicio rápido.

Cuando comprenda los conceptos básicos, estará preparado para utilizar la información incluida en el resto del documento, ya que está relacionada con el efecto específico que desea conseguir. Por ejemplo, si desea aprender a escribir un script que realice una acción concreta cuando un usuario haga clic con el ratón, consulte [Capítulo 4, “Gestión de eventos”, en la página 85](#).

Cuando encuentre información sobre un comando concreto que desee utilizar, puede consultar la entrada correspondiente en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#); todos los elementos del lenguaje están enumerados en orden alfabético.

## Convenciones tipográficas

En este manual se utilizan las siguientes convenciones tipográficas:

- La fuente para código indica que se trata de código de ActionScript.
- *La fuente para código en cursiva* indica que es un elemento, por ejemplo un nombre de objeto o un parámetro de ActionScript, que se reemplaza con el texto escrito por el usuario al escribir un script.

## Términos utilizados en este documento

En este manual se utilizan los términos siguientes:

- *Usted* hace referencia al desarrollador que escribe un script o una aplicación.
- *El usuario* hace referencia a la persona que ejecutará los scripts y las aplicaciones.
- *La fase de compilación* es la fase en la que se publica, exporta, prueba o depura un documento.
- *El tiempo de ejecución* es el espacio de tiempo en el que se ejecuta un script en Flash Player.

Los términos de ActionScript como *método* y *objeto* se definen en el [Capítulo 2, “Conceptos básicos de ActionScript”, en la página 27](#).

## Recursos adicionales

La documentación específica sobre Flash y los productos relacionados se proporciona por separado.

- Para información sobre cómo trabajar en el entorno de edición de Flash, consulte el apartado Utilización de Flash de la Ayuda. Si desea información sobre los componentes, consulte el apartado Utilización de componentes de la Ayuda.
- Si desea información sobre cómo crear aplicaciones de comunicación con Flash Communication Server, consulte *Desarrollo de aplicaciones con Flash Communication Server y Administrando Flash Communication Server*.
- Para más información sobre el acceso a los servicios Web con aplicaciones de Flash, consulte *Utilización de convirtiendo a remoto de Flash*.

El sitio Web Macromedia DevNet ([http://www.macromedia.com/go/developer\\_es](http://www.macromedia.com/go/developer_es)) se actualiza regularmente con la última información sobre Flash, además de consejos de usuarios expertos, temas más complejos, ejemplos, sugerencias y otras actualizaciones. Visite el sitio Web regularmente para conocer las últimas noticias sobre Flash y cómo sacar el máximo partido del programa.

El centro de soporte de Macromedia Flash ([http://www.macromedia.com/go/flash\\_support\\_sp](http://www.macromedia.com/go/flash_support_sp)) proporciona notas técnicas, documentación actualizada y vínculos a otros recursos de la comunidad Flash.



# PARTE I

## Bienvenido a ActionScript

En esta sección se incluye información básica acerca del lenguaje ActionScript.

En el capítulo 1 se incluye información acerca de las novedades y modificaciones de ActionScript y Flash Player 7. Si ha utilizado ActionScript anteriormente, asegúrese de leer esta información detenidamente.

Si no tiene experiencia con ActionScript, lea los capítulos 2 y 3 para tener una buena base para entender la terminología y la sintaxis de ActionScript, así como para aprender a crear y depurar scripts.

Capítulo 1: Novedades de Flash MX 2004 ActionScript .....	15
Capítulo 2: Conceptos básicos de ActionScript .....	27
Capítulo 3: Escritura y depuración de scripts .....	57



# CAPÍTULO 1

## Novedades de Flash MX 2004 ActionScript

Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004 proporcionan varias mejoras que facilitan la escritura de scripts más sólidos con el lenguaje ActionScript. Entre estas nuevas funciones, que se explican más adelante en este capítulo, se incluyen nuevos elementos de lenguaje, herramientas mejoradas para la edición y la depuración (véase [“Cambios en el editor de ActionScript” en la página 23](#) y [“Cambios en la depuración” en la página 24](#)), y la introducción de un modelo de programación más orientada a objetos (véase [“Nuevo modelo de programación orientada a objetos” en la página 25](#)).

En este capítulo también se incluye una amplia sección que debe leer atentamente si tiene previsto publicar en Flash Player 7 archivos Flash MX existentes o archivos de una versión anterior (véase [“Transferencia de scripts existentes a Flash Player 7” en la página 17](#)).

### Elementos de lenguaje nuevos y modificados

En esta sección se describen los elementos del lenguaje ActionScript nuevos o que han cambiado en Flash MX 2004. Para utilizar dichos elementos en los scripts, debe utilizar Flash Player 7 (predeterminado) al publicar los documentos.

- Los métodos `Array.sort()` y `Array.sortOn()` permiten añadir parámetros para especificar opciones de ordenación adicionales, como orden ascendente o descendente, si se debe distinguir entre mayúsculas y minúsculas al ordenar, etc.
- Las propiedades `Button.menu`, `MovieClip.menu` y `TextField.menu` funcionan con las clases `ContextMenu` y `ContextMenuItem` para permitirle asociar elementos de menú contextuales con objetos `Button`, `MovieClip` o `TextField`.
- La [Clase `ContextMenu`](#) y la [Clase `ContextMenuItem`](#) permiten personalizar el menú contextual que se muestra cuando un usuario hace clic con el botón derecho (Microsoft Windows) o hace clic mientras presiona la tecla Control (Macintosh) en Flash Player.
- La [Clase `Error`](#) y los comandos `throw` y `try..catch..finally` permiten implementar una gestión de excepciones más fiable.
- Los métodos `LoadVars.setRequestHeader()` y `XML.setRequestHeader()` añaden o cambian encabezados de petición HTTP (como `Content-Type` o `SOAPAction`) enviados con acciones POST.
- La función `MMExecute()` permite enviar comandos de la interfaz API JavaScript de Flash desde ActionScript.

- (Sólo para Windows) El detector de eventos `Mouse.onMouseWheel` se genera cuando el usuario se desplaza utilizando la rueda del ratón.
- El método `MovieClip.getNextHighestDepth()` permite crear instancias de clip de película durante la ejecución y garantizar que sus objetos aparecerán delante de los otros objetos en el espacio del orden z de un clip de película principal. El método `MovieClip.getInstanceAtDepth()` permite acceder a instancias de clip de película creadas de forma dinámica utilizando la profundidad como índice de búsqueda.
- El método `MovieClip.getSWFVersion()` permite determinar qué versión de Flash Player admite el archivo SWF cargado.
- El método `MovieClip.getTextSnapshot()` y el **Objeto `TextSnapshot`** permiten trabajar con texto que se encuentra en campos de texto estáticos de un clip de película.
- La propiedad `MovieClip._lockroot` permite especificar que un clip de película actuará como `_root` para los clips de película que se carguen en él o que el significado de `_root` en un clip de película no cambiará si dicho clip se carga en otro clip de película.
- La **Clase `MovieClipLoader`** permite controlar el progreso de los archivos a medida que se cargan en clips de película.
- La **Clase `NetConnection`** y la **Clase `NetStream`** permiten transmitir archivos de vídeo locales (archivos FLV).
- La **Clase `PrintJob`** proporciona, tanto a usted como al usuario, más control sobre la impresión desde Flash Player.
- El controlador de eventos `Sound.onID3` proporciona acceso a datos ID3 asociados con un objeto Sound que contiene un archivo MP3.
- La propiedad `Sound.ID3` proporciona acceso a los metadatos que forman parte de un archivo MP3.
- La **Clase `System`** tiene objetos y métodos nuevos, mientras que el **Objeto `System.capabilities`** tiene varias propiedades nuevas.
- La propiedad `TextField.condenseWhite` permite quitar el espacio en blanco adicional de los campos de texto HTML que se muestran en un navegador.
- La propiedad `TextField.mouseWheelEnabled` permite especificar si el contenido de un campo de texto se debe desplazar cuando se sitúa el puntero del ratón sobre un campo de texto y el usuario hace girar la rueda del ratón.
- La **Clase `TextField.StyleSheet`** permite crear un objeto de hoja de estilos que contenga reglas de formato de texto, como tamaño de fuente, color y otros estilos de formato.
- La propiedad `TextField.styleSheet` permite adjuntar un objeto de hoja de estilos a un campo de texto.
- El método `TextFormat.getTextExtent()` acepta un nuevo parámetro y el objeto que devuelve contiene un nuevo miembro.
- El método `XML.setRequestHeader()` permite añadir o cambiar encabezados de petición HTTP (como `Content-Type` o `SOAPAction`) enviados con acciones POST.



## Nuevo modelo de seguridad y archivos SWF heredados

Las reglas que utiliza Flash Player para determinar si dos dominios son el mismo han cambiado en Flash Player 7. Además, han cambiado las reglas que determinan cuándo y cómo un archivo SWF proporcionado desde un dominio HTTP puede acceder a un archivo SWF o cargar datos de un dominio HTTPS. En la mayoría de los casos, estos cambios no le afectarán, a menos que esté transfiriendo los archivos SWF existentes a Flash Player 7.

No obstante, si tiene publicados archivos SWF para Flash Player 6 o una versión anterior que cargan datos desde un archivo almacenado en un servidor, y el archivo SWF que efectúa la llamada se ejecuta en Flash Player 7, el usuario puede ver un cuadro de diálogo que antes no aparecía, en el que se le pregunta si desea otorgar acceso. Puede evitar la aparición de este cuadro de diálogo implementando un *archivo de política* en el sitio en el que se guardan los datos. Para obtener más información sobre este cuadro de diálogo, consulte [“Compatibilidad con modelos de seguridad de Flash Player anteriores” en la página 199](#).

Puede que también tenga que implementar un archivo de política si utiliza bibliotecas compartidas en tiempo de ejecución. Si el archivo SWF que se está cargando o que se ha cargado se ha publicado para Flash Player 7 y los archivos que se están cargando y los que se han cargado no se encuentran disponibles en el mismo dominio exacto, utilice un archivo de política para permitir el acceso. Para obtener más información sobre los archivos de política, consulte [“Carga de datos de varios dominios” en la página 198](#).

## Transferencia de scripts existentes a Flash Player 7

Al igual que en cualquier nueva versión, Flash Player 7 admite más comandos de ActionScript que las versiones anteriores del reproductor; estos comandos le permitirán implementar scripts más sólidos. (Consulte [“Elementos de lenguaje nuevos y modificados” en la página 15](#).) Sin embargo, si ya ha utilizado alguno de estos comandos en sus scripts existentes, es posible que el script no funcione correctamente al publicarlo en Flash Player 7.

Por ejemplo, si tiene un script con una función denominada Error, el script puede compilarse aparentemente, pero es posible que no funcione de la forma prevista en Flash Player 7, ya que Error es una clase incorporada (y, por lo tanto, una palabra reservada) en ActionScript. Puede arreglar el script cambiando el nombre de la función Error por otro, como ErrorCondition.

Además, en Flash Player 7 se incluyen varios cambios que influyen en cómo un archivo SWF puede acceder a otro archivo SWF, cómo pueden cargarse los datos externos y cómo se puede acceder a la configuración y los datos locales (como la configuración de confidencialidad y los objetos compartidos localmente persistentes). Por último, el comportamiento de algunas de las funciones existentes ha cambiado.

Si tiene scripts escritos para Flash Player 6 o para una versión anterior que desea publicar para Flash Player 7, es posible que necesite modificarlos para que se adapten a la implementación de Flash Player 7 y funcionen correctamente. Estas modificaciones se tratan en esta sección.

## Conformidad con el estándar ECMA-262, edición 4

Se han implementado varios cambios en Flash Player 7 para adaptarlo más a la propuesta ECMA-262, edición 4 (consulte [www.mozilla.org/js/language/es4/index.html](http://www.mozilla.org/js/language/es4/index.html)). Además de las técnicas de programación basadas en clases disponibles en ActionScript 2.0 (véase “[Nuevo modelo de programación orientada a objetos](#)” en la página 25), se han añadido otras funciones y se han modificado determinados comportamientos. Además, al publicar para Flash Player 7 y utilizar ActionScript 2.0, se puede convertir un tipo de objeto en otro. Para más información, consulte “[Conversión de objetos](#)” en la página 41. Para utilizar estas funciones no es necesario actualizar los scripts existentes; sin embargo, utilícelas si publica sus scripts en Flash Player 7 y los revisa y mejora constantemente.

A diferencia de los cambios mencionados anteriormente, los cambios detallados en la tabla siguiente (algunos de los cuales también mejoran la conformidad con ECMA) pueden hacer que los scripts existentes funcionen de forma diferente a como funcionaban anteriormente. Si ha utilizado estas funciones en scripts existentes que desea publicar en Flash Player 7, repase los cambios para comprobar si el código sigue funcionando correctamente o si debe volver a escribirlo. En concreto, puesto que `undefined` se evalúa de forma distinta en determinados casos, debe inicializar todas las variables en los scripts que transfiera a Flash Player 7.

Archivo SWF publicado para Flash Player 7	Archivo SWF publicado para versiones anteriores de Flash Player
Se admite la distinción entre mayúsculas y minúsculas (los nombres de variables que se diferencian sólo en este aspecto se interpretan como variables distintas). Este cambio también afecta a los archivos cargados con <code>#include</code> y a las variables externas cargadas con <code>LoadVars.load()</code> . Para más información, consulte “ <a href="#">Distinción entre mayúsculas y minúsculas</a> ” en la página 31.	No se admite la distinción entre mayúsculas y minúsculas (los nombres de variables que se diferencian sólo en este aspecto se interpretan como las mismas variables).
La evaluación de <code>undefined</code> en un contexto numérico devuelve <code>NaN</code> . <pre>myCount +=1; trace(myCount); // NaN</pre>	La evaluación de <code>undefined</code> en un contexto numérico devuelve <code>0</code> . <pre>myCount +=1; trace(myCount); // 1</pre>
Cuando <code>undefined</code> se convierte en una cadena, el resultado es <code>undefined</code> . <pre>firstname = "Joan "; lastname = "Flender"; trace(firstname + middlename + lastname); // Joan undefinedFlender</pre>	Cuando <code>undefined</code> se convierte en una cadena, el resultado es <code>""</code> (cadena vacía). <pre>firstname = "Joan "; lastname = "Flender"; trace(firstname + middlename + lastname); // Joan Flender</pre>

Archivo SWF publicado para Flash Player 7	Archivo SWF publicado para versiones anteriores de Flash Player
<p>Cuando se convierte una cadena en un valor booleano, el resultado es <code>true</code> si la cadena tiene una longitud mayor que cero; el resultado es <code>false</code> para una cadena vacía.</p> <p>Cuando se define la longitud de una matriz, sólo se puede utilizar una cadena numérica válida. Por ejemplo, "6" funciona, pero " 6" o "6xyz" no.</p> <pre>my_array=new Array(); my_array[" 6"] ="x"; trace(my_array.length); // 0 my_array["6xyz"] ="x"; trace(my_array.length); // 0 my_array["6"] ="x"; trace(my_array.length); // 7</pre>	<p>Cuando se convierte una cadena en un valor booleano, la cadena se convierte primeramente en un número; el resultado es <code>true</code> si el número es diferente de cero; en caso contrario, el resultado es <code>false</code>.</p> <p>Cuando se define la longitud de una matriz, incluso puede utilizarse una cadena numérica mal formada.</p> <pre>my_array=new Array(); my_array[" 6"] ="x"; trace(my_array.length); // 7 my_array["6xyz"] ="x"; trace(my_array.length); // 7 my_array["6"] ="x"; trace(my_array.length); // 7</pre>

## Reglas de nombres de dominio para la configuración y los datos locales

En Flash Player 6, se utilizan de forma predeterminada las reglas de coincidencia de superdominio al acceder a la configuración local (como los permisos de acceso a la cámara o al micrófono) o a los datos localmente persistentes (objetos compartidos). Es decir, la configuración y los datos para los archivos SWF albergados en `here.xyz.com`, `there.xyz.com` y `xyz.com` están compartidos y se almacenan todos en `xyz.com`.

En Flash Player 7, se utilizan de forma predeterminada las reglas de coincidencia de dominio exacto. Es decir, la configuración y los datos para un archivo albergado en `here.xyz.com` se almacenan en `here.xyz.com`, la configuración y los datos para un archivo albergado en `there.xyz.com` se almacenan en `there.xyz.com`, y así sucesivamente.

La nueva propiedad `System.exactSettings` permite especificar las reglas que se van a utilizar. Esta propiedad se admite para los archivos publicados en Flash Player 6 o una versión posterior. Para los archivos publicados en Flash Player 6, el valor predeterminado es `false`, lo cual significa que se utilizarán las reglas de coincidencia de superdominio. Para los archivos publicados en Flash Player 7, el valor predeterminado es `true`, lo cual significa que se utilizarán las reglas de coincidencia de dominio exacto.

Si utiliza la configuración o los datos locales persistentes y desea publicar un archivo SWF de Flash Player 6 en Flash Player 7, es posible que tenga que establecer este valor en `false` en el archivo transferido.

Para más información, consulte [System.exactSettings](#) en la página 663.

## Acceso a varios dominios y subdominios entre archivos SWF

Al desarrollar una serie de archivos SWF que se comunican entre sí —por ejemplo, cuando utiliza `loadMovie()`, `MovieClip.loadMovie()`, `MovieClipLoader.LoadClip()` u objetos `LocalConnection`—, puede albergar las películas en diferentes dominios, o en diferentes subdominios de un mismo superdominio.

En los archivos publicados en Flash Player 5 o una versión anterior, no hay restricciones en el acceso a varios dominios o subdominios.

En los archivos publicados en Flash Player 6, se podía utilizar el controlador `LocalConnection.allowDomain` o el método `System.security.allowDomain()` para especificar que se permitía el acceso a varios dominios (por ejemplo, para permitir que un archivo en `someOtherSite.com` pudiera acceder a un archivo en `someSite.com`), y no se necesitaba ningún comando para permitir el acceso a subdominios (por ejemplo, un archivo en `store.someSite.com` podía acceder a un archivo en `www.someSite.com`).

Los archivos publicados en Flash Player 7 implementan el acceso entre archivos SWF de forma distinta a como lo hacían en versiones anteriores, de dos formas. En primer lugar, Flash Player 7 implementa las reglas de coincidencia de dominio exacto en lugar de las reglas de coincidencia de superdominio. Por lo tanto, el archivo al cual se accede (aunque se haya publicado en una versión de Player anterior a Flash Player 7) debe permitir explícitamente el acceso a varios dominios o subdominios; este tema se trata a continuación. En segundo lugar, un archivo albergado en un sitio que utilice un protocolo seguro (HTTPS) debe permitir explícitamente el acceso de un archivo albergado en un sitio que utilice un protocolo inseguro (HTTP o FTP); este tema se trata en la siguiente sección ([“Acceso de protocolo HTTP a HTTPS entre archivos SWF” en la página 21](#)).

Puesto que Flash Player 7 implementa reglas de coincidencia de dominio exacto en lugar de reglas de coincidencia de superdominio, es posible que tenga que modificar los scripts existentes si quiere acceder a ellos desde archivos publicados en Flash Player 7. Los archivos modificados se podrán publicar igualmente en Flash Player 6. Si ha utilizado alguna sentencia `LocalConnection.allowDomain()` o `System.security.allowDomain()` en sus archivos y ha especificado que se permita el acceso a sitios de superdominio, deberá cambiar sus parámetros de modo que especifiquen dominios exactos. El código siguiente muestra un ejemplo de los tipos de cambios que puede tener que realizar:

```
// Comandos de Flash Player 6 en un archivo SWF en www.anyOldSite.com
// permitir el acceso de los archivos SWF albergados en www.someSite.com
// o en store.someSite.com
System.security.allowDomain("someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="someSite.com");
}
// Comandos correspondientes para permitir el acceso de los archivos SWF
// publicados en Flash Player 7
System.security.allowDomain("www.someSite.com", "store.someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com" ||
        sendingDomain=="store.someSite.com");
}
```

Tendrá que añadir sentencias como éstas a sus archivos si actualmente no las está utilizando. Por ejemplo, si el archivo SWF está albergado en `www.someSite.com` y desea permitir el acceso de un archivo SWF publicado en Flash Player 7 en `store.someSite.com`, deberá añadir sentencias como las siguientes al archivo que está en `www.someSite.com` (el archivo que está en `www.someSite.com` podrá publicarse igualmente en Flash Player 6):

```
System.security.allowDomain("store.someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="store.someSite.com");
}
```

En resumen, tendrá que modificar sus archivos añadiendo o cambiando sentencias `allowDomain` si quiere publicar archivos en Flash Player 7 que cumplen las siguientes condiciones:

- Ha implementado la creación de scripts entre archivos SWF (mediante `loadMovie()`, `MovieClip.loadMovie()`, `MovieClipLoader.LoadClip()` u objetos `LocalConnection`).
- El archivo SWF (de cualquier versión) llamado no está albergado en un sitio que utiliza un protocolo seguro (HTTPS) o bien los archivos SWF que efectúan la llamada y que reciben la llamada están albergados en sitios HTTPS. Si sólo se alberga en HTTPS el archivo SWF llamado, consulte [“Acceso de protocolo HTTP a HTTPS entre archivos SWF” en la página 21](#).
- Los archivos SWF no están en el mismo dominio (por ejemplo, un archivo está en `www.domain.com` y otro en `store.domain.com`).

Deberá realizar los siguientes cambios:

- Si el archivo SWF llamado se publica en Flash Player 7, incluya `System.security.allowDomain` o `LocalConnection.allowDomain` en el mismo, para que utilice la coincidencia de nombre de dominio exacto.
- Si el archivo SWF llamado se publica en Flash Player 6, modifíquelo añadiendo o cambiando una sentencia `System.security.allowDomain` o `LocalConnection.allowDomain`, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección. Puede publicar el archivo modificado en Flash Player 6 o 7.
- Si el archivo SWF llamado se publica en Flash Player 5 o una versión anterior, transféralo a Flash Player 6 o 7 y añada una sentencia `System.security.allowDomain`, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección. Los objetos `LocalConnection` no se admiten en Flash Player 5 o una versión anterior.

## Acceso de protocolo HTTP a HTTPS entre archivos SWF

Tal como se ha explicado en la sección anterior, las reglas para el acceso a varios dominios y para el acceso a subdominios han cambiado en Flash Player 7. Además de las reglas de coincidencia de dominio exacto que ahora se implementan, debe permitir explícitamente que los archivos albergados en sitios que utilizan un protocolo inseguro puedan acceder a los archivos albergados en sitios que utilizan un protocolo seguro (HTTPS). Según si el archivo llamado se publica en Flash Player 7 o Flash Player 6, deberá implementar una de las sentencias `allowDomain` (consulte [“Acceso a varios dominios y subdominios entre archivos SWF” en la página 19](#)) o utilizar las nuevas sentencias `LocalConnection.allowInsecureDomain` o `System.security.allowInsecureDomain()`.

**Advertencia:** al implementar una sentencia `allowInsecureDomain()` se pone en juego la seguridad ofrecida por el protocolo HTTPS. Estos cambios sólo deben realizarse si no puede reorganizar el sitio de manera que todos los archivos SWF se encuentren disponibles en el protocolo HTTPS.

El código siguiente muestra un ejemplo de los tipos de cambios que puede tener que realizar:

```
// Comandos en un archivo SWF de Flash Player 6 en https://www.someSite.com
// permitir el acceso de los archivos SWF de Flash Player 7 albergados
// en http://www.someSite.com o en http://www.someOtherSite.com
System.security.allowDomain("someOtherSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="someOtherSite.com");
}
// Comandos correspondientes en un archivo SWF de Flash Player 7
// permitir el acceso de los archivos SWF de Flash Player 7 albergados
// en http://www.someSite.com o en http://www.someOtherSite.com
System.security.allowInsecureDomain("www.someSite.com",
    "www.someOtherSite.com");
my_lc.allowInsecureDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com" ||
        sendingDomain=="www.someOtherSite.com");
}
```

Tendrá que añadir sentencias como éstas a sus archivos si actualmente no las está utilizando. Deberá hacerse una modificación aunque ambos archivos se encuentren en el mismo dominio (por ejemplo, si un archivo en <http://www.domain.com> llama a otro archivo en <https://www.domain.com>).

En resumen, deberá modificar sus archivos añadiendo o cambiando sentencias si quiere publicar archivos en Flash Player 7 que cumplen las siguientes condiciones:

- Ha implementado la creación de scripts entre archivos SWF (mediante `loadMovie()`, `MovieClip.loadMovie()`, `MovieClipLoader.LoadClip()` u objetos `LocalConnection`).
- El archivo que efectúa la llamada no se alberga en un sitio con protocolo HTTPS, y el archivo llamado sí que utiliza HTTPS.

Deberá realizar los siguientes cambios:

- Si el archivo llamado se publica en Flash Player 7, incluya `System.security.allowInsecureDomain` o `LocalConnection.allowInsecureDomain` en el mismo, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección. Esta sentencia es necesaria aunque el archivo SWF que efectúa la llamada y el archivo SWF llamado se encuentren en el mismo dominio.
- Si el archivo llamado se publica en Flash Player 6 o una versión anterior, y tanto el archivo que efectúa la llamada como el archivo llamado se encuentran en el mismo dominio (por ejemplo, un archivo en <http://www.domain.com> llama a otro archivo en <https://www.domain.com>), no es necesaria ninguna modificación.
- Si el archivo llamado se publica en Flash Player 6, los archivos no se encuentran en el mismo dominio y no desea transferir el archivo llamado a Flash Player 7, modifique dicho archivo añadiendo o cambiando una sentencia `System.security.allowDomain` o `LocalConnection.allowDomain`, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección.
- Si el archivo llamado se publica en Flash Player 6 y desea transferirlo a Flash Player 7, incluya `System.security.allowInsecureDomain` o `LocalConnection.allowInsecureDomain` en el mismo, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección. Esta sentencia es necesaria aunque ambos archivos se encuentren en el mismo dominio.

- Si el archivo llamado se publica en Flash Player 5 o una versión anterior, y los archivos no se encuentran en el mismo dominio, puede realizar una de estas dos acciones. Puede transferir el archivo llamado a Flash Player 6 y añadir o cambiar una sentencia `System.security.allowDomain`, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección, o bien puede transferir el archivo llamado a Flash Player 7 e incluir una sentencia `System.security.allowInsecureDomain` en el mismo, para que utilice la coincidencia de nombre de dominio exacto, tal como se muestra en los ejemplos de código explicados en esta misma sección.

## Archivos de política de servidor para permitir el acceso a los datos

Un documento de Flash puede cargar datos desde una fuente externa mediante una de las siguientes llamadas para cargar datos: `XML.load()`, `XML.sendAndLoad()`, `LoadVars.load()`, `LoadVars.sendAndLoad()`, `loadVariables()`, `loadVariablesNum()`, `MovieClip.loadVariables()`, `XMLSocket.connect()` y `Macromedia Flash Remoting (NetServices.createGatewayConnection)`. Asimismo, un archivo SWF puede importar bibliotecas compartidas en tiempo de ejecución (RSL) o elementos definidos en otro archivo SWF durante el tiempo de ejecución. De forma predeterminada, los datos o las RSL deben residir en el mismo dominio que el archivo SWF que está cargando los medios o datos externos.

Para que los datos y los elementos de las bibliotecas compartidas en tiempo de ejecución puedan ser utilizados por los archivos SWF de diferentes dominios, utilice un *archivo de política para distintos dominios*. Un archivo de política para distintos dominios es un archivo XML que permite al servidor indicar que sus datos y documentos están disponibles para los archivos SWF de determinados dominios o de todos los dominios. Cualquier archivo SWF que se encuentre disponible en un dominio especificado por el archivo de política del servidor podrá acceder a los datos y RSL desde dicho servidor.

Si está cargando datos externos, debe crear archivos de política aunque no tenga previsto transferir ningún archivo a Flash Player 7. Si utiliza RSL, deberá crear archivos de política si el archivo que efectúa la llamada o el archivo llamado se publica en Flash Player 7.

Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

## Cambios en el editor de ActionScript

El editor de ActionScript se ha actualizado para que sea más consistente y más fácil de utilizar. En esta sección se ofrece un resumen de dichos cambios.

**Ajustar texto** Ahora puede utilizar el menú emergente Opciones de los paneles Script, Depurador y Salida para activar o desactivar el ajuste de texto. También puede alternar el ajuste de texto mediante el menú emergente del panel Acciones. El método abreviado de teclado es `Ctrl+Mayús+W` (Windows) o `Comando+Mayús+W` (Macintosh).

**Visualización de la ayuda contextual** Cuando el puntero se sitúa sobre un elemento de lenguaje ActionScript en la caja de herramientas Acciones o en el panel Script, puede utilizar el elemento Ver ayuda del menú contextual para ver una página de ayuda relativa al elemento.

**Importación de scripts** Cuando selecciona Importar script en el menú emergente del panel Acciones, el script importado se copia en el script en el punto de inserción del archivo de código. En versiones anteriores de Flash, cuando se importaba un script se sobrescribía el contenido del script existente.

**Puntos de corte con un solo clic** Para añadir un punto de corte de depuración antes de una línea de código en el panel Depurador o el panel Script del panel Acciones, puede hacer clic en el margen izquierdo. En versiones anteriores de Flash, al hacer clic en el margen izquierdo se seleccionaba una línea de código. Ahora, para seleccionar una línea de código se debe presionar Ctrl y hacer clic (Windows) o presionar Comando y hacer clic (Macintosh).

**Desaparición de los modos Normal y Experto en el panel Acciones** En versiones anteriores de Flash, podía trabajar en el panel Acciones en modo Normal (se especificaban las opciones y los parámetros para crear el código) o en modo Experto (se añadían comandos directamente en el panel Script). En Flash MX 2004 y Flash MX Professional 2004, sólo se puede utilizar el panel Acciones añadiendo comandos directamente en el panel Script. Sigue siendo posible arrastrar comandos de la caja de herramientas Acciones al panel Script o utilizar el botón Añadir (+) situado encima del panel Script para añadir comandos a un script.

**Fijación de varios scripts** Puede fijar varios scripts dentro de un archivo FLA a lo largo de la parte inferior del panel Script en el panel Acciones. En las versiones anteriores de Flash sólo se puede fijar un script a la vez.

**Navegador de scripts** Ahora el panel Acciones contiene dos paneles en la parte izquierda: la caja de herramientas Acciones y un nuevo navegador de scripts. Dicho navegador de scripts es una representación visual de la estructura del archivo FLA; puede desplazarse por el archivo FLA para localizar código de ActionScript.

**Ventana Script integrada para editar archivos externos (sólo para Flash**

**Professional)** Puede utilizar el editor de ActionScript en una ventana Script (independiente del panel Acciones) para escribir y editar archivos de script externos. Ahora se admiten la aplicación de color a la sintaxis, las sugerencias para el código y otras preferencias en la ventana Script; además, la caja de herramientas Acciones también está disponible. Para ver la ventana Script, utilice Archivo > Nuevo y seleccione el tipo de archivo externo que desea editar. Puede tener varios archivos externos abiertos al mismo tiempo; los nombres de archivo se muestran en las fichas situadas en la parte superior de la ventana Script (las fichas sólo aparecen en Windows).

## Cambios en la depuración

En esta sección se describen los cambios que mejoran la depuración de los scripts.

**La ventana Salida ahora es el panel Salida** Ahora puede mover y acoplar el panel Salida igual que lo haría con cualquier otro panel de Flash.

**Informe de errores mejorado durante la compilación** Además de proporcionar una gestión de excepciones más fiable, ActionScript 2.0 proporciona diversos errores de tiempo de compilación nuevos. Para más información, consulte [Apéndice A, “Mensajes de error”, en la página 783](#).

**Gestión de excepciones mejorada** La clase Error y los comandos `throw` y `try...catch...finally` permiten implementar una gestión de excepciones más fiable.



## Nuevo modelo de programación orientada a objetos

El lenguaje ActionScript ha crecido y se ha desarrollado desde su introducción hace algunos años. Con cada nueva versión de Flash se han añadido al lenguaje palabras clave, objetos, métodos y otros elementos adicionales. No obstante, a diferencia de las versiones anteriores de Flash, Flash MX 2004 y Flash MX Professional 2004 introducen diversos elementos de lenguaje nuevos que implementan la programación orientada a objetos de una manera más estándar. Como estos elementos de lenguaje constituyen una mejora significativa del núcleo del lenguaje ActionScript, representan una nueva versión de ActionScript por sí mismos: ActionScript 2.0.

ActionScript 2.0 no es un lenguaje nuevo. Comprende un conjunto central de elementos de lenguaje que facilitan el desarrollo de programas orientados a objetos. Con la introducción de palabras clave tales como `class`, `interface`, `extends` e `implements`, ahora la sintaxis de ActionScript resulta más sencilla de aprender para los programadores que están familiarizados con otros lenguajes. Los programadores sin experiencia pueden aprender una terminología más estándar que podrán aplicar a otros lenguajes orientados a objetos que puedan estudiar en el futuro.

ActionScript 2.0 admite todos los elementos estándar del lenguaje ActionScript; simplemente, permite escribir scripts que se ajustan mejor a los estándares utilizados en otros lenguajes orientados a objetos, como Java. ActionScript 2.0 resultará de especial interés para los desarrolladores con conocimientos intermedios o avanzados de Flash que crean aplicaciones que requieren la implementación de clases y subclases. ActionScript 2.0 también permite declarar el tipo de objeto de una variable al crearla (véase [“Strict data typing” en la página 40](#)) y proporciona una gestión de errores de compilador muy mejorada (véase [Apéndice A, “Mensajes de error”, en la página 783](#)).

A continuación, se indican los nuevos elementos de lenguaje de ActionScript 2.0.

- `class`
- `extends`
- `implements`
- `interface`
- `dynamic`
- `static`
- `public`
- `private`
- `get`
- `set`
- `import`

Puntos clave de ActionScript 2.0:

- Los scripts que utilizan ActionScript 2.0 para definir clases o interfaces deben almacenarse como archivos de script externos, con una sola clase definida en cada script; es decir, las clases y las interfaces no pueden definirse en el panel Acciones.
- Puede importar archivos de clase individuales implícitamente (almacenándolos en una ubicación especificada por rutas de búsqueda globales o específicas del documento y luego utilizándolos en un script) o explícitamente (utilizando el comando `import`); puede importar paquetes (conjuntos de archivos de clase en un directorio) utilizando caracteres comodín.
- Las aplicaciones desarrolladas con ActionScript 2.0 son compatibles con Flash Player 6 y versiones posteriores.

**Atención:** la configuración de publicación predeterminada para los nuevos archivos creados en Flash MX 2004 es ActionScript 2.0. Si tiene previsto modificar un archivo FLA existente de modo que utilice la sintaxis de ActionScript 2.0, asegúrese de que el archivo FLA tiene especificado ActionScript 2.0 en su configuración de publicación. Si no lo tiene especificado, el archivo no se compilará de forma correcta, aunque Flash no genere errores de compilador.

Para más información sobre la utilización de ActionScript 2.0 para escribir programas orientados a objetos en Flash, consulte [Capítulo 9, “Creación de clases con ActionScript 2.0”](#), en [la página 161](#).

# CAPÍTULO 2

## Conceptos básicos de ActionScript

ActionScript sigue reglas gramaticales y de puntuación que determinan qué caracteres y palabras se utilizan para dar significado a una sentencia y el orden en que se deben escribir. Por ejemplo, en español, una oración termina con un punto. En ActionScript, se utiliza un punto y coma para finalizar una sentencia.

A continuación se detallan las reglas generales que se aplican a todo el lenguaje ActionScript. La mayoría de los términos de ActionScript también tienen sus propios requisitos; para saber cuáles son las reglas de un término determinado, consulte la entrada correspondiente en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

### Diferencias entre ActionScript y JavaScript

ActionScript es parecido al lenguaje de programación JavaScript. No es necesario tener conocimientos de JavaScript para utilizar y aprender ActionScript; sin embargo, si los tiene, ActionScript le resultará familiar.

En este manual no se pretende enseñar programación en general. Existen muchos recursos disponibles que proporcionan información sobre los conceptos generales de programación y sobre el lenguaje JavaScript.

- El documento ECMA-262 de la Asociación europea de fabricantes de PC (ECMA, European Computers Manufacturers Association) se deriva de JavaScript y sirve de estándar internacional para el lenguaje JavaScript. ActionScript se basa en la especificación ECMA-262.
- Netscape DevEdge Online tiene un sitio Web central para programadores de JavaScript (<http://developer.netscape.com/tech/javascript/index.html>) que contiene documentación y artículos que son útiles para comprender ActionScript. El recurso más útil es el manual *Core JavaScript Guide*.

A continuación se detallan algunas de las diferencias entre ActionScript y JavaScript:

- ActionScript no admite objetos específicos de navegador como Documento, Ventana y Ancla.
- ActionScript no admite completamente todos los objetos incorporados de JavaScript.
- ActionScript no admite algunas construcciones sintácticas de JavaScript, como las etiquetas de sentencia.
- En ActionScript, la acción `eval()` sólo puede realizar referencias de variables.

## Compatibilidad de ActionScript con Unicode

Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004 admiten la codificación de texto Unicode para ActionScript. Ello significa que se puede incluir texto en diferentes idiomas en un archivo de ActionScript. Por ejemplo, puede incluir texto en inglés, japonés y francés en un mismo archivo.

Puede definir en las preferencias de ActionScript el tipo de codificación que se debe utilizar al importar o exportar archivos de ActionScript. Puede seleccionar la codificación UTF-8 o la codificación predeterminada. UTF-8 es un formato Unicode de 8 bits, mientras que la codificación predeterminada es el formato de codificación que admite el idioma del sistema que esté utilizando, que también se conoce como *página de códigos tradicional*.

Normalmente, si se importan o exportan archivos de ActionScript en formato UTF-8, se utiliza la preferencia UTF-8. Si se importan o exportan archivos con la página de códigos tradicional que utiliza el sistema, se utiliza la preferencia Codificación predeterminada.

Si el texto de los scripts no tiene el aspecto esperado al abrir o al importar un archivo, cambie la preferencia de codificación de importación. Si recibe un mensaje de advertencia al exportar archivos de ActionScript, cambie la preferencia de codificación de exportación o desactive este mensaje en las preferencias de ActionScript.

### Para seleccionar las opciones de codificación de texto para importar o exportar archivos de ActionScript:

- 1 En el cuadro de diálogo Preferencias (Edición > Preferencias), haga clic en la ficha ActionScript.
- 2 En Opciones de edición, realice una o las dos acciones siguientes:
  - En Abrir/importar, seleccione UTF-8 para abrir o importar con la codificación Unicode, o seleccione Codificación predeterminada para abrir o importar con la codificación del idioma que utilice el sistema.
  - En Guardar/exportar, seleccione UTF-8 para guardar o exportar con la codificación Unicode, o seleccione Codificación predeterminada para guardar o exportar con la codificación del idioma que utilice el sistema.

### Para activar o desactivar el mensaje de advertencia de la codificación de exportación:

- 1 En el cuadro de diálogo Preferencias (Edición > Preferencias), haga clic en la ficha Advertencias.
- 2 Seleccione o anule la selección de Avisar acerca de conflictos de codificación al exportar archivos .as.

**Atención:** el comando Probar película (véase [“Depuración de los scripts” en la página 71](#)) fallará si alguna parte de la ruta del archivo SWF presenta caracteres no representables mediante el esquema de codificación MBCS. Por ejemplo, las rutas en japonés no funcionan en un sistema inglés. Esta limitación se aplica a todas las áreas de la aplicación que utilizan el reproductor externo.

## Terminología

Como con cualquier otro lenguaje de creación de scripts, ActionScript utiliza su propia terminología. A continuación, se explican brevemente los términos de ActionScript importantes.

**Acciones:** sentencias que indican a un archivo SWF que debe llevar a cabo alguna acción durante su reproducción. Por ejemplo, `gotoAndStop()` desplaza la cabeza lectora a un fotograma o etiqueta determinados. En este manual, los términos *acción* y *sentencia* tienen el mismo significado.

**Booleano:** valor verdadero (true) o falso (false).

**Clase:** tipo de datos que puede emplearse para definir un nuevo tipo de objeto. Para definir una clase, se utiliza la palabra clave `class` en un archivo de script externo (no en un script que se esté escribiendo en el panel Acciones).

**Constante:** elemento cuyo valor no cambia. Por ejemplo, la constante `Key.TAB` siempre tiene el mismo significado: indica la tecla Tabulador de un teclado. Las constantes son útiles para comparar valores.

**Constructor:** función que se utiliza para definir las propiedades y métodos de una clase. Por definición, los constructores son funciones incluidas en definiciones de clases que tienen el mismo nombre que la clase. Por ejemplo, el código siguiente define una clase `Circle` e implementa una función constructora:

```
// archivo Circle.as
class Circle {
    private var radius:Number
    private var circumference:Number
// constructor
    function Circle(radius:Number) {
        circumference = 2 * Math.PI * radius;
    }
}
```

El término *constructor* también se emplea para crear un objeto (crear instancias del mismo) basado en una clase determinada. Las sentencias siguientes son constructores de la clase `Array` incorporada y de la clase `Circle` personalizada:

```
my_array:Array = new Array();
my_circle:Circle = new Circle();
```

**Tipo de datos:** describe el tipo de información que puede contener una variable o un elemento de `ActionScript`. Los tipos de datos de `ActionScript` son: cadena, número, valor booleano, objeto, clip de película, función, nulo y no definido. Para más información, consulte [“Tipos de datos” en la página 36](#).

**Eventos:** acciones que tienen lugar durante la reproducción de un archivo SWF. Por ejemplo, cuando se carga un clip de película se generan diferentes eventos: la cabeza lectora accede a un fotograma, el usuario hace clic en un botón o clip de película o el usuario introduce información mediante el teclado.

**Controladores de eventos:** acciones especiales que gestionan eventos como `mouseDown` o `load`. Se distinguen dos tipos de controladores de eventos de `ActionScript`: métodos de controlador de eventos y detectores de eventos. También existen dos controladores de eventos, `on()` y `onClipEvent()`, que pueden asignarse directamente a botones y clips de película. En la caja de herramientas Acciones, cada objeto de `ActionScript` que tiene métodos de controlador de eventos o detectores de eventos cuenta con una subcategoría denominada Eventos o Detectores. Algunos comandos pueden utilizarse como controladores de eventos y como detectores de eventos y se incluyen en ambas subcategorías.

**Expresión:** cualquier combinación válida de símbolos de `ActionScript` que representan un valor. Una expresión está formada por operadores y operandos. Por ejemplo, en la expresión `x + 2`, `x` y `2` son operandos y `+` es un operador.

**Función:** bloque de código reutilizable que acepta parámetros y puede devolver un valor. Para más información, consulte [“Creación de funciones” en la página 54](#).

**Identificador:** nombre que se utiliza para identificar una variable, una propiedad, un objeto, una función o un método. El primer carácter debe ser una letra, un carácter de subrayado (\_) o un símbolo de dólar (\$). Los caracteres siguientes deben ser una letra, un número, un carácter de subrayado (\_) o un símbolo de dólar (\$). Por ejemplo, `firstName` es el nombre de una variable.

**Instancia:** objeto que pertenece a una determinada clase. Cada instancia de una clase contiene todas las propiedades y métodos de dicha clase. Por ejemplo, todos los clips de película son instancias de la clase `MovieClip`, de modo que puede utilizar cualquiera de los métodos o propiedades de la clase `MovieClip` con cualquier instancia de clip de película.

**Nombre de instancia:** nombre exclusivo que se puede utilizar en instancias de clips de película y de botones a través de scripts. Utilice el inspector de propiedades para asignar nombres de instancia a las instancias del escenario. Por ejemplo, un símbolo maestro de la biblioteca podría denominarse `counter` y las dos instancias de dicho símbolo en el archivo SWF podrían denominarse `scorePlayer1_mc` y `scorePlayer2_mc`. En el siguiente código se utilizan nombres de instancia para establecer una variable denominada `score` en cada instancia del clip de película:

```
_root.scorePlayer1_mc.score += 1;
_root.scorePlayer2_mc.score -= 1;
```

Se pueden emplear sufijos especiales al denominar instancias para que aparezcan sugerencias para el código (véase [“Utilización de las sugerencias para el código” en la página 66](#)) a medida que se escribe el código. Para más información, consulte [“Utilización de sufijos para activar las sugerencias para el código” en la página 64](#).

**Palabra clave:** palabra reservada que tiene un significado especial. Por ejemplo, `var` es una palabra clave que se utiliza para declarar variables locales. Una palabra clave no puede utilizarse como identificador. Por ejemplo, `var` no es un nombre de variable válido. Para obtener una lista de palabras clave, consulte [“Palabras clave” en la página 36](#).

**Método:** función asociada a una clase. Por ejemplo, `getBytesLoaded()` es un método incorporado asociado a la clase `MovieClip`. También puede crear funciones que actúen como métodos, ya sea para objetos basados en clases incorporadas o para objetos basados en clases que haya creado. Por ejemplo, en el código siguiente, `clear()` pasa a ser un método de un objeto `controller` definido anteriormente:

```
function reset(){
    this.x_pos = 0;
    this.x_pos = 0;
}
controller.clear = reset;
controller.clear();
```

**Objeto:** conjunto de propiedades y métodos; cada objeto tiene su propio nombre y es una instancia de una clase determinada. Los objetos incorporados están predefinidos en el lenguaje `ActionScript`. Por ejemplo, el objeto incorporado `Date` ofrece información procedente del reloj del sistema.

**Operador:** término que calcula un nuevo valor a partir de uno o más valores. Por ejemplo, el operador de suma (+) suma dos o más valores para generar un nuevo valor. Los valores manipulados por los operadores se denominan *operandos*.

**Parámetro** (denominado también *argumento*): marcador de posición que permite pasar valores a las funciones. La siguiente función `welcome()`, por ejemplo, utiliza dos valores que recibe de los parámetros `firstName` y `hobby`:

```
function welcome(firstName, hobby) {  
    welcomeText = "Hola, " + firstName + "Se nota que te gusta " + hobby;  
}
```

**Paquetes:** son directorios que contienen uno o más archivos de clase y que residen en un directorio classpath determinado (véase [“Introducción a la ruta de clases” en la página 175](#)).

**Propiedad:** atributo que define un objeto. Por ejemplo, `_visible` es una propiedad de los clips de película que define si el clip está visible u oculto.

**Rutas de destino:** direcciones jerárquicas de nombres de instancias de clips de película, variables y objetos de un archivo SWF. El nombre de una instancia de clip de película se asigna en el inspector de propiedades del clip de película (la línea de tiempo principal siempre tiene el nombre `_root`). Se puede utilizar una ruta de destino para dirigir una acción a un clip de película u obtener o definir el valor de una variable. Por ejemplo, la sentencia siguiente es la ruta de destino a la variable `volume` dentro del clip de película `stereoControl`:

```
_root.stereoControl.volume
```

Para obtener más información sobre rutas de destino, consulte “Rutas de destino absolutas y relativas” en el apartado Utilización de Flash de la Ayuda.

**Variable:** identificador que almacena valores de cualquier tipo de datos. Las variables pueden crearse, modificarse y actualizarse. Los valores almacenados en una variable pueden recuperarse para ser utilizados en scripts. En el siguiente ejemplo, los identificadores situados a la izquierda de los signos igual son variables:

```
var x = 5;  
var name = "Lolo";  
var c_color = new Color(mcinstanceName);
```

Para más información sobre el uso de variables, consulte [“Variables” en la página 43](#).

## Sintaxis

Como en todos los lenguajes, ActionScript tiene reglas sintácticas que deben cumplirse para escribir scripts que se puedan compilar y ejecutar correctamente. En esta sección se describen los elementos que conforman la sintaxis de ActionScript.

### Distinción entre mayúsculas y minúsculas

En un lenguaje de programación que distingue entre mayúsculas y minúsculas, los nombres de variables que sólo se diferencian en las mayúsculas o minúsculas (`book` y `Book`) se consideran diferentes. Por lo tanto, es aconsejable seguir un criterio coherente en el uso de mayúsculas y minúsculas, como el utilizado en este manual, para facilitar la identificación de nombres de funciones y variables en el código ActionScript.

Al publicar archivos en Flash Player 7 o una versión posterior, Flash aplica la distinción entre mayúsculas y minúsculas tanto si utiliza ActionScript 1 como ActionScript 2.0. Esto significa que en las palabras clave, nombres de clase, variables, nombres de método, etc. se distingue entre mayúsculas y minúsculas. Por ejemplo:

```
// En archivos de Flash Player 7
// y ActionScript 1 o ActionScript 2.0
//
// Define las propiedades de dos objetos diferentes
cat.hilite = true;
CAT.hilite = true;

// Crea tres variables diferentes
var myVar=10;
var myvar=10;
var mYvAr=10;
// No genera un error
var array = new Array();
var date = new Date();
```

Este cambio también afecta a las variables externas cargadas con `LoadVars.load()`.

Además, la distinción entre mayúsculas y minúsculas se aplica a los scripts externos, como los scripts o los archivos de clase de ActionScript 2.0 que se importan con el comando `#include`. Si publica archivos en Flash Player 7 y previamente ha creado archivos externos que ha añadido a los scripts mediante la sentencia `#include`, revise cada archivo y verifique que ha utilizado un criterio de distinción entre mayúsculas y minúsculas coherente en todo el archivo. Una forma de hacerlo es abrir el archivo en la ventana Script (sólo en Flash Professional) o en un archivo FLA nuevo, definir la configuración de publicación para Flash Player 7 y copiar el contenido del archivo en el panel Acciones. A continuación, utilice el botón Revisar sintaxis (véase [“Comprobación de la sintaxis y la puntuación” en la página 69](#)) o publique el archivo; los errores causados por conflictos de denominación aparecerán en el panel Salida.

Si activa la función Color de sintaxis, los elementos del lenguaje que haya escrito con el formato correcto de mayúsculas y minúsculas aparecen en azul de forma predeterminada. Para más información, consulte [“Palabras clave” en la página 36](#) y [“Resaltado de la sintaxis” en la página 63](#).

## Sintaxis con punto

En ActionScript, se utiliza un punto (.) para indicar las propiedades o métodos relacionados con un objeto o un clip de película. También se utiliza para identificar la ruta de destino a un clip de película, variable, función u objeto. Una expresión que utiliza una sintaxis con punto empieza con el nombre del objeto o clip de película seguido de un punto y termina con el elemento que desee especificar.

Por ejemplo, la propiedad de clip de película `_x` indica la posición del clip de película en el eje *x* en el escenario. La expresión `ballMC._x` se refiere a la propiedad `_x` de la instancia del clip de película `ballMC`.

Otro ejemplo es `submit`, una variable definida en el clip de película `form` que se encuentra anidado dentro del clip de película `shoppingCart`. La expresión `shoppingCart.form.submit = true` define la variable `submit` de la instancia `form` como `true`.



La expresión de un método de un objeto o de un clip de película sigue el mismo esquema. Por ejemplo, el método `play()` de la instancia de clip de película `ball_mc` desplaza la cabeza lectora en la línea de tiempo de `ball_mc`, como se indica en la sentencia siguiente:

```
ball_mc.play();
```

La sintaxis con punto también utiliza dos alias especiales: `_root` y `_parent`. El alias `_root` se refiere a la línea de tiempo principal. Puede utilizar el alias `_root` para crear una ruta de destino absoluta. Por ejemplo, la siguiente sentencia llama a la función `buildGameBoard` en el clip de película `functions` en la línea de tiempo principal:

```
_root.functions.buildGameBoard();
```

Puede utilizar el alias `_parent` para referirse a un clip de película en el que está anidado el objeto actual. También puede utilizar `_parent` para crear una ruta de destino relativa. Por ejemplo, si el clip de película `dog_mc` está anidado en el clip de película `animal_mc`, la sentencia siguiente de la instancia `dog_mc` indica a `animal_mc` que pare:

```
_parent.stop();
```

## Sintaxis con barras

La sintaxis con barras se utilizó en Flash 3 y 4 para indicar la ruta de destino de un clip de película o de una variable. Flash Player 7 todavía admite esta sintaxis, aunque no es aconsejable utilizarla. ActionScript 2.0 no la admite. No obstante, si está creando contenido específicamente para Flash Player 4, debe utilizar la sintaxis con barras. Para más información, consulte [“Utilización de sintaxis con barras” en la página 799](#).

## Llaves

Los controladores de eventos, las definiciones de clase y las funciones de ActionScript se agrupan en bloques mediante llaves (`{}`). Puede colocar la llave de apertura en la misma línea de la declaración o en la línea siguiente, tal como se muestra en los ejemplos que aparecen a continuación. Para facilitar la lectura del código, es recomendable elegir un formato y utilizarlo de forma coherente.

```
// Controlador de eventos
on(release) {
    myDate = new Date();
    currentMonth = myDate.getMonth();
}

on(release)
{
    myDate = new Date();
    currentMonth = myDate.getMonth();
}

// Clase
class Circle(radius) {
}

class Square(side)
{
}

// Función
circleArea = function(radius) {
```

```

    return radius * radius * MATH.PI;
}
squareArea = function(side)
{
    return side * side;
}

```

Puede comprobar si las llaves de apertura tienen sus correspondientes llaves de cierre en los scripts; véase [“Comprobación de la sintaxis y la puntuación” en la página 69](#).

## Punto y coma

Una sentencia ActionScript se termina con un punto y coma (;), como se indica en los ejemplos siguientes:

```

var column = passedDate.getDay();
var row    = 0;

```

Aun cuando omita el punto y coma final, Flash compilará el script correctamente. No obstante, al crear scripts se recomienda utilizar el punto y coma final.

## Paréntesis

Al definir una función, los parámetros deben incluirse entre paréntesis:

```

function myFunction (name, age, reader){
    // el código se escribe aquí
}

```

Al llamar una función, incluya los parámetros que desee pasar a la misma entre paréntesis, como se muestra a continuación:

```

myFunction ("Steve", 10, true);

```

También puede utilizar paréntesis para modificar el orden de precedencia de ActionScript o para hacer más legibles las sentencias de ActionScript. (Véase [“Precedencia y asociatividad de operadores” en la página 48](#).)

También puede utilizar los paréntesis para calcular el resultado de una expresión a la izquierda de un punto en la sintaxis con punto. Por ejemplo, en la sentencia siguiente, los paréntesis hacen que `new Color(this)` se evalúe y se cree un nuevo objeto Color:

```

onClipEvent(enterFrame) {
    (new Color(this)).setRGB(0xffffffff);
}

```

Si no utiliza paréntesis, debe agregar una sentencia para evaluar la expresión:

```

onClipEvent(enterFrame) {
    myColor = new Color(this);
    myColor.setRGB(0xffffffff);
}

```

Puede comprobar si los paréntesis de apertura tienen sus correspondientes paréntesis de cierre en los scripts; véase [“Comprobación de la sintaxis y la puntuación” en la página 69](#).

## Comentarios

Es muy aconsejable añadir notas a los scripts utilizando comentarios. Los comentarios son de gran utilidad para realizar un seguimiento de las acciones que se han deseado llevar a cabo, para proporcionar muestras y, en el caso de que se trabaje en un entorno de trabajo en equipo, para pasar información a otros desarrolladores. Incluso un script sencillo es más fácil de entender si se anotan comentarios durante el proceso de creación.

Para indicar que una línea o parte de una línea es un comentario, iníciela con dos barras inclinadas (//):

```
on(release) {  
    // crear un objeto Date nuevo  
    myDate = new Date();  
    currentMonth = myDate.getMonth();  
    // convertir número de mes en nombre de mes  
    monthName = calcMonth(currentMonth);  
    year = myDate.getFullYear();  
    currentDate = myDate.getDate();  
}
```

Si la función Color de sintaxis está activada (véase [“Resaltado de la sintaxis” en la página 63](#)), los comentarios aparecen en gris de forma predeterminada. Los comentarios pueden tener cualquier longitud sin que ello afecte al tamaño del archivo exportado y no es necesario que sigan las reglas de las palabras clave y de la sintaxis de ActionScript.

Si desea que excluya como si fuera un comentario una parte completa del script, colóquela en un bloque de comentario en lugar de añadir // al principio de cada línea. Esta técnica es más sencilla y resulta útil cuando se desea probar sólo partes del script convirtiendo en comentario gran parte del mismo.

Para crear un bloque de comentario, coloque /\* al principio de las líneas de comentario y \*/ al final. Por ejemplo, cuando se ejecute el siguiente script, no se ejecutará el código incluido en el bloque de comentario.

```
// El código siguiente se ejecuta  
var x:Number = 15;  
var y:Number = 20;  
// El código siguiente no se ejecuta  
/*  
on(release) {  
    // crear un objeto Date nuevo  
    myDate = new Date();  
    currentMonth = myDate.getMonth();  
    // convertir número de mes en nombre de mes  
    monthName = calcMonth(currentMonth);  
    year = myDate.getFullYear();  
    currentDate = myDate.getDate();  
}  
*/  
// El código siguiente se ejecuta  
var name:String = "Me llamo";  
var age:Number = 20;
```

## Palabras clave

ActionScript reserva palabras para usarlas específicamente en su lenguaje de programación, de modo que no se pueden utilizar como identificadores; por ejemplo, nombres de variables, de funciones y de etiquetas. En la siguiente tabla se muestra una lista de las palabras clave de ActionScript:

break	case	class	continue
default	delete	dynamic	else
extends	for	function	get
if	implements	import	in
instanceof	interface	intrinsic	new
private	public	return	set
static	switch	this	typeof
var	void	while	with

## Constantes

Una constante es una propiedad cuyo valor nunca cambia.

Por ejemplo, las constantes BACKSPACE, ENTER, QUOTE, RETURN, SPACE y TAB son propiedades del objeto Key y se refieren a las teclas del teclado. Para comprobar si el usuario está presionando la tecla Intro, utilice la siguiente sentencia:

```
if(Key.getCode() == Key.ENTER) {  
    alert = "¿Está listo para jugar?";  
    controlMC.gotoAndStop(5);  
}
```

## Tipos de datos

Un tipo de datos describe la clase de información que puede contener una variable o un elemento de ActionScript. En Flash se distinguen dos tipos de datos: primitivos y de referencia. Los tipos de datos primitivos (String, Number y Boolean) tienen un valor constante y, por consiguiente, pueden contener el valor real del elemento que representan. Los tipos de datos de referencia (MovieClip y Object) tienen valores que pueden cambiar y, por consiguiente, contienen referencias al valor real del elemento. Las variables que contienen datos de tipo primitivo se comportan de modo diferente en ciertas situaciones que las que contienen datos de tipo referencia. (Véase [“Utilización de variables en un programa” en la página 46.](#)) Hay dos tipos de datos especiales: null y undefined.

En Flash, los objetos incorporados que no sean un tipo de datos primitivo o un tipo de datos de clip de película, como Array o Math, son del tipo de datos Object.

Cada tipo de datos tiene sus propias reglas y está definido en los temas siguientes:

- [“String” en la página 37](#)
- [“Number” en la página 38](#)
- [“Boolean” en la página 38](#)
- [“Object” en la página 38](#)

- “MovieClip” en la página 39
- “Null” en la página 39
- “Undefined” en la página 39

Cuando se depuran scripts, a veces es necesario determinar el tipo de datos de una expresión o variable para entender por qué se comporta de cierta manera. Esto se puede hacer con el operador `typeof` (véase [“Determinación del tipo de datos de un elemento” en la página 39](#)).

Puede convertir un tipo de datos en otro mediante una de las funciones de conversión siguientes: `Array()`, `Boolean()`, `Number()`, `Object()`, `String()`.

## String

Una cadena es una secuencia de caracteres tales como letras, números y signos de puntuación. Las cadenas se introducen en una sentencia de `ActionScript` entre comillas simples o dobles. Las cadenas se tratan como caracteres, no como variables. Por ejemplo, en la siguiente sentencia, `"L7"` es una cadena:

```
favoriteBand = "L7";
```

Puede utilizar el operador de suma (+) para *concatenar* o unir dos cadenas. `ActionScript` trata los espacios del comienzo o del final de una cadena como parte literal de la cadena. La siguiente expresión incluye un espacio después de la coma:

```
greeting = "Bienvenido," + firstName;
```

Para incluir un signo de interrogación en una cadena, ponga delante el carácter de barra inversa (\). A esto se le llama *utilizar una secuencia de escape* en un carácter. Existen otros caracteres que no pueden representarse en `ActionScript`, a menos que se utilice una secuencia de escape especial. La siguiente tabla muestra todos los caracteres de escape de `ActionScript`:

Secuencia de escape	Carácter
<code>\b</code>	Carácter de retroceso (ASCII 8)
<code>\f</code>	Carácter de salto de página (ASCII 12)
<code>\n</code>	Carácter de avance de línea (ASCII 10)
<code>\r</code>	Carácter de retorno de carro (ASCII 13)
<code>\t</code>	Carácter de tabulación (ASCII 9)
<code>\"</code>	Comillas dobles
<code>\'</code>	Comillas simples
<code>\\</code>	Barra inversa
<code>\000 - \377</code>	Byte especificado en octal
<code>\x00 - \xFF</code>	Byte especificado en hexadecimal
<code>\u0000 - \uFFFF</code>	Carácter Unicode de 16 bits especificado en hexadecimal

## Number

El tipo de datos numérico es un número de coma flotante de doble precisión. Puede manipular los números mediante los operadores aritméticos de suma (+), resta (-), multiplicación (\*), división (/), módulo (%), incremento (++) y decremento (--). También puede utilizar métodos de las clases incorporadas Math y Number para manipular los números. En el ejemplo siguiente se utiliza el método `sqrt()` (raíz cuadrada) para devolver la raíz cuadrada del número 100:

```
Math.sqrt(100);
```

Para más información, consulte [“Operadores numéricos” en la página 48](#).

## Boolean

Un valor booleano puede ser `true` o `false`. ActionScript también convierte los valores `true` y `false` en 1 y 0 cuando sea adecuado. Los valores booleanos se usan con mayor frecuencia con los operadores lógicos en sentencias de ActionScript que realizan comparaciones para controlar el flujo de un script. Por ejemplo, en el siguiente script, el archivo SWF se reproduce si la variable `password` es `true`:

```
onClipEvent(enterFrame) {  
    if (userName == true && password == true){  
        play();  
    }  
}
```

Véase [“Utilización de funciones incorporadas” en la página 54](#) y [“Operadores lógicos” en la página 50](#).

## Object

Un objeto es un conjunto de propiedades. Cada propiedad tiene un nombre y un valor. El valor de la propiedad puede ser cualquier tipo de datos de Flash, incluso el tipo de datos de objeto. Esto permite organizar unos objetos dentro de otros, o *anidarlos*. Para especificar objetos y sus propiedades, debe utilizar el operador punto (.). Por ejemplo, en el siguiente código, `hoursWorked` es una propiedad de `weeklyStats`, que a su vez es una propiedad de `employee`:

```
employee.weeklyStats.hoursWorked
```

Puede utilizar los objetos incorporados de ActionScript para acceder y manipular tipos de información específicos. Por ejemplo, el objeto Math tiene métodos que realizan operaciones matemáticas con los números que le pasan. En este ejemplo se utiliza el método `sqrt()`:

```
squareRoot = Math.sqrt(100);
```

El objeto MovieClip de ActionScript tiene métodos que permiten controlar las instancias del símbolo de clip de película en el escenario. En este ejemplo se utilizan los métodos `play()` y `nextFrame()`:

```
mcInstanceName.play();  
mc2InstanceName.nextFrame();
```

También puede crear objetos personalizados para organizar la información en la aplicación Flash. Para añadir interactividad a una aplicación con ActionScript, necesita diversos tipos de información: por ejemplo, puede necesitar un nombre de usuario, la velocidad de una pelota, los nombres de los artículos de un carrito de la compra, el número de fotogramas cargados, el código postal del usuario o la última tecla que se presionó. La creación de objetos personalizados permite organizar esta información en grupos, simplificar la creación de scripts y reutilizarlos.

## MovieClip

Los clips de película son símbolos que pueden reproducir animaciones en una aplicación Flash. Son el único tipo de datos que hace referencia a elementos gráficos. El tipo de datos MovieClip permite controlar los símbolos de clips de película mediante los métodos de la clase MovieClip. Puede llamar a los métodos mediante el operador punto (`.`), como se muestra a continuación:

```
my_mc.startDrag(true);
parent_mc.getURL("http://www.macromedia.com/support/" + product);
```

## Null

El tipo de datos nulo tiene únicamente un valor: `null`. Este valor significa “ningún valor”, es decir, no hay datos. El valor `null` puede utilizarse en distintas situaciones. A continuación se indican algunos ejemplos:

- Para indicar que una variable todavía no ha recibido ningún valor
- Para indicar que una variable ya no contiene ningún valor
- Como valor de retorno de una función, para indicar que la función no ha encontrado ningún valor disponible para devolverlo
- Como parámetro de una función, para indicar que un parámetro se ha omitido

## Undefined

El tipo de datos no definido tiene un valor, `undefined`, y se utiliza para una variable a la que no se ha asignado ningún valor.

## Determinación del tipo de datos de un elemento

Durante la prueba y la depuración de los programas, es posible que detecte problemas que parezcan estar relacionados con los tipos de datos de diferentes elementos. En dichos casos, debe determinar el tipo de datos del elemento. Para ello, utilice el operador `typeof`, como se indica en este ejemplo:

```
trace(typeof(variableName));
```

Para más información sobre la prueba y la depuración, consulte el [Capítulo 3, “Escritura y depuración de scripts”](#), en la [página 57](#).

## Asignación de tipos de datos a elementos

Flash asigna de forma automática tipos de datos a las clases siguientes de elementos del lenguaje, como se indica en la sección siguiente, “[Introducción automática de datos](#)”:

- Variables
- Parámetros que se pasan a una función, método o clase
- Valores devueltos por una función o método
- Objetos creados como subclases de clases existentes

No obstante, también puede asignar de forma explícita tipos de datos a elementos, que pueden ayudarle a impedir o diagnosticar determinados errores en los scripts. Para más información, consulte “[Strict data typing](#)” en la [página 40](#).

## Introducción automática de datos

En Flash, no es necesario definir explícitamente si el elemento contiene un número, una cadena u otro tipo de datos. Flash determina el tipo de datos de un elemento cuando se asigna:

```
var x = 3;
```

En la expresión `var x = 3`, Flash comprueba el valor del elemento que aparece en la parte derecha del operador y determina que pertenece al tipo de datos de número. Una asignación posterior puede cambiar el tipo de `x`; por ejemplo, la sentencia `x="hola"` cambia el tipo de `x` por una cadena. Una variable a la que no se ha asignado ningún valor tiene el tipo `undefined`.

ActionScript convierte los tipos de datos automáticamente cuando lo requiere una expresión. Por ejemplo, cuando se pasa un valor a la acción `trace()`, `trace()` convierte automáticamente el valor en una cadena y la envía al panel Salida. En las expresiones con operadores, ActionScript convierte los tipos de datos según sea necesario; por ejemplo, si el operador `+` se utiliza con una cadena, se espera que el otro operando sea una cadena.

```
"Línea siguiente, número " + 7
```

ActionScript convierte el número 7 en la cadena "7" y la agrega al final de la primera cadena, lo que da como resultado la siguiente cadena:

```
"Línea siguiente, número 7"
```

## Strict data typing

ActionScript 2.0 permite declarar de forma explícita el tipo de objeto de una variable al crearla. Esto recibe el nombre de *strict data typing*. Puesto que las discordancias entre tipos de datos activan errores del compilador, *strict data typing* impide que se asigne un tipo de datos incorrecto a una variable existente. Para asignar un tipo de datos específico a un elemento, especifique el tipo con la palabra clave `var` y la sintaxis de signo de dos puntos posterior:

```
// strict typing de variable u objeto
var x:Number = 7;
var birthday:Date = new Date();

// strict typing de los parámetros
function welcome(firstName:String, age:Number){
}

// strict typing de los parámetros y del valor devuelto
function square(x:Number):Number {
    var squared = x*x;
    return squared;
}
```

Puesto que debe utilizar la palabra clave `var` al realizar un *strict typing* de la variable, no puede realizar un *strict type* de una variable global (consulte ["Ámbito y declaración de variables" en la página 44](#)).

Puede declarar el tipo de datos de objetos basándose en clases incorporadas (`Button`, `Date`, `MovieClip`, etc.) y en las clases e interfaces que haya creado. Por ejemplo, si tiene un archivo llamado `Student.as` en el que define la clase `Student`, puede especificar que los objetos que cree sean del tipo `Student`:

```
var student:Student = new Student();
```

También puede especificar que los objetos sean del tipo `Function` o `Void`.



Strict typing garantiza que no se asigne por despiste un tipo de valor incorrecto a un objeto. Durante la compilación, Flash comprueba errores de discordancia entre los tipos. Por ejemplo, suponga que escribe el código siguiente:

```
// en el archivo de clase Student.as
class Student {
    var status:Boolean; // propiedad de los objetos Student
}

// en un script
var studentMaryLago:Student = new Student();
studentMaryLago.status = "matriculado";
```

Cuando Flash compile este script, se generará un error de “discordancia entre los tipos”.

Otra de las ventajas de strict data typing es que Flash MX 2004 muestra de forma automática sugerencias para el código para los objetos incorporados si se introducen con strict typing. Para más información, consulte [“Strict typing de objetos para activar las sugerencias para el código” en la página 64](#).

Los archivos publicados con ActionScript 1 no respetan las asignaciones de strict data typing durante la fase de compilación. Por eso, al asignar un tipo de valor erróneo a una variable que ha introducido con strict typing no se genera un error de compilador.

```
var x:String = "abc"
x = 12 ; // no hay errores en ActionScript 1, error de discordancia de tipo en
        ActionScript 2
```

La razón de este comportamiento es que al publicar un archivo en ActionScript 1, Flash interpreta una sentencia, como por ejemplo `var x:String = "abc"`, como sintaxis con barras, en lugar de strict typing. (ActionScript 2.0 no admite la sintaxis con barras.) Este comportamiento puede dar como resultado que se asigne un objeto a una variable de tipo incorrecto y que el compilador permita llamadas de método no válidas y que se pasen referencias de propiedades no definidas sin notificarlas.

Por lo tanto, si está implementando strict data typing, asegúrese de que está publicando archivos en ActionScript 2.0.

## Conversión de objetos

ActionScript 2.0 permite convertir un tipo de datos en otro. El operador de conversión que utiliza Flash adopta la forma de una llamada de función y coincide con la *coerción explícita*, tal como se especifica en la propuesta del estándar ECMA-262, edición 4. La conversión permite afirmar que un objeto es de un determinado tipo, de modo que cuando se lleva a cabo la comprobación del tipo, el compilador puede ver que el objeto tiene un conjunto de propiedades que su tipo inicial no contiene. Esto resulta útil, por ejemplo, al repetir una matriz de objetos que pueden ser de tipos diferentes.

En los archivos publicados en Flash Player 7 o una versión posterior, las sentencias de conversión que fallan en tiempo de ejecución devuelven el valor `null`. En los archivos publicados en Flash Player 6, no se implementa soporte en tiempo de ejecución para los fallos de conversión.

La sintaxis para la conversión es *type(item)*, donde se pretende que el compilador se comporte como si el tipo de datos de *item* fuese *type*. La conversión es básicamente una llamada de función, y la llamada de función devuelve *null* si la conversión falla. Si la conversión se lleva a cabo correctamente, la llamada de función devuelve el objeto original. No obstante, el compilador no genera errores de discordancia de tipo al convertir elementos en tipos de datos creados en archivos de clase externos, aunque la conversión falle en tiempo de ejecución.

```
// en Animal.as
class Animal {}

// en Dog.as
class Dog extends Animal { function bark (){} }

// en Cat.as
class Cat extends Animal { function meow (){} }

// en archivo FLA
var spot:Dog = new Dog();
var temp:Ccat = Cat (spot); // afirma que un objeto Dog es de tipo Cat
temp.meow(); // no realiza ninguna acción, pero tampoco genera ningún error de
              compilador
```

En esta situación, ha afirmado al compilador que *temp* es un objeto *Cat* y, por lo tanto, el compilador asume que *temp.meow()* es una sentencia válida. Sin embargo, el compilador no sabe que la conversión fallará (es decir, que ha intentado convertir un objeto *Dog* en un tipo *Cat*), por lo que no se produce ningún error de tiempo de compilación. Si incluye una comprobación en el script para asegurarse de que la conversión se lleva a cabo correctamente, se podrán detectar errores de discordancia de tipo durante el tiempo de ejecución.

```
var spot:Dog = new Dog();
var temp:Ccat = Cat (spot);
trace(temp); // muestra "null" en tiempo de ejecución
```

Puede convertir una expresión en una interfaz. Si la expresión es un objeto que implementa la interfaz, o tiene una clase base que implementa la interfaz, se devuelve el objeto. De lo contrario, se devuelve *null*.

El siguiente ejemplo muestra los resultados de convertir tipos de objetos incorporados. Tal como muestra la primera línea del bloque *with(results)*, una conversión no válida (en este caso, convertir una cadena en un clip de película) devuelve *null*. Las dos últimas líneas muestran que convertir a *null* o *undefined* devuelve *undefined*.

```
var mc:MovieClip;
var arr:Array;
var bool:Boolean;
var num3:Number;
var obj:Object;
var str:String;
_root.createTextField("results",2,100,100,300,300);
with(results){
    text = "type MovieClip : "+(typeof MovieClip(str)); // devuelve null
    text += "\ntype object : "+(typeof Object(str)); // devuelve objeto
    text += "\ntype Array : "+(typeof Array(num3)); // devuelve objeto
    text += "\ntype Boolean : "+(typeof Boolean(mc)); // devuelve valor
    booleano
    text += "\ntype String : "+(typeof String(mc)); // devuelve cadena
    text += "\ntype Number : "+(typeof Number(obj)); // devuelve número
    text += "\ntype Function : "+(typeof Function(mc)); // devuelve objeto
    text += "\ntype null : "+(typeof null(arr)); // devuelve undefined
```

```

text += "\ntype undefined : "+(typeof undefined(obj)); // devuelve undefined
}
//Visualiza el panel Salida
type MovieClip : null
type object : objeto
type Array : objeto
type Boolean : booleano
type String : cadena
type Number : número
type Function : objeto
type null : undefined
type undefined : undefined

```

No puede anular los tipos de datos primitivos como Boolean, Date y Number con un operador de conversión del mismo nombre.

## Variables

Una *variable* es un contenedor que almacena información. El contenedor en sí es siempre el mismo, pero el contenido puede cambiar. La modificación del valor de una variable a medida que se reproduce el archivo SWF permite registrar y guardar información sobre las acciones del usuario, registrar valores que se modifican conforme se reproduce el archivo SWF o comprobar si una determinada condición es true o false.

Cuando se define una variable por primera vez, se recomienda asignarle un valor conocido. Esta acción se conoce como *inicializar una variable* y suele realizarse en el primer fotograma del archivo SWF. Inicializar las variables permite realizar un seguimiento y comparar el valor de la variable a medida que se reproduce el archivo SWF.

Las variables pueden contener cualquier tipo de datos (véase [“Tipos de datos” en la página 36](#)). El tipo de datos que contiene una variable afecta al modo en el que cambia el valor de la variable cuando se asigna en un script.

El tipo de información que habitualmente se guarda en una variable es una URL, un nombre de usuario, el resultado de una operación matemática, el número de veces que ocurre un evento o si se ha hecho clic en un botón. Cada instancia de clip de película y archivo SWF tiene un conjunto de variables, cada una de ellas con su propio valor independiente del de otras variables definidas en otros archivos SWF o clips de película.

Para comprobar el valor de una variable, utilice la acción `trace()` para enviar el valor al panel Salida. Por ejemplo, `trace(hoursWorked)` envía el valor de la variable `hoursWorked` al panel Salida en modo de prueba. También puede comprobar y establecer los valores de las variables en el Depurador en modo de prueba. Para más información, consulte [“Utilización de la sentencia trace” en la página 82](#) y [“Visualización y modificación de variables” en la página 74](#).

### Asignación de un nombre a una variable

El nombre de una variable debe seguir estas reglas:

- Debe ser un identificador (véase [“Terminología” en la página 28](#)).
- No puede ser una palabra clave ni un literal de ActionScript, como `true`, `false`, `null` o `undefined`.
- Debe ser exclusivo en su ámbito (véase [“Ámbito y declaración de variables” en la página 44](#)).

Asimismo, no debe utilizar ningún elemento del lenguaje ActionScript como nombre de variable; en caso de hacerlo, se pueden producir errores de sintaxis o resultados inesperados. Por ejemplo, si denomina `String` a una variable y, a continuación, intenta crear un objeto `String` mediante `newString()`, el nuevo objeto será `undefined`.

```
hello_str = new String();  
trace(hello_str.length); // devuelve 0
```

```
String = "hello"; // Asignar a una variable el mismo nombre que una clase  
                incorporada  
hello_str = new String();  
trace(hello_str.length); // devuelve undefined
```

El editor de ActionScript admite las sugerencias de código para las clases incorporadas y para las variables basadas en dichas clases. Si desea que Flash proporcione sugerencias de código para un tipo de objeto determinado asignado a una variable, puede introducir una variable con `strict typing` o denominar la variable mediante un sufijo determinado.

Por ejemplo, suponga que escribe el código siguiente:

```
var members:Array = new Array();  
members.
```

En cuanto escriba el punto (`.`), Flash mostrará una lista de métodos y propiedades disponibles para los objetos `Array`. Para más información, consulte [“Escritura de código que activa las sugerencias para el código” en la página 64](#).

## Ámbito y declaración de variables

El *ámbito* de una variable se refiere al área en la que se conoce la variable y se puede hacer referencia a ella. ActionScript contiene tres tipos de ámbito de variable:

- Las [Variables locales](#) están disponibles en el cuerpo de la función en la que se han declarado (incluidas entre llaves).
- Las [Variables de línea de tiempo](#) están disponibles para cualquier script de dicha línea de tiempo.
- Las funciones y las [Variables globales](#) son visibles para todas las líneas de tiempo y todos los ámbitos del documento.

**Nota:** las clases de ActionScript 2.0 que crea el usuario son compatibles con los ámbitos de variable público, privado y estático. Para más información, consulte [“Control del acceso de miembros” en la página 170](#) y [“Creación de miembros de clase” en la página 171](#).

## Variables locales

Para declarar una variable local, utilice la sentencia `var` en el cuerpo de una función. El ámbito de una variable local es el bloque y caduca al final del bloque. Una variable local que no esté declarada dentro de un bloque caduca al final de su script.

Por ejemplo, las variables `i` y `j` a menudo se utilizan como contadores de reproducciones indefinidas. En el siguiente ejemplo, `i` se utiliza como variable local; solamente existe dentro de la función `makeDays()`:

```
function makeDays() {
  var i;
  for( i = 0; i < monthArray[month]; i++ ) {

    _root.Days.attachMovie( "DayDisplay", i, i + 2000 );

    _root.Days[i].num = i + 1;
    _root.Days[i]._x = column * _root.Days[i]._width;
    _root.Days[i]._y = row * _root.Days[i]._height;

    column = column + 1;

    if (column == 7 ) {
      column = 0;
      row = row + 1;
    }
  }
}
```

Las variables locales también pueden evitar conflictos de nombres, que pueden originar errores en la aplicación. Por ejemplo, si utiliza `name` como variable local, podría utilizarla para almacenar un nombre de usuario en un contexto y un nombre de una instancia de clip de película en otro; puesto que estas variables se ejecutarán en ámbitos independientes, ello no causará ningún conflicto.

Es usual y recomendable utilizar variables locales en el cuerpo de una función de modo que ésta pueda actuar como un segmento de código independiente. Una variable local solamente puede cambiar dentro de su propio bloque de código. Si la expresión contenida en una función utiliza una variable global, un elemento externo a la función podría modificar su valor, lo cual cambiaría la función.

Puede asignar un tipo de datos a una variable local al definirla, lo cual impide asignar el tipo de datos incorrecto a una variable existente. Para más información, consulte [“Strict data typing” en la página 40](#).

## Variables de línea de tiempo

Las variables de línea de tiempo están disponibles para cualquier script de dicha línea de tiempo. Para declarar variables de línea de tiempo, debe inicializarlas en cualquier fotograma de la línea de tiempo. Asegúrese de inicializar la variable antes de intentar acceder a la misma en un script. Por ejemplo, si coloca el código `var x = 10;` en el fotograma 20, un script adjunto a un fotograma anterior al fotograma 20 no podrá acceder a dicha variable.

## Variables globales

Las funciones y las variables globales son visibles para todas las líneas de tiempo y todos los ámbitos del documento. Para crear una variable de ámbito global, escriba el identificador `_global` delante del nombre de la variable y no utilice la sintaxis `var =`. Por ejemplo, el código siguiente crea la variable global `myName`:

```
var _global.myName = "George"; // error de sintaxis
_global.myName = "George";
```

Sin embargo, si inicializa una variable local con el mismo nombre que la variable global, no tendrá acceso a la variable global mientras se encuentra en el ámbito de la variable local:

```
_global.counter = 100;
counter++;
trace(counter); // muestra 101
function count(){
    for( var counter = 0; counter <= 10 ; counter++ ) {
        trace(counter); // muestra de 0 a 10
    }
}
count();
counter++;
trace(counter); // muestra 102
```

## Utilización de variables en un programa

Debe declarar una variable en un script antes de poder utilizarla en una expresión. Si se utiliza una variable no declarada, como en el ejemplo siguiente, el valor de la variable será NaN o undefined y el script podría producir resultados imprevistos:

```
var squared = x*x;
trace(squared); // NaN
var x = 6;
```

En el ejemplo siguiente, la sentencia que declara la variable `x` debe ir al principio, de modo que `squared` pueda sustituirse por un valor:

```
var x = 6;
var squared = x*x;
trace(squared); // 36
```

Ocorre lo mismo al pasar una variable no definida a un método o a una función:

```
getURL(myWebSite); // sin acción
var myWebSite = "http://www.macromedia.com";

var myWebSite = "http://www.macromedia.com";
getURL(myWebSite); // el navegador mostrará www.macromedia.com
```

El valor de una variable puede cambiar muchas veces en un script. El tipo de datos que contiene la variable afecta cómo y cuándo cambia la misma. Las variables que contienen datos de tipo primitivo, como son las cadenas y números, se pasan por valor. Esto quiere decir que se pasa a la variable el contenido real de la misma.

En el siguiente ejemplo, `x` está establecida en 15 y ese valor se copia en `y`. Cuando `x` pasa a ser 30 en la línea 3, el valor de `y` sigue siendo 15 ya que `y` no busca en `x` su valor, sino que contiene el valor de `x` que ha recibido en la línea 2.

```
var x = 15;
var y = x;
var x = 30;
```

En otro caso, por ejemplo, la variable `inValue` contiene un valor primitivo, 3, de modo que el valor real se pasa a la función `sqrt()` y el valor que devuelve es 9:

```
function sqrt(x){
    return x * x;
}

var inValue = 3;
var out = sqrt(inValue);
```

El valor de la variable `inValue` no cambia.

Los datos de tipo objeto pueden contener una cantidad de información tan compleja y grande que una variable cuyo contenido es este tipo de dato no contiene el valor real, sino una referencia al valor. Esta referencia es como un alias que apunta al contenido de la variable. Cuando la variable necesita conocer su valor, la referencia solicita el contenido y devuelve la respuesta sin transferir el valor a la variable.

A continuación se muestra un ejemplo en el que se pasa un valor por referencia:

```
var myArray = ["tom", "josie"];
var newArray = myArray;
myArray[1] = "jack";
trace(newArray);
```

El código anterior crea un objeto Array denominado `myArray` que tiene dos elementos. Se crea la variable `newArray` y se pasa una referencia a `myArray`. Cuando el segundo elemento de `myArray` cambia, afectará a cualquier variable que haga referencia al mismo. La acción `trace()` envía `tom`, `jack` al panel Salida.

En el siguiente ejemplo, `myArray` contiene un objeto Array, de modo que se pasa a la función `zeroArray()` por referencia. La función `zeroArray()` cambia el contenido de la matriz en `myArray`.

```
function zeroArray (theArray){
    var i;
    for (i=0; i < theArray.length; i++) {
        theArray[i] = 0;
    }
}

var myArray = new Array();
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;
zeroArray(myArray);
```

La función `zeroArray()` acepta un objeto Array como parámetro y define el valor 0 para todos los elementos de dicha matriz. Puede modificar la matriz porque se pasa por referencia.

## Utilización de operadores para manipular los valores de las expresiones

Una expresión es cualquier sentencia para la que Flash puede calcular el resultado y que devuelve un valor. Puede crear una expresión combinando operadores y valores, o bien llamando a una función.

Los operadores son caracteres que especifican cómo combinar, comparar o modificar los valores de una expresión. Los elementos sobre los que el operador actúa se denominan *operandos*. Por ejemplo, en la sentencia siguiente, el operador `+` agrega el valor de un literal numérico al valor de la variable `foo`; `foo` y `3` son los operandos:

```
foo + 3
```

En esta sección se describen las reglas generales para los tipos comunes de operadores, precedencia del operador y asociatividad de operadores. Para obtener información detallada sobre cada uno de los operadores mencionados, así como sobre otros operadores especiales que no se incluyen en estas categorías, consulte las entradas del [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Precedencia y asociatividad de operadores

Cuando se utilizan dos o más operadores en la misma sentencia, algunos operadores tienen prioridad sobre otros. ActionScript sigue una jerarquía muy estricta para determinar qué operadores deben ejecutarse en primer lugar. Por ejemplo, la multiplicación siempre se realiza antes que la suma; sin embargo, los elementos entre paréntesis tienen precedencia sobre la multiplicación. De modo que, sin paréntesis, ActionScript realiza primero la multiplicación en el ejemplo siguiente:

```
total = 2 + 4 * 3;
```

El resultado es 14.

Pero cuando la operación de suma está entre paréntesis, ActionScript realiza la suma en primer lugar:

```
total = (2 + 4) * 3;
```

El resultado es 18.

Cuando dos o más operadores comparten la misma precedencia, su asociatividad determina el orden en el que se llevan a cabo. La asociatividad puede ser de izquierda a derecha o de derecha a izquierda. Por ejemplo, el operador de multiplicación tiene una asociatividad de izquierda a derecha; por consiguiente, las dos sentencias siguientes son equivalentes:

```
total = 2 * 3 * 4;  
total = (2 * 3) * 4;
```

En el [Apéndice B, “Precedencia y asociatividad de los operadores”, en la página 789](#), hallará una tabla con todos los operadores, su precedencia y su asociatividad.

## Operadores numéricos

Los operadores numéricos realizan sumas, restas, multiplicaciones, divisiones y otras operaciones aritméticas.

El uso más común del operador de incremento es `i++` en lugar del uso más detallado `i = i+1`. Un operador de incremento puede especificarse antes o después de un operando. En el ejemplo siguiente, primero se aumenta el valor de `age` y luego se verifica frente a 30:

```
if (++age >= 30)
```

En el ejemplo siguiente, se aumenta el valor de `age` tras realizar la comprobación:

```
if (age++ >= 30)
```



En la tabla siguiente se enumeran los operadores numéricos de ActionScript:

Operador	Operación realizada
+	Suma
*	Multiplicación
/	División
%	Módulo (resto de la división)
-	Resta
++	Incremento
--	Decremento

## Operadores de comparación

Los operadores de comparación comparan los valores de las expresiones y devuelven un valor booleano (`true` o `false`). Estos operadores se utilizan con mayor frecuencia en las reproducciones indefinidas y en las sentencias condicionales. En el ejemplo siguiente, si la variable `score` es 100, se carga un archivo SWF determinado; de lo contrario, se carga un archivo SWF diferente:

```
if (score > 100){  
    loadMovieNum("winner.swf", 5);  
} else {  
    loadMovieNum("loser.swf", 5);  
}
```

En la tabla siguiente se enumeran los operadores de comparación de ActionScript:

Operador	Operación realizada
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

## Operadores de cadena

El operador `+` tiene un efecto especial en las cadenas: concatena los dos operandos de cadena. Por ejemplo, la sentencia siguiente agrega `"¡Felicidades, "` a `"David!"`:

```
"¡Felicidades, " + "David!"
```

El resultado es `"Felicidades, David!"`. Si sólo uno de los operandos del operador `+` es una cadena, Flash convierte el otro operando en una cadena.

Los operadores de comparación `>`, `>=`, `<` y `<=` también tienen un efecto especial cuando operan sobre cadenas. Estos operadores comparan dos cadenas para determinar cuál es la primera de acuerdo a su orden alfabético. Los operadores de comparación solamente comparan cadenas si ambos operandos son cadenas. Si solamente uno de los operandos es una cadena, ActionScript convierte ambos operandos en números y realiza una comparación numérica.

## Operadores lógicos

Los operadores lógicos comparan los valores booleanos (`true` y `false`) y devuelven un tercer valor booleano. Por ejemplo, si ambos operandos son `true`, el operador lógico AND (`&&`) devuelve `true`. Si uno o ambos operandos son `true`, el operador lógico OR (`||`) devuelve `true`. Los operadores lógicos se utilizan a menudo con los operadores de comparación para determinar la condición de una acción `if`. Por ejemplo, en el siguiente script, si ambas expresiones son verdaderas, la acción `if` se ejecutará:

```
if (i > 10 && _framesloaded > 50){
    play();
}
```

En la tabla siguiente se enumeran los operadores lógicos de ActionScript:

Operador	Operación realizada
<code>&amp;&amp;</code>	AND lógico
<code>  </code>	OR lógico
<code>!</code>	NOT lógico

## Operadores en modo bit

Los operadores en modo bit manipulan internamente los números de coma flotante para cambiarlos por enteros de 32 bits. La operación exacta realizada depende del operador, pero todas las operaciones en modo bit calculan el resultado de cada dígito binario (bit) del entero de 32 bits de forma individual para calcular un nuevo valor.

En la tabla siguiente se enumeran los operadores en modo bit de ActionScript:

Operador	Operación realizada
<code>&amp;</code>	AND en modo bit
<code> </code>	OR en modo bit
<code>^</code>	XOR en modo bit
<code>~</code>	NOT en modo bit
<code>&lt;&lt;</code>	Desplazamiento a la izquierda
<code>&gt;&gt;</code>	Desplazamiento a la derecha
<code>&gt;&gt;&gt;</code>	Desplazamiento a la derecha con relleno con ceros

## Operadores de igualdad

Puede utilizar el operador de igualdad (`==`) para determinar si los valores o las identidades de dos operandos son iguales. Esta comparación devuelve un valor booleano (`true` o `false`). Si los operandos son cadenas, números o valores booleanos, se comparan por su valor. Si los operandos son objetos o matrices, se comparan por referencia.

Es un error muy común utilizar el operador de asignación para comprobar la igualdad. En el código siguiente, por ejemplo, se compara `x` con 2:

```
if (x == 2)
```

En el mismo ejemplo, la expresión `x = 2` no es correcta porque no compara los operandos, sino que asigna el valor 2 a la variable `x`.

El operador de igualdad estricta (`===`) es igual al operador de igualdad excepto por una diferencia importante: el operador de igualdad estricta no lleva a cabo la conversión de tipo. Si los dos operandos son de distinto tipo, el operador de igualdad estricta devuelve `false`. El operador de desigualdad estricta (`!==`) devuelve la inversión del operador de igualdad estricta.

En la tabla siguiente se enumeran los operadores de igualdad de `ActionScript`:

Operador	Operación realizada
<code>==</code>	Igualdad
<code>===</code>	Igualdad estricta
<code>!=</code>	Desigualdad
<code>!==</code>	Desigualdad estricta

## Operadores de asignación

Puede utilizar el operador de asignación (`=`) para asignar un valor a una variable, como se muestra en el ejemplo siguiente:

```
var password = "Sk8tEr";
```

También puede utilizar el operador de asignación para asignar valores a diversas variables en la misma expresión. En la siguiente sentencia, el valor de `a` se asigna a las variables `b`, `c` y `d`:

```
a = b = c = d;
```

También puede utilizar operadores de asignación compuestos para combinar operaciones. Los operadores compuestos actúan sobre los dos operandos y después asignan un nuevo valor al primer operando. Por ejemplo, las dos sentencias siguientes son equivalentes:

```
x += 15;  
x = x + 15;
```

El operador de asignación también puede utilizarse dentro de una expresión, como se muestra en el ejemplo siguiente:

```
// Si el sabor no es vainilla, generar un mensaje.  
if ((flavor = getIceCreamFlavor()) != "vainilla") {  
    trace("El sabor " + flavor + ", no era vainilla.");  
}
```

Este código es equivalente al código ligeramente más detallado que se muestra a continuación:

```
flavor = getIceCreamFlavor();  
if (flavor != "vainilla") {  
    trace("El sabor " + flavor + ", no era vainilla.");  
}
```

En la tabla siguiente se enumeran los operadores de asignación de ActionScript:

Operador	Operación realizada
=	Asignación
+=	Suma y asignación
-=	Resta y asignación
*=	Multipliación y asignación
%=	Módulo y asignación
/=	División y asignación
<<=	Desplazamiento a la izquierda en modo bit y asignación
>>=	Desplazamiento a la derecha en modo bit y asignación
>>>=	Desplazamiento a la derecha rellenando con ceros y asignación
^=	XOR en modo bit y asignación
=	OR en modo bit y asignación
&=	AND en modo bit y asignación

## Operadores de punto y de acceso a una matriz

Puede utilizar el operador de punto (.) y el operador de acceso a una matriz ([ ]) para acceder a cualquiera de las propiedades del objeto incorporadas o personalizadas de ActionScript, incluidas las de un clip de película.

El operador de punto utiliza el nombre de un objeto a su lado izquierdo y el nombre de una propiedad o variable a su lado derecho. El nombre de la propiedad o la variable no puede ser una cadena o una variable que dé como resultado una cadena, sino que debe ser un identificador. En los ejemplos siguientes se utiliza el operador de punto:

```
year.month = "Junio";  
year.month.day = 9;
```

El operador de punto y el operador de acceso a una matriz se comportan de la misma manera, pero el operador de punto toma un identificador como su propiedad, mientras que el operador de acceso a una matriz comprueba su contenido respecto a un nombre y después accede al valor de esa propiedad con nombre. Por ejemplo, las expresiones siguientes acceden a la misma variable `velocity` en el clip de película `rocket`:

```
rocket.velocity;  
rocket["velocity"];
```

Puede utilizar el operador de acceso a una matriz para establecer y recuperar dinámicamente nombres de instancias y variables. Por ejemplo, en el código que se muestra a continuación, se calcula el resultado de la expresión dentro del operador [ ] y el resultado se utiliza como nombre de la variable que se va a recuperar del clip de película `name`:

```
name["mc" + i ]
```

También puede utilizar la función `eval()`, como se muestra a continuación:

```
eval("mc" + i)
```

El operador de acceso a una matriz también puede utilizarse al lado izquierdo de una sentencia de asignación. Esto permite establecer dinámicamente los nombres de instancia, de variable y de objeto, como se muestra en el ejemplo siguiente:

```
name[index] = "Gary";
```

Para crear matrices multidimensionales en ActionScript, debe construir una matriz, cuyos elementos son también matrices. Para acceder a los elementos de una matriz multidimensional, puede anidar el operador de acceso a la matriz en sí mismo, como se muestra en el ejemplo siguiente:

```
var chessboard = new Array();
for (var i=0; i<8; i++) {
    chessboard.push(new Array(8));
}
function getContentsOfSquare(row, column){
    chessboard[row][column];
}
```

Puede comprobar si los operadores de apertura tienen sus correspondientes operadores de cierre [] en los scripts; véase [“Comprobación de la sintaxis y la puntuación” en la página 69](#).

## Especificación de la ruta de un objeto

Para utilizar una acción a fin de controlar un clip de película o un archivo SWF que ha sido cargado, debe especificar su *ruta de destino*, esto es, su nombre y su dirección.

En ActionScript se identifica a un clip de película por su nombre de instancia. Por ejemplo, en la siguiente sentencia, la propiedad `_alpha` del clip de película denominado `star` está establecida en un 50% de visibilidad:

```
star._alpha = 50;
```

### Para asignar un nombre de instancia a un clip de película:

- 1 Seleccione el clip de película en el escenario.
- 2 Introduzca un nombre de instancia en el inspector de propiedades.

### Para identificar un archivo SWF cargado:

- Utilice `_levelX`, donde *X* es el número de nivel especificado en la acción `loadMovie()` que ha cargado el archivo SWF.

Por ejemplo, un archivo SWF que se ha cargado en el nivel 5 tiene la ruta de destino `_level5`. En el ejemplo siguiente, se carga un archivo SWF en el nivel 5 y su visibilidad se establece en `false`:

```
onClipEvent (load) {
    loadMovieNum("myMovie.swf", 5);
}
onClipEvent (enterFrame) {
    _level5._visible = false;
}
```

### Para introducir la ruta de destino de un archivo SWF:

- En el panel Acciones (Ventana > Paneles de desarrollo > Acciones), haga clic en el botón Insertar ruta de destino y seleccione uno de los clips de película de la lista que aparece.

Para obtener más información sobre rutas de destino, consulte “Rutas de destino absolutas y relativas” en el apartado Utilización de Flash de la Ayuda.

## Utilización de funciones incorporadas

Una función es un bloque de código de ActionScript que puede volver a utilizarse en cualquier parte de un archivo SWF. Si pasa valores a una función como parámetros, la función actuará sobre esos valores. Una función también puede devolver valores.

Flash incorpora funciones que permiten acceder a un tipo determinado de información y realizar determinadas tareas, tales como obtener el número de versión del Flash Player que alberga el archivo SWF (`getVersion()`). Las funciones que pertenecen a un objeto se denominan *métodos*. Las funciones que no pertenecen a un objeto se denominan *funciones de nivel superior* y se encuentran en la categoría Funciones del panel Acciones.

Cada función tiene sus propias características y algunas necesitan que se le pasen determinados valores. Si se pasan a la función más argumentos de los que necesita, se pasarán por alto los valores. Si no se le pasa un argumento necesario, los argumentos vacíos se asignarán al tipo de datos `undefined`, que puede causar errores cuando exporte un script. Para llamar a una función, ésta debe encontrarse en un fotograma al que haya llegado la cabeza lectora.

Para llamar a una función, utilice el nombre de función e indique los parámetros necesarios:

```
isNaN(someVar);  
getTimer();  
eval("someVar");
```

Para más información sobre cada función, consulte las entradas correspondientes del [Capítulo 12](#), “Diccionario de ActionScript”, en la [página 213](#).

## Creación de funciones

Puede definir funciones para que éstas ejecuten una serie de sentencias en función de los valores que se le han pasado. Sus funciones también pueden devolver valores. Una vez que se ha definido una función, podrá llamarla desde cualquier línea de tiempo, incluida la línea de tiempo de un archivo SWF que se ha cargado.

Una función correctamente escrita puede compararse a una "caja negra". Si se incluyen comentarios acerca de sus entradas, salidas y su finalidad, no es necesario que el usuario de la función comprenda exactamente cómo funciona internamente.

## Definición de una función

Las funciones, al igual que las variables, están asociadas a la línea de tiempo del clip de película que las define y, para llamarlas, debe utilizarse una ruta de destino. Al igual que con las variables, puede utilizarse el identificador `_global` para declarar una función global que está disponible para todas las líneas de tiempo sin utilizar una ruta de destino. Para definir una función global, debe anteponer el identificador `_global` al nombre de la función, como se muestra en el ejemplo siguiente:

```
_global.myFunction = function (x) {  
    return (x*2)+3;  
}
```

Para definir una función de línea de tiempo, utilice la acción `function` seguida del nombre de la función, los parámetros que va a pasar a la función y las sentencias de ActionScript que indican qué acción lleva a cabo la función.

En el ejemplo siguiente se muestra una función denominada `areaOfCircle` con el parámetro `radius`:

```
function areaOfCircle(radius) {  
    return Math.PI * radius * radius;  
}
```

También puede definir una función creando un *literal de función*, es decir, una función sin nombre que se declara en una expresión, no en una sentencia. Puede utilizar la expresión literal de una función para definir una función, devolver su valor y asignarlo a una variable en una expresión, como se muestra en el ejemplo siguiente:

```
area = (function() {return Math.PI * radius *radius;})(5);
```

Cuando se vuelve a definir una función, la nueva definición sustituye a la definición anterior.

## Paso de parámetros a una función

Los parámetros son los elementos sobre los que una función ejecuta su código (en este manual, los términos *parámetro* y *argumento* pueden utilizarse indistintamente). Por ejemplo, la función siguiente toma los parámetros `initials` y `finalScore`:

```
function fillOutScorecard(initials, finalScore) {  
    scorecard.display = initials;  
    scorecard.score = finalScore;  
}
```

Cuando se llama a la función, los parámetros que ésta necesita deberán pasarse a la misma. La función sustituye los valores pasados por los parámetros en la definición de la función. En este ejemplo, `scorecard` es el nombre de instancia de un clip de película; `display` y `score` son campos de introducción de texto en la instancia. La siguiente llamada de función asigna el valor "JEB" a la variable `display` y el valor 45000 a la variable `score`:

```
fillOutScorecard("JEB", 45000);
```

El parámetro `initials` de la función `fillOutScorecard()` es similar a una variable local; existe mientras se llama a la función y deja de existir cuando se cierra la función. Si omite los parámetros durante la llamada a una función, los parámetros omitidos se pasan como `undefined`. Si introduce parámetros adicionales en una llamada de función que no son necesarios para la declaración de la función, éstos se pasarán por alto.

## Utilización de variables en una función

Las variables locales son herramientas muy valiosas para organizar el código y hacer que sea más fácil de entender. Cuando una función utiliza variables locales, la función puede ocultar sus variables de todos los demás scripts del archivo SWF; las variables locales tienen su ámbito en el cuerpo de la función y desaparecen cuando se sale de la función. Cualquier parámetro que se pase a una función también se tratará como una variable local.

En una función también pueden utilizarse variables globales y regulares. No obstante, si modifica las variables globales o regulares, es aconsejable utilizar comentarios de script para documentar los cambios.

## Devolución de valores de una función

Utilice la sentencia `return` para devolver valores desde las funciones. La sentencia `return` detiene la función y la sustituye por el valor de la acción `return`. El uso de la sentencia `return` se rige por las siguientes reglas:

- Si especifica un tipo de devolución que no sea `void` para una función, debe incluir una sentencia `return` en la función.
- Si especifica un tipo de devolución `void`, no debe incluir una sentencia `return`.
- Si no especifica ningún tipo de devolución, la inclusión de una sentencia `return` es opcional. Si no incluye ninguna, se devolverá una cadena vacía.

Por ejemplo, la siguiente función devuelve el cuadrado del parámetro `x` y especifica que el valor devuelto debe ser un número:

```
function sqr(x):Number {  
    return x * x;  
}
```

Algunas funciones realizan una serie de tareas sin devolver un valor. Por ejemplo, la siguiente función inicializa una serie de variables globales:

```
function initialize() {  
    boat_x = _global.boat._x;  
    boat_y = _global.boat._y;  
    car_x = _global.car._x;  
    car_y = _global.car._y;  
}
```

## Llamada a una función definida por el usuario

Puede utilizar una ruta de destino para llamar a una función en cualquier línea de tiempo y desde cualquier línea de tiempo, inclusive desde la línea de tiempo de un archivo SWF cargado. Si se ha declarado una función mediante el identificador `_global`, no es necesario utilizar ninguna ruta de destino para llamar a dicha función.

Para llamar a una función, indique la ruta de destino del nombre de la función, si es necesario, y pase los parámetros necesarios entre paréntesis. Por ejemplo, la siguiente sentencia invoca la función `sqr()` del clip de película `MathLib` en la línea de tiempo principal, le pasa el parámetro `3` y almacena el resultado en la variable `temp`:

```
var temp = _root.MathLib.sqr(3);
```

En el ejemplo siguiente se utiliza una ruta absoluta para llamar a la función `initialize()` que se ha definido en la línea de tiempo principal y que no requiere ningún parámetro:

```
_root.initialize();
```

En el ejemplo siguiente se utiliza una ruta relativa para llamar a la función `list()` que se ha definido en el clip de película `functionsClip`:

```
_parent.functionsClip.list(6);
```



# CAPÍTULO 3

## Escritura y depuración de scripts

En Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004, puede escribir scripts que están incorporados en el archivo FLA o almacenados externamente en el equipo. Si está escribiendo archivos de clase ActionScript 2.0, debe almacenar cada clase como un archivo externo que tenga el mismo nombre que la clase. Para escribir scripts incorporados, utilice el panel Acciones y asocie la acción a un botón o clip de película, o a un fotograma de la línea de tiempo (véase [“Control de la ejecución de ActionScript” en la página 57](#)). Para escribir archivos de script externos, puede utilizar cualquier editor de texto o de código. En Flash MX Professional, también puede utilizar la ventana Script incorporada. Para más información, consulte [“Utilización del panel Acciones y la ventana Script” en la página 60](#).

Al utilizar el editor de ActionScript, también puede comprobar errores de sintaxis, aplicar automáticamente formato al código y utilizar las sugerencias para el código para completar la sintaxis. Además, la función de balance de puntuación ayuda a establecer los pares de paréntesis, corchetes o llaves. Para más información, consulte [“Utilización del editor de ActionScript” en la página 63](#).

A medida que trabaje con un documento, pruébelo a menudo para asegurarse de que se reproduce sin problemas y tal como se espera. Puede utilizar el Visor de anchos de banda para simular cómo aparecerá el documento con distintas velocidades de conexión (véase [“Comprobación del rendimiento de descarga de documentos” en el apartado Utilización de Flash de la Ayuda](#)). Para probar los scripts, utilice una versión especial de depuración de Flash Player que le ayudará a resolver los problemas. Si utiliza técnicas de programación adecuadas con el código de ActionScript, será más fácil resolver los problemas de scripts cuando se produzca un comportamiento inesperado. Para más información, consulte [“Depuración de los scripts” en la página 71](#).

### Control de la ejecución de ActionScript

Al escribir scripts se utiliza el panel Acciones para asociar el script a un fotograma de una línea de tiempo o para asociarlo a un botón o a un clip de película del escenario. Los scripts asociados a un fotograma se *ejecutan* cuando la cabeza lectora entra en el fotograma en cuestión. No obstante, los scripts asociados al primer fotograma de un archivo SWF pueden comportarse de manera diferente de los asociados a fotogramas posteriores, porque el primer fotograma de un archivo SWF se va generando gradualmente (los objetos van apareciendo en el escenario a medida que se descargan en Flash Player), lo que puede afectar a la ejecución de las acciones. Todos los fotogramas que aparecen después del primer fotograma se generan de una vez, cuando todos los objetos del fotograma están disponibles.

Los scripts asociados a los clips de película o a los botones se ejecutan cuando se produce un evento. Un *evento* representa algún tipo de actividad en el archivo SWF, como el movimiento del ratón, la pulsación de una tecla o la carga de un clip de película. Puede utilizar ActionScript para saber cuándo se producen dichos eventos y ejecutar scripts determinados en función del evento. Para más información, consulte [Capítulo 4, “Gestión de eventos”, en la página 85](#).

Para realizar una acción según si existe una condición o no, o para repetir una acción, puede utilizar las sentencias `if`, `else`, `else if`, `for`, `while`, `do while`, `for..in` o `switch`, descritas brevemente en el resto de esta sección.

## Comprobación de una condición

Las sentencias que comprueban si el valor de una condición es `true` o `false` comienzan con el término `if`. Si la condición se cumple, ActionScript ejecuta la sentencia que aparece a continuación de la misma. Si la condición no se cumple, ActionScript salta a la siguiente sentencia fuera del bloque de código.

Para optimizar el rendimiento de su código, compruebe primero las condiciones más probables.

Las sentencias siguientes comprueban tres condiciones. El término `else if` especifica comprobaciones alternativas que se deberán llevar a cabo si las condiciones anteriores son falsas.

```
if (password == null || email == null) {
    gotoAndStop("reject");
} else if (password == userID){
    gotoAndPlay("startMovie");
}
```

Si desea comprobar una de varias condiciones, puede utilizar la sentencia `switch` en lugar de varias sentencias `else if`.

## Repetición de una acción

ActionScript puede repetir una acción un número especificado de veces o mientras se dé una condición. Utilice las acciones `while`, `do..while`, `for` y `for..in` para crear reproducciones indefinidas.

**Para repetir una acción mientras se cumpla una condición:**

- Utilice la sentencia `while`.

Un bucle `while` calcula el resultado de una expresión y ejecuta el código en el cuerpo del bucle si la expresión es `true`. Tras ejecutar cada una de las sentencias del bucle, se calcula de nuevo el resultado de la expresión. En el siguiente ejemplo, la reproducción se ejecuta cuatro veces:

```
i = 4;
while (var i > 0) {
    my_mc.duplicateMovieClip("newMC" + i, i );
    i--;
}
```

Puede utilizar la sentencia `do...while` para crear el mismo tipo de reproducción que `while`. En un bucle `do...while`, el resultado de la expresión se calcula al final del bloque de código de modo que el bucle siempre se ejecuta al menos una vez, como se muestra en el ejemplo siguiente:

```
i = 4;
do {
    my_mc.duplicateMovieClip("newMC" + i, i );
    i--;
} while (var i > 0);
```

#### Para repetir una acción mediante un contador incorporado:

- Utilice la sentencia `for`.

La mayoría de las reproducciones indefinidas utilizan algún tipo de contador para controlar cuántas veces se ejecutan. Cada ejecución se denomina *repetición*. Puede declarar una variable y escribir una sentencia que incremente o disminuya el valor de la variable cada vez que se ejecute la reproducción. En la acción `for`, el contador y la sentencia que incrementa el contador son parte de la acción. En el ejemplo siguiente, la primera expresión (`var i = 4`) es la expresión inicial cuyo resultado se calcula antes de la primera repetición. La segunda expresión (`i > 0`) es la condición que se comprueba antes de que se ejecute la reproducción indefinida. La tercera expresión (`i--`) se denomina *expresión posterior* y su resultado siempre se calcula después de que se ejecute la reproducción.

```
for (var i = 4; i > 0; i--){
    myMC.duplicateMovieClip("newMC" + i, i + 10);
}
```

#### Para reproducir indefinidamente a través de un clip de película u objeto secundario:

- Utilice la sentencia `for...in`.

Los subniveles incluyen otros clips de película, funciones, objetos y variables. El ejemplo siguiente utiliza la sentencia `trace` para imprimir el resultado en el panel Salida:

```
myObject = { name:'Joe', age:25, city:'San Francisco' };
for (propertyName in myObject) {
    trace("myObject tiene la propiedad: " + propertyName + ", con el valor: " +
        myObject[propertyName]);
}
```

Este ejemplo genera el siguiente resultado en el panel Salida:

```
myObject tiene la propiedad: name, con el valor: Joe
myObject tiene la propiedad: age, con el valor: 25
myObject tiene la propiedad: city, con el valor: San Francisco
```

Si lo desea, puede establecer que su script se repita sobre un tipo de subnivel en particular, por ejemplo, solamente sobre clips de película secundarios. Puede hacer esto con `for...in` junto con el operador `typeof`.

```
for (name in myMovieClip) {
    if (typeof (myMovieClip[name]) == "movieclip") {
        trace("Tengo un clip de película secundario llamado " + name);
    }
}
```

Para más información sobre cada acción, consulte las entradas individuales en el [Capítulo 12](#), “Diccionario de ActionScript”, en la página 213.

## Utilización del panel Acciones y la ventana Script

Puede incorporar scripts de Flash a su archivo FLA o guardarlos como archivos externos. Es recomendable guardar tanto código ActionScript como sea posible en archivos externos. Esto facilita la reutilización del código en múltiples archivos FLA. A continuación, en el archivo FLA, cree un script que utilice sentencias `#include` para acceder al código que ha almacenado externamente. Utilice el sufijo `.as` para identificar sus scripts como archivos ActionScript (AS). Si escribe archivos de clase personalizados, debe guardarlos como archivos AS externos.

**Nota:** el código ActionScript de los archivos externos se compila en un archivo SWF cuando se publica, se exporta, se prueba o se depura un archivo FLA. Por lo tanto, si realiza cambios en un archivo externo, debe guardarlo y volver a compilar todos los archivos FLA que lo utilizan.

Cuando se incorpora código ActionScript en un archivo FLA, se puede adjuntar código a fotogramas y objetos. Intente asociar ActionScript incorporado al primer fotograma de la línea de tiempo siempre que pueda. De ese modo, no tendrá que buscar en un archivo FLA para encontrar todo el código, ya que estará concentrado en una sola ubicación. Cree una capa con el nombre “Acciones” y coloque el código en ella. De esta manera, incluso si coloca código en otros fotogramas o si lo asocia a objetos, sólo deberá buscar en una capa para encontrarlo todo.

Para crear scripts que formen parte del documento, introduzca ActionScript directamente en el panel Acciones. Para crear scripts externos, utilice el editor de texto que desee o, en Flash Professional, utilice la ventana Script. Cuando utiliza el panel Acciones o la ventana Script, utiliza el mismo editor de ActionScript y escribe el código en el panel Script situado en la parte derecha del panel o ventana. Para reducir el proceso de escritura, puede seleccionar o arrastrar acciones desde la caja de herramientas Acciones hasta el panel Script.

**Para mostrar el panel Acciones, utilice uno de los siguientes procedimientos:**

- Seleccione Ventana > Paneles de desarrollo > Acciones.
- Presione F9.

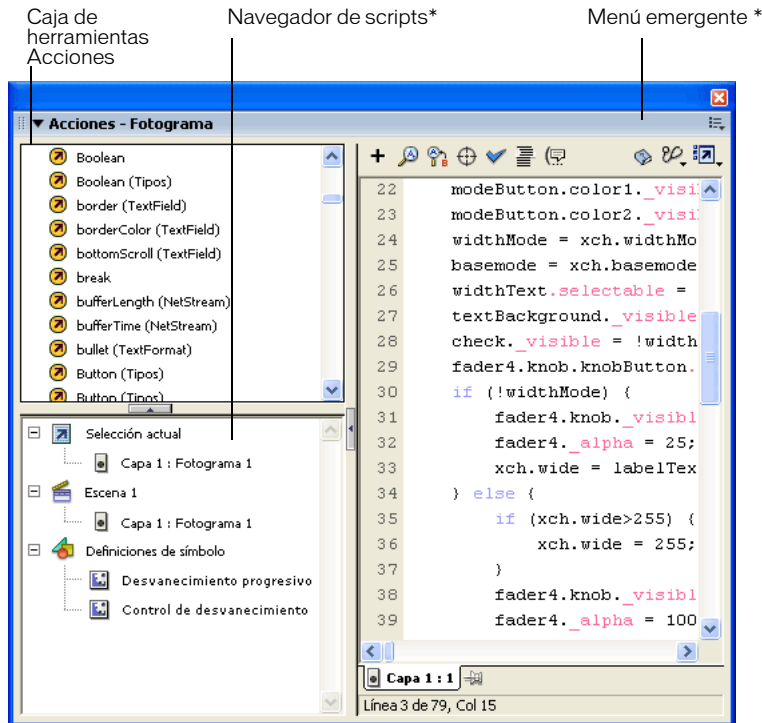
**(Sólo para Flash Professional) Para ver la ventana Script, siga uno de estos procedimientos:**

- Para empezar a escribir un nuevo script, seleccione Archivo > Nuevo > Archivo ActionScript.
- Para abrir un script existente, seleccione Archivo > Abrir y, a continuación, abra un archivo AS.
- Para editar un script que ya está abierto, haga clic en la ficha de documento que muestra el nombre del script (las fichas de documento sólo aparecen en Windows).

## Entorno del editor de ActionScript

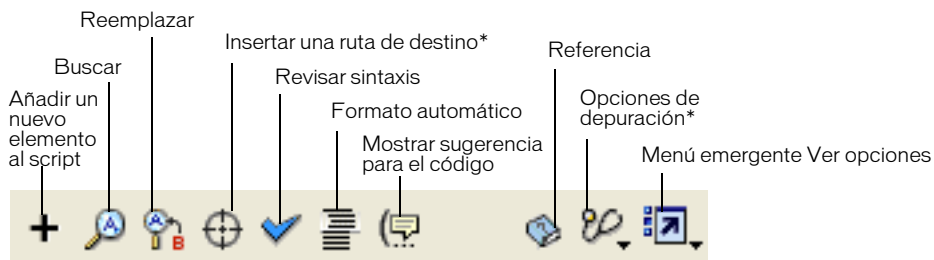
El entorno del editor de ActionScript consta de dos secciones. La sección de la derecha es el panel Script, el área en la que se escribe el código. La sección de la izquierda es una caja de herramientas Acciones que contiene una entrada para cada elemento del lenguaje ActionScript.

En el panel Acciones, la caja de herramientas Acciones también contiene un navegador de scripts, que es una representación visual de las ubicaciones del archivo FLA asociadas a ActionScript; puede desplazarse por dicho archivo FLA para localizar código de ActionScript. Si hace clic en un elemento del navegador de scripts, el script asociado con ese elemento aparecerá en el panel Script y la cabeza lectora se desplazará a esa posición en la línea de tiempo. Si hace doble clic en un elemento del navegador de scripts, el script quedará fijado (véase “[Administración de scripts en un archivo FLA](#)” en la página 62).



\* Sólo panel Acciones

También hay varios botones por encima del panel Script:



\* Sólo panel Acciones

Puede editar acciones, especificar parámetros para acciones o eliminar acciones directamente en el panel Script. También puede hacer doble clic en un elemento de la caja de herramientas Acciones o en el botón Añadir (+) situado sobre el panel Script para añadir acciones al panel Script.

## Administración de scripts en un archivo FLA

Si no concentra todo su código en un archivo FLA, en una única ubicación, puede *fijar* (bloquear en su sitio) varios scripts del panel Acciones para que sea más fácil moverse por ellos. En la figura siguiente, el script asociado con la ubicación actual de la línea de tiempo se encuentra en el fotograma 1 de la capa llamada Simplificar. La ficha del extremo izquierdo siempre indica la ubicación a lo largo de la línea de tiempo. Ese script también se fija (se muestra como la ficha situada más a la derecha). Se fijan otros dos scripts: uno en el fotograma 1 y otro en el fotograma 15 de la capa llamada Intro. Puede desplazarse entre los scripts fijados haciendo clic en las fichas o utilizando métodos abreviados de teclado. Desplazarse entre scripts fijados no cambia la posición actual en la línea de tiempo.



**Sugerencia:** si el contenido que muestra el panel Script no cambia de acuerdo con la ubicación que seleccione en la línea de tiempo, probablemente el panel Script está mostrando un script fijado. Haga clic en la ficha situada más a la izquierda en la parte inferior izquierda del panel Script para mostrar el ActionScript asociado a su ubicación a lo largo de la línea de tiempo.

### Para fijar un script:

- 1 Coloque el puntero en la línea de tiempo de manera que el script aparezca en una ficha en la parte inferior izquierda del panel Script del panel Acciones.
- 2 Siga uno de estos procedimientos:



- Haga clic en el icono de la chincheta situado a la derecha de la ficha. Si la chincheta tiene el aspecto del icono del situado más a la izquierda, el script estará ya fijado; si hace clic en el icono, lo liberará.
- Haga clic con el botón derecho del ratón (Windows) o haga clic con la tecla Control presionada (Macintosh) en la ficha y seleccione Fijar script.



- Seleccione Fijar script en el menú emergente Opciones (en la parte superior derecha del panel).

### Para liberar uno o más scripts:

- Siga uno de estos procedimientos:



- Si aparece un script fijado en una ficha en la parte inferior izquierda del panel Script del panel Acciones, haga clic en el icono de la chincheta situado a la derecha de la ficha. Si la chincheta tiene el aspecto del icono situado más a la izquierda, el script estará ya liberado; si hace clic en el icono, lo fijará.
- Haga clic con el botón derecho del ratón (Windows) o haga clic con la tecla Control presionada (Macintosh) en una ficha y seleccione Cerrar script o Cerrar todos los script.



- Seleccione Cerrar script o Cerrar todos los script en el menú emergente Opciones (en la parte superior derecha del panel).

### Para utilizar los métodos abreviados de teclado con scripts fijados:

- Puede utilizar los siguientes métodos abreviados de teclado para trabajar con scripts fijados:

Acción	Tecla de método abreviado de Windows	Tecla de método abreviado de Macintosh
Fijar script	Ctrl+= (signo igual)	Comando+=
Liberar script	Ctrl+- (signo menos)	Comando+-
Desplazar selección a la ficha de la derecha	Ctrl+Mayús+. (punto)	Comando+Mayús+.
Desplazar selección a la ficha de la izquierda	Ctrl+Mayús+, (coma)	Comando+Mayús+,
Liberar todos los scripts	Ctrl+Mayús+- (menos)	Comando+Mayús+-

## Utilización del editor de ActionScript

Flash MX 2004 y Flash MX Professional 2004 proporcionan diversas herramientas que le ayudan a escribir código con la sintaxis correcta y permiten configurar las preferencias para el formato de código y otras opciones. Encontrará una explicación de todo ello en esta sección.

### Resaltado de la sintaxis

En ActionScript, como en cualquier lenguaje, la *sintaxis* es la manera en la que los elementos se agrupan para crear un significado. Si utiliza una sintaxis de ActionScript incorrecta, los scripts no funcionarán.

Cuando escribe scripts en Flash MX 2004 y Flash MX Professional 2004, los comandos no admitidos por la versión del reproductor que esté utilizando se muestran en amarillo en la caja de herramientas Acciones. Por ejemplo, si la versión de Flash Player SWF está establecida en Flash 6, el código de ActionScript que sólo se admite en Flash Player 7 aparece en color amarillo en la caja de herramientas Acciones. Para más información sobre cómo establecer la versión de Flash Player SWF, consulte “Establecimiento de las opciones de publicación para el formato de archivo Flash SWF” en el apartado Utilización de Flash de la Ayuda.

También puede definir una preferencia para que Flash coloree partes de código de los scripts a medida que los escribe, a fin de resaltar los errores de escritura. Por ejemplo, supongamos que establece la preferencia de coloreado de la sintaxis de tal modo que las palabras clave se muestren en color verde oscuro. Mientras escribe código, si escribe `var`, la palabra `var` aparecerá en verde. No obstante si, por error, escribe `vae`, la palabra `vae` permanecerá de color negro, con lo cual usted sabrá inmediatamente que la ha escrito incorrectamente.

**Para definir las preferencias para el coloreado de la sintaxis a medida que escribe, realice una de las acciones siguientes:**

- Seleccione Edición > Preferencias y especifique la configuración de Color de sintaxis en la ficha ActionScript.
- En el panel Acciones, seleccione Preferencias en el menú emergente Opciones (parte superior derecha del panel) y especifique la configuración de Color de sintaxis en la ficha ActionScript.



## Escritura de código que activa las sugerencias para el código

Cuando trabaja con el editor de ActionScript (en el panel Acciones o la ventana Script), Flash puede detectar qué acción está especificando y visualizar una *sugerencia para el código*, es decir, una sugerencia que contiene la sintaxis completa para la acción, o un menú emergente que enumera los posibles nombres de método o propiedad. Las sugerencias para el código aparecen para los parámetros, propiedades y eventos cuando utiliza strict typing o asigna un nombre a los objetos de modo que el editor de ActionScript sepa qué sugerencias para el código debe mostrar, como se explica en esta sección. Para más información sobre cómo utilizar las sugerencias para el código cuando aparecen, consulte [“Utilización de las sugerencias para el código” en la página 66](#).

**Nota:** las sugerencias para el código están activadas automáticamente para clases nativas que no requieren que se cree y asigne un nombre a un objeto de la clase, como Math, Key, Mouse, etc.

### Strict typing de objetos para activar las sugerencias para el código

Cuando se utiliza ActionScript 2.0, se puede utilizar strict typing para una variable basada en una clase incorporada, como Button, Array, etc. De este modo, el editor de ActionScript muestra las sugerencias para el código para la variable. Por ejemplo, suponga que escribe lo siguiente:

```
var names:Array = new Array();  
names.
```

En cuanto escriba el punto (.), Flash mostrará una lista de métodos y propiedades disponibles para los objetos Array, porque ha definido la variable como una matriz. Para más información sobre la escritura de datos, consulte [“Strict data typing” en la página 40](#). Para más información sobre cómo utilizar las sugerencias para el código cuando aparecen, consulte [“Utilización de las sugerencias para el código” en la página 66](#).

### Utilización de sufijos para activar las sugerencias para el código

Si utiliza ActionScript 1 o desea visualizar las sugerencias para el código de los objetos que crea sin strict typing (véase [“Strict typing de objetos para activar las sugerencias para el código” en la página 64](#)), debe añadir un sufijo especial al nombre de cada objeto cuando lo cree. Por ejemplo, los sufijos que activan las sugerencias para el código para la clase Array y la clase Camera son `_array` y `_cam`, respectivamente. Si escribe el siguiente código:

```
var my_array = new Array();  
var my_cam = Camera.get();
```

y, a continuación, escribe uno de los siguientes (el nombre de variable seguido por un punto), se mostrarán las sugerencias para el código para el objeto Array y el objeto Camera, respectivamente.

```
my_array.  
my_cam.
```

Para los objetos que aparecen en el escenario, utilice el sufijo del cuadro de texto Nombre de instancia del inspector de propiedades. Por ejemplo, para ver las sugerencias para el código para objetos MovieClip, utilice el inspector de propiedades para asignar nombres de instancia con el sufijo `_mc` a todos los objetos MovieClip. Así, siempre que escriba el nombre de instancia seguido de un punto, se mostrarán las sugerencias para el código.

Aunque los sufijos no son necesarios para activar las sugerencias para el código cuando se utiliza el strict typing de un objeto, su utilización ayuda a comprender los scripts.



En la tabla siguiente se enumeran los sufijos necesarios para que se presenten sugerencias para el código automáticas.

Tipo de objeto	Sufijo de la variable
Array	_array
Button	_btn
Camera	_cam
Color	_color
ContextMenu	_cm
ContextMenuItem	_cmi
Date	_date
Error	_err
LoadVars	_lv
LocalConnection	_lc
Microphone	_mic
MovieClip	_mc
MovieClipLoader	_mcl
PrintJob	_pj
NetConnection	_nc
NetStream	_ns
SharedObject	_so
Sound	_sound
String	_str
TextField	_txt
TextFormat	_fmt
Video	_video
XML	_xml
XMLNode	_xmlnode
XMLSocket	_xmlsocket

Para más información sobre cómo utilizar las sugerencias para el código cuando aparecen, consulte [“Utilización de las sugerencias para el código” en la página 66](#).

## Utilización de comentarios para activar las sugerencias para el código

También puede utilizar comentarios de ActionScript para especificar una clase de objeto para las sugerencias para el código. En el ejemplo siguiente se indica a ActionScript que la clase de la instancia `theObject` es `Object` y así sucesivamente. Si especificara `mc` seguido de un punto después de estos comentarios, las sugerencias para el código mostrarían la lista de métodos y propiedades de `MovieClip`; si especificara `theArray` seguido de un punto, las sugerencias para el código mostrarían la lista de métodos y propiedades de `Array`, y así sucesivamente.

```
// Object theObject;  
// Array theArray;  
// MovieClip mc;
```

No obstante, Macromedia recomienda utilizar `strict data typing` (véase [“Strict typing de objetos para activar las sugerencias para el código” en la página 64](#)) o sufijos (véase [“Utilización de sufijos para activar las sugerencias para el código” en la página 64](#)) en lugar de esta técnica, porque dichas técnicas activan automáticamente las sugerencias para el código y hacen que el código sea más comprensible.

## Utilización de las sugerencias para el código

Las sugerencias para el código están desactivadas de forma predeterminada. Al definir las preferencias, puede desactivar las sugerencias para el código o determinar la rapidez con la que aparecen. Aunque las sugerencias para el código estén desactivadas en las preferencias, puede ver una sugerencia para el código de un comando específico.

**Para especificar la configuración de las sugerencias para el código automáticas, realice una de las acciones siguientes:**

- Seleccione Edición > Preferencias y active o desactive Sugerencias para el código en la ficha ActionScript.
- En el panel Acciones, seleccione Preferencias en el menú emergente Opciones (en la parte superior derecha del panel) y active o desactive Sugerencias para el código en la ficha ActionScript.

Si activa las sugerencias para el código, también puede especificar los segundos que deben transcurrir antes de que aparezcan dichas sugerencias para el código. Por ejemplo, si no tiene experiencia con ActionScript, seguramente preferirá que no haya ninguna demora, porque así las sugerencias para el código aparecen inmediatamente. No obstante, si suele saber lo que desea escribir y tan sólo necesita sugerencias cuando utiliza elementos de lenguaje con los que no está familiarizado, puede especificar una demora para que las sugerencias para el código no aparezcan cuando no desee utilizarlas.

**Para trabajar con sugerencias para el código con formato de información sobre herramientas:**

- 1 Visualice la sugerencia para el código escribiendo un paréntesis de apertura `[` después de un elemento que requiere paréntesis, como un nombre de método, un comando del tipo `if` o `do while`, etc.

Aparece la sugerencia para el código.

```
if {  
    if { condición } {  
    }  
}  
  
my_array.splice(  
    Array.splice( índice, conteo, elem1, ..., elemN )
```

**Nota:** si no aparece la sugerencia para el código, asegúrese de que no ha desactivado las sugerencias para el código en la ficha ActionScript. Si desea visualizar sugerencias para el código para una variable u objeto que ha creado, asegúrese de que ha asignado un nombre correcto a la variable o al objeto (véase [“Utilización de sufijos para activar las sugerencias para el código” en la página 64](#)), o que ha utilizado strict typing para la variable u objeto (véase [“Strict typing de objetos para activar las sugerencias para el código” en la página 64](#)).

- 2 Introduzca un valor para el parámetro. Si hay más de un parámetro, separe los valores con comas.

Los comandos sobrecargados, como gotoAndPlay() o for (es decir, funciones o métodos que pueden invocarse con diferentes conjuntos de parámetros) muestran un indicador que permite seleccionar el parámetro que desea establecer. Haga clic en los botones de flecha o presione Ctrl+Flecha izquierda o Ctrl+Flecha derecha para seleccionar el parámetro.

```
for {  
    1 de 2 For { inic; condición; siguiente } {  
    }  
}
```

- 3 Para cerrar la sugerencia para el código, utilice uno de los procedimientos siguientes:

- Escriba un paréntesis de cierre [)].
- Haga clic fuera de la sentencia.
- Presione Esc.

#### Para trabajar con las sugerencias para el código con formato de menú:

- 1 Visualice la sugerencia para el código escribiendo un punto después del nombre de objeto o variable.

Aparece el menú de sugerencias para el código.

```
my_mc.  
_alpha  
_currentframe  
_droptarget  
_focusrect  
_framesloaded  
_height  
_lockroot  
_name
```

**Nota:** si no aparece la sugerencia para el código, asegúrese de que no ha desactivado las sugerencias para el código en la ficha ActionScript. Si desea visualizar sugerencias para el código para una variable u objeto que ha creado, asegúrese de que ha asignado un nombre correcto a la variable o al objeto (véase [“Utilización de sufijos para activar las sugerencias para el código” en la página 64](#)), o que ha utilizado strict typing para la variable u objeto (véase [“Strict typing de objetos para activar las sugerencias para el código” en la página 64](#)).

- 2 Para desplazarse por las sugerencias para el código, utilice las teclas de flecha arriba y flecha abajo.
- 3 Para seleccionar un elemento del menú, presione Intro o el tabulador, o haga doble clic en el elemento.
- 4 Para cerrar la sugerencia para el código, utilice uno de los procedimientos siguientes:
  - Seleccione uno de los elementos del menú.
  - Haga clic fuera de la sentencia.
  - Escriba un paréntesis de cierre [)] si ya ha utilizado un paréntesis de apertura.
  - Presione Esc.

#### Para mostrar manualmente una sugerencia para el código:

- 1 Haga clic en un lugar del código donde puedan aparecer sugerencias para el código. A continuación se indican algunos ejemplos:
  - Después del punto que aparece a continuación de una sentencia o comando, donde debe especificarse una propiedad o un método
  - Entre paréntesis en un nombre de método
- 2 Siga uno de estos procedimientos:
  - Haga clic en el botón Mostrar sugerencias para el código que se encuentra encima del panel Script.
  - Presione Ctrl+barra espaciadora (Windows) o Comando+barra espaciadora (Macintosh).
  - Si trabaja con el panel Acciones, abra el menú emergente (en la parte derecha de la barra de título) y seleccione Mostrar sugerencias para el código.

## Utilización de las teclas de método abreviado de Esc

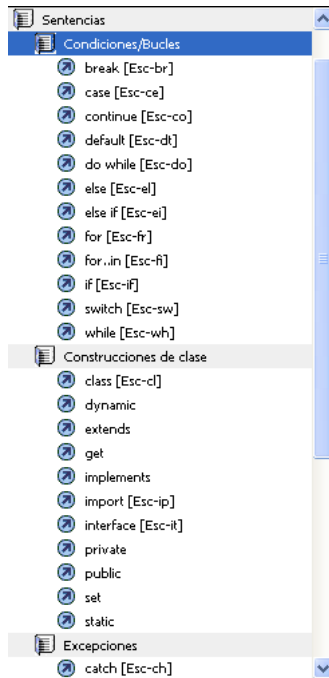
Puede añadir muchos elementos a un script mediante las teclas de método abreviado, es decir, presionando la tecla Esc y, a continuación, dos teclas más. Estos métodos abreviados son diferentes de los métodos abreviados de teclado que activan determinados comandos de menú. Por ejemplo, si trabaja en el panel Script y escribe Esc+d+o, se añade el siguiente código al script y el punto de inserción se coloca justo después de la palabra `while`, donde puede comenzar a escribir su condición:

```
do {  
} while ();
```

Del mismo modo, si escribe Esc+c+h, se añade el siguiente código al script y el punto de inserción se coloca entre los paréntesis, donde puede empezar a escribir la condición:

```
catch () {  
}
```

Para saber qué comandos (o acordarse de ellos) tienen teclas de método abreviado de Esc, puede verlos junto a los elementos del panel Acciones:



**Para ver u ocultar las teclas de método abreviado de Esc:**

- En el menú emergente Ver opciones, active o desactive Ver teclas de método abreviado de Esc.

## Comprobación de la sintaxis y la puntuación

Para determinar con precisión si el código que ha escrito responde como se esperaba, es necesario publicar o probar el archivo. Sin embargo, puede realizar una comprobación rápida del código ActionScript sin salir del archivo FLA. Los errores de sintaxis se muestran en el panel Salida. Cuando se comprueba la sintaxis, sólo se comprueba el script actual; el resto de los scripts que puedan encontrarse en el archivo FLA no se comprueban. También puede comprobar si los paréntesis, las llaves o los corchetes (operadores de acceso a matriz) de un bloque de código están emparejados.

**Para revisar la sintaxis, utilice uno de los procedimientos siguientes:**

- ✓ Haga clic en el botón Revisar sintaxis que se encuentra encima del panel Script.
- En el panel Acciones, visualice el menú emergente (en la parte superior derecha del panel) y seleccione Revisar sintaxis.
- Presione Ctrl+T (Windows) o Comando+T (Macintosh).

**Para comprobar si los signos de puntuación están emparejados:**

- 1 Haga clic entre las llaves ({}), los operadores de acceso a matriz ([] ) o los paréntesis (( )) del script.

- 2 Presione Ctrl+' (Windows) o Comando+' (Macintosh) para resaltar el texto que se encuentra entre los corchetes, las llaves o los paréntesis.  
El resaltado ayuda a comprobar si la puntuación de apertura dispone de su correspondiente puntuación de cierre.

## Aplicación de formato al código

Puede especificar una configuración para determinar si se debe aplicar formato y sangrías automática o manualmente al código. También puede especificar si se deben ver los números de línea y si se debe ajustar el texto de las líneas largas del código.

### Para establecer las opciones de formato:

- 1 Siga uno de estos procedimientos:



- En el panel Acciones, seleccione Opciones de Formato automático en el menú emergente Opciones (en la parte superior derecha del panel).
- (sólo para Flash Professional) En un archivo de script externo, seleccione Edición > Opciones de Formato automático.

Aparece el cuadro de diálogo Opciones de Formato automático.

- 2 Seleccione cualquiera de las casillas de verificación. Para ver el efecto de cada selección, examine el panel Vista previa.

Una vez establecidas las opciones de formato automático, la configuración se aplicará automáticamente al código que escriba, pero no al código que ya exista. Para aplicar la configuración al código ya existente, deberá hacerlo manualmente. Puede utilizar este procedimiento para aplicar formato al código al que se aplicó formato con una configuración diferente, que importó de otro editor, etc.

### Para aplicar formato al código según la configuración de Opciones de Formato automático, realice una de las acciones siguientes:



- Haga clic en el botón Formato automático que se encuentra encima del panel Script.
- Elija Formato automático en el menú emergente del panel Acciones.
- Presione Ctrl+Mayús+F (Windows) o Comando+Mayúsculas+F (Macintosh).

### Para utilizar una sangría automática:

- La sangría automática está activa de forma predeterminada. Para desactivarla, anule la selección de Sangría automática en las preferencias de ActionScript.

Cuando la sangría automática está activa, el texto que se escribe después de ( o { se sangra de forma automática según el valor de Tamaño de tabulación establecido en las preferencias de ActionScript. Para aplicar la sangría a otra línea, seleccione la línea y presione el tabulador. Para eliminar la sangría, presione Mayús y el tabulador simultáneamente.

### Para activar o desactivar los números de línea y el ajuste del texto:



- En el menú emergente Ver opciones, active o desactive Ver números de línea y Ajustar texto.

## Depuración de los scripts

Flash proporciona varias herramientas para comprobar el código de ActionScript en sus archivos SWF. El depurador, descrito en el resto de esta sección, permite detectar errores en un archivo SWF mientras se ejecuta en Flash Player. Flash también proporciona las siguientes herramientas de depuración adicionales:

- El panel Salida, que muestra mensajes de error y listas de variables y objetos (véase [“Utilización del panel Salida” en la página 80](#))
- La sentencia `trace`, que envía notas de programación y valores de expresiones al panel Salida (véase [“Utilización de la sentencia trace” en la página 82](#))
- Las sentencias `throw` y `try...catch...finally`, que sirven para probar y responder a los errores de tiempo de ejecución desde el script
- Completos mensajes de error de compilador, que permiten diagnosticar y solucionar problemas más fácilmente (véase el [Apéndice A, “Mensajes de error”, en la página 783](#))

Debe visualizar el archivo SWF con una versión especial de Flash Player llamada Reproductor de depuración de Flash. Cuando se instala la herramienta de edición, se instala automáticamente Reproductor de depuración de Flash. Así pues, si instala Flash y explora un sitio Web con contenido Flash, o ejecuta Probar película, estará utilizando Reproductor de depuración de Flash. También puede ejecutar el programa de instalación situado en el directorio `<app_dir>\Players\Debug\`, o iniciar la aplicación Reproductor de depuración de Flash independiente desde el mismo directorio.

Si utiliza el comando Probar película para probar películas con controles de teclado implementados (tabulación, métodos abreviados de teclado creados mediante `Key.addListener()`, etc), seleccione Control > Deshabilitar métodos abreviados de teclado. Al seleccionar esta opción impedirá que el entorno de edición “capte” las pulsaciones de teclado, permitiendo que lleguen hasta el reproductor. Por ejemplo, en el entorno de edición, Ctrl+U abre el cuadro de diálogo Preferencias. Si el script asigna Ctrl+U a una acción que subraya el texto en pantalla, cuando se utilice Probar película, al presionar Ctrl+U abrirá el cuadro de diálogo Preferencias en lugar de ejecutar la acción de subrayar texto. Para permitir que el comando Ctrl+U llegue hasta el reproductor, debe seleccionar Control > Deshabilitar métodos abreviados de teclado.

**Atención:** el comando Probar película produce un error si algún segmento de la ruta al archivo SWF presenta caracteres no representables mediante el esquema de codificación MBCS. Por ejemplo, las rutas japonesas no funcionan en un sistema inglés. Esta limitación se aplica a todas las áreas de la aplicación que utilizan el reproductor externo.

El depurador muestra una lista jerárquica de clips de película cargados actualmente en Flash Player. Con el depurador, puede ver y modificar los valores de variables y de propiedades mientras se reproduce el archivo SWF; además, puede utilizar puntos de corte para detener el archivo SWF y desplazarse por el código ActionScript línea a línea.

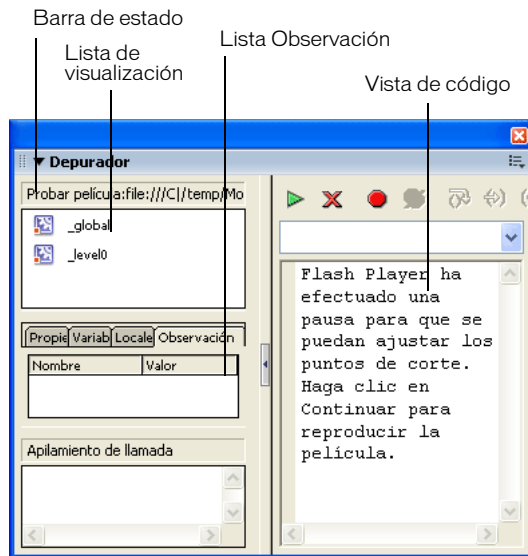
Puede utilizar el depurador en modo de prueba con archivos locales, o bien para probar archivos de un servidor Web que se encuentren en una ubicación remota. El depurador permite establecer puntos de corte en el código de ActionScript que detienen Flash Player y desplazarse por el código a medida que se ejecuta. Después puede volver a editar los scripts de modo que produzcan los resultados correctos.

Una vez activado el depurador, su barra de estado muestra la URL o la ruta del archivo, indica si el depurador se ejecuta en modo de prueba o desde una ubicación remota y muestra una vista dinámica de la lista de visualización del clip de película. Cuando se agregan o eliminan clips de película del archivo, la lista de visualización refleja los cambios al instante. Puede cambiar el tamaño de la lista de visualización moviendo el separador horizontal.

#### Para activar el Depurador en modo de prueba:

- Seleccione Control > Depurar película.

El depurador se abrirá. También abre el archivo SWF en modo de prueba.



## Depuración de un archivo SWF desde una ubicación remota

Puede depurar un archivo SWF remoto utilizando las versiones independientes, ActiveX o de plugin de Flash Player. Durante la exportación de un archivo SWF, puede activar la depuración en el archivo y crear una contraseña de depuración. Si la depuración no se activa, el depurador no se activará.

Para asegurarse de que sólo los usuarios de confianza puedan ejecutar sus archivos SWF en el reproductor de depuración de Flash, publique el archivo con una contraseña de depuración. Del mismo modo que ocurre con JavaScript o HTML, los usuarios tienen la posibilidad de ver variables de cliente en ActionScript. Para guardar variables de un modo seguro, debe enviarlas a una aplicación de servidor en lugar de almacenarlas en su archivo. No obstante, como creador de Flash, podría tener otros secretos comerciales, como estructuras de clips de película, que no desea revelar. Puede utilizar una contraseña de depuración para proteger su trabajo.

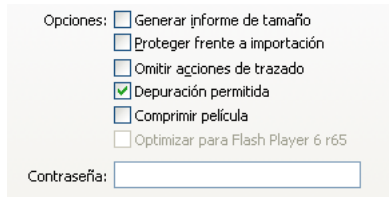
Al exportar, publicar, o probar una película, Flash crea un archivo SWD que contiene información de depuración. Para depurar de forma remota, debe colocar el archivo SWD en el mismo directorio que el archivo SWF del servidor.

#### Para activar la depuración remota de una película Flash:

- 1 Seleccione Archivo > Configuración de publicación.



- 2 En la ficha Flash del cuadro de diálogo Configuración de publicación, seleccione Depuración permitida.



Opciones: ☐ Generar informe de tamaño  
☐ Proteger frente a importación  
☐ Omitir acciones de trazado  
☒ Depuración permitida  
☐ Comprimir película  
☐ Optimizar para Flash Player 6 r65

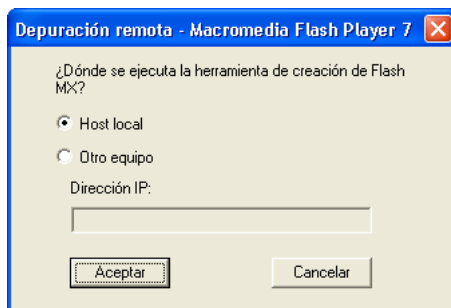
Contraseña:

- 3 Para establecer una contraseña, especifíquela en el cuadro Contraseña.  
Una vez que haya establecido esta contraseña, nadie podrá descargar información al depurador sin ella. Sin embargo, si deja vacío el campo de contraseña, no será necesaria ninguna contraseña.
- 4 Cierre el cuadro de diálogo Configuración de publicación y seleccione uno de los comandos siguientes:
  - Control > Depurar película
  - Archivo > Exportar película
  - Archivo > Configuración de publicación/PublicarFlash crea un archivo de depuración con la extensión .swd y lo guarda junto con el archivo SWF. El archivo SWD contiene información que permite utilizar puntos de corte y desplazarse por el código.
- 5 Coloque el archivo SWD en el mismo directorio que el archivo SWF en el servidor.  
Si el archivo SWD no está en el mismo directorio que el archivo SWF podrá igualmente depurar de forma remota, pero el depurador pasará por alto los puntos de corte y no podrá desplazarse paso a paso por el código.
- 6 En Flash, seleccione Ventana > Paneles de desarrollo > Depurador.
  - En el depurador, seleccione Habilitar depuración remota en el menú emergente Opciones (en la parte superior derecha del panel).

#### Para activar el depurador desde una ubicación remota:

- 1 Abra la aplicación de edición de Flash.
- 2 En un navegador o reproductor independiente, abra el archivo SWF publicado desde la ubicación remota.

Aparecerá el cuadro de diálogo de depuración remota.



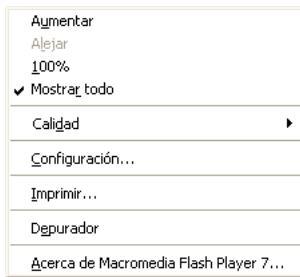
Depuración remota - Macromedia Flash Player 7

¿Dónde se ejecuta la herramienta de creación de Flash MX?

☒ Host local  
☐ Otro equipo

Dirección IP:

Si no aparece este cuadro de diálogo es porque Flash no ha podido encontrar el archivo SWD. En tal caso, haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) en el archivo SWF para ver el menú contextual y seleccione Depurador.



- 3 En el cuadro de diálogo de depuración remota, seleccione Host local u Otro equipo:
  - Seleccione la primera opción si el reproductor de depuración y la aplicación de edición de Flash están instalados en el mismo equipo.
  - Seleccione la segunda opción si el reproductor de depuración y la aplicación de edición de Flash no están instalados en el mismo equipo. Introduzca la dirección IP del equipo en el que se ejecuta la aplicación de edición de Flash.
- 4 Al establecer una conexión, aparece un mensaje de solicitud de contraseña. Introduzca la contraseña de depuración, si ha establecido una.  
La lista de visualización del archivo SWF aparecerá en el depurador.

## Visualización y modificación de variables

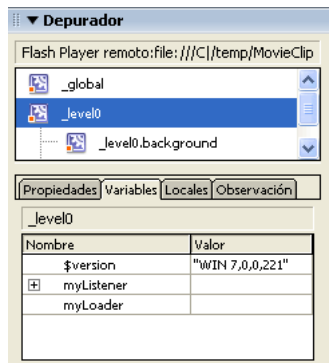
La ficha Variables del depurador muestra los nombres y los valores de las variables globales y de línea de tiempo existentes en el archivo SWF. Si se modifica el valor de una variable en la ficha Variables, el cambio se reflejará en el archivo SWF mientras éste se ejecuta. Por ejemplo, para comprobar la detección de una colisión en un juego, puede introducir el valor de la variable para situar una pelota en el lugar correcto junto a una pared.

En la ficha Locales del depurador se muestran los nombres y los valores de las variables locales que están disponibles en los sitios donde el archivo SWF se ha detenido, a causa de un punto de corte o en cualquier otro lugar dentro de una función definida por el usuario.

### Para ver una variable:

- 1 Seleccione el clip de película que contiene la variable en la lista de visualización.  
Para mostrar las variables globales, seleccione el clip `_global` de la lista de visualización.
- 2 Haga clic en la ficha Variables.

La lista de visualización se actualizará automáticamente según se vaya reproduciendo el archivo SWF. Si se elimina del archivo SWF de un clip de película en un fotograma determinado, dicho clip de película se eliminará también de la lista de visualización del depurador, junto con su variable y el nombre de la variable. No obstante, si marca una variable para la lista Observación (véase [“Utilización de la lista Observación” en la página 75](#)), dicha variable no se elimina.



**Para modificar el valor de una variable:**

- Haga doble clic en el valor y especifique otro.

El valor no puede ser una expresión. Por ejemplo, puede utilizar "Hola", 3523 o "http://www.macromedia.com", pero no puede utilizar `x + 2` ni `eval("name:" + i)`. El valor puede ser una cadena (cualquier valor entre comillas), un número o un valor booleano (`true` o `false`).

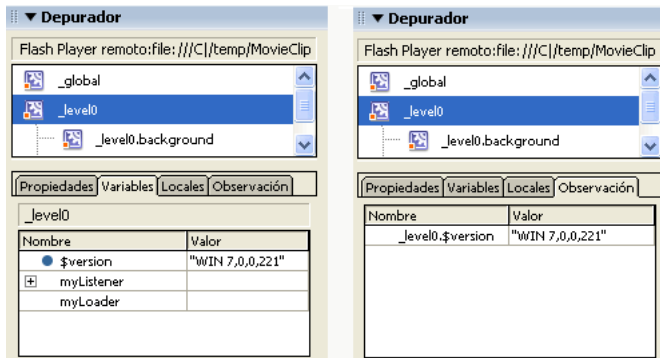
**Nota:** para escribir el valor de una expresión en el panel Salida en modo de prueba, utilice la sentencia `trace`. Véase [“Utilización de la sentencia trace” en la página 82](#).

## Utilización de la lista Observación

Para controlar un conjunto de variables críticas de un modo razonable, debe marcar las variables que desee que aparezcan en la lista Observación. La lista Observación muestra la ruta absoluta de la variable, así como su valor. También puede introducir un nuevo valor de variable en la lista Observación del mismo modo que puede hacerlo en la ficha Variables.

Si añade una variable local a la lista Observación, su valor sólo aparece cuando Flash Player se detiene en una línea de ActionScript donde la variable se encuentra dentro de su ámbito. Todas las demás variables aparecen mientras se reproduce el archivo SWF. Si el depurador no encuentra el valor de la variable, éste aparece en la lista como no definido.

En la lista Observación sólo pueden mostrarse variables, no propiedades ni funciones.



*Variables marcadas para la lista Observación y variables de la lista Observación*

**Utilice uno de los siguientes procedimientos para agregar variables a la lista Observación:**

- En la ficha Variables o Locales, haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) en una variable seleccionada y elija Observación en el menú contextual. Aparecerá un punto azul junto a la variable.
- En la ficha Observación, haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) y seleccione Añadir en el menú contextual. Introduzca la ruta de destino para el nombre de la variable y el valor en los campos correspondientes.

**Para eliminar variables de la lista Observación:**

- En la ficha Observación, haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) y seleccione Quitar en el menú contextual.

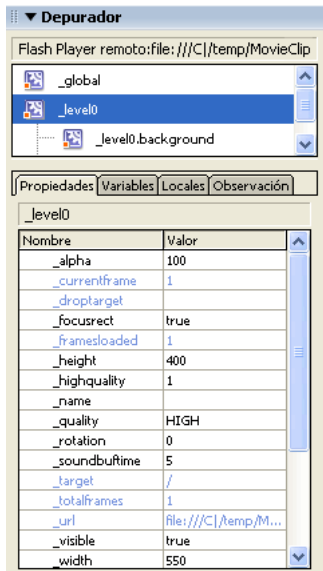
## Visualización de las propiedades de un clip de película y modificación de las propiedades editables

La ficha Propiedades del depurador muestra todos los valores de propiedades de los clips de película del escenario. Puede modificar un valor y ver su efecto en el archivo SWF mientras se ejecuta. Algunas propiedades de los clips de película son de sólo lectura y no pueden modificarse.

**Para ver las propiedades de un clip de película en el depurador:**

- 1 Seleccione un clip de película de la lista de visualización.

## 2 Haga clic en la ficha Propiedades del depurador.



### Para modificar el valor de una propiedad:

- Haga doble clic en el valor y especifique otro.

El valor no puede ser una expresión. Por ejemplo, puede introducir 50 o "clearwater", pero no  $x + 50$ . El valor puede ser una cadena (cualquier valor entre comillas), un número o un valor booleano (true o false). No se permiten valores de objeto o matriz (por ejemplo, {id: "rogue"} o [1, 2, 3]) en el depurador.

Para más información, consulte [“Operadores de cadena” en la página 49](#) y [“Utilización de operadores para manipular los valores de las expresiones” en la página 47](#).

**Nota:** para escribir el valor de una expresión en el panel Salida en modo de prueba, utilice la sentencia `trace`. Véase [“Utilización de la sentencia trace” en la página 82](#).

## Establecimiento y eliminación de puntos de corte

Los puntos de corte permiten detener un archivo SWF que se está ejecutando en Flash Player en una línea específica del código de ActionScript. Puede utilizar puntos de corte para probar posibles puntos problemáticos en el código. Por ejemplo, si ha escrito un conjunto de sentencias `if...else if` y no puede determinar cuál se está ejecutando, añada un punto de corte delante de las sentencias y desplácese por ellas de una en una en el depurador.

Puede establecer puntos de corte en el panel Acciones o en el depurador (para establecer puntos de corte en scripts externos, debe utilizar el depurador). Los puntos de corte establecidos en el panel Acciones se guardan con el documento de Flash (archivo FLA). Los puntos de corte establecidos en el depurador no se guardan en el archivo FLA y sólo son válidos para la sesión de depuración actual.



**Para establecer o eliminar un punto de corte del panel Acciones, realice una de las acciones siguientes:**

- Haga clic en el margen izquierdo. Un punto rojo indica un punto de corte.
- Haga clic en el botón Opciones de depuración situado encima del panel Script.
- Haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) para ver el menú contextual y seleccione Definir punto de corte, Quitar punto de corte o Quitar todos los puntos de corte.
- Presione las teclas Ctrl+Mayús+B (Windows) o Comando+Mayúsculas+B (Macintosh).

**Nota:** en las versiones anteriores de Flash, al hacer clic en el margen izquierdo del panel Script se seleccionaba la línea de código; ahora, se añade o elimina un punto de corte. Para seleccionar una línea de código, haga clic mientras presiona la tecla Control (Windows) o Comando (Macintosh).

**Para establecer y quitar puntos de corte en el depurador, realice una de las acciones siguientes:**

- Haga clic en el margen izquierdo. Un punto rojo indica un punto de corte.
- Haga clic en el botón Alternar punto de corte o Quitar todos los puntos de corte que se encuentra encima de la vista de código.
- Haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) para ver el menú contextual y seleccione Definir punto de corte, Quitar punto de corte o Quitar todos los puntos de corte.
- Presione las teclas Ctrl+Mayús+B (Windows) o Comando+Mayúsculas+B (Macintosh).

Cuando Flash Player se haya detenido en un punto de corte, puede entrar, salir de esa línea de código o pasar por encima de ella. Si establece un punto de corte en un comentario o en una línea vacía en el panel Acciones, el punto de corte se pasará por alto.

## Desplazamiento por las líneas de código

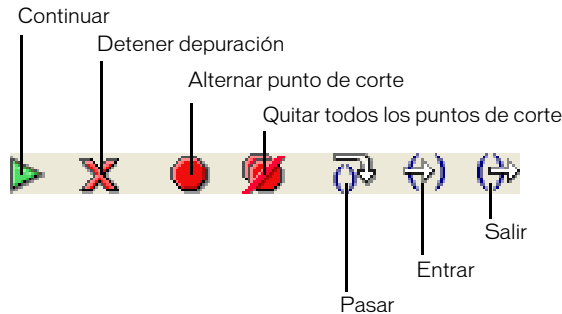
Al iniciar una sesión de depuración, Flash Player efectúa una pausa. Si define puntos de corte en el panel Acciones, bastará con hacer clic en el botón Continuar para reproducir el archivo SWF hasta que llegue a un punto de corte. Por ejemplo, en el código siguiente, supongamos que se define un punto de corte dentro de un botón en la línea `myFunction()`:

```
on (press) {  
    myFunction();  
}
```

Al hacer clic en el botón, se alcanza el punto de corte y Flash Player efectúa una pausa. A continuación, puede entrar en el código para llevar el depurador a la primera línea de la función `myFunction()`, sin importar en qué punto del documento está definida. También puede avanzar hasta el final de la función o salir de ésta.

Si no se han definido puntos de corte en el panel Acciones, puede utilizar el menú de salto del depurador para seleccionar el script que desee de la película. Tras seleccionar un script, puede agregarle puntos de corte. Tras agregar puntos de corte, deberá hacer clic en el botón Continuar para iniciar la película. El depurador se detiene al llegar al punto de corte.

A medida que se desplaza por las líneas de código, los valores de las variables y las propiedades cambian en la lista Observación y en las fichas Variables, Locales y Propiedades. La flecha amarilla en la parte izquierda de la vista de código del depurador indica la línea en la que se ha detenido el depurador. Utilice los botones siguientes situados en la parte superior de la vista de código:



**Entrar** hace que el depurador (indicado por la flecha amarilla) entre en una función. Entrar sólo funciona para las funciones definidas por el usuario.

En el ejemplo siguiente, si coloca un punto de corte en la línea 7 y hace clic en Entrar, el depurador avanza a la línea 2 y al hacer otro clic en Entrar se avanzará hasta la línea 3. Al hacer clic en Entrar para líneas que no contienen funciones definidas por el usuario, el depurador pasa una línea de código. Por ejemplo, si se detiene en la línea 2 y selecciona Entrar, el depurador avanzará hasta la línea 3, como se muestra en el ejemplo siguiente:

```
1 function myFunction() {
2   x = 0;
3   y = 0;
4 }
5
6 mover = 1;
7 myFunction();
8 mover = 0;
```

**Salir** hace avanzar el depurador hasta salir de una función. Este botón sólo funciona si el depurador está detenido en una función definida por el usuario; desplaza la flecha amarilla hasta la línea posterior a la línea desde donde se llamó a la función. En el ejemplo anterior, si coloca un punto de corte en la línea 3 y hace clic en Salir, el depurador pasa a la línea 8. Cuando se hace clic en una línea que no se encuentra en una función definida por el usuario, se produce el mismo resultado que cuando se hace clic en Continuar. Por ejemplo, si se detiene en la línea 6 y hace clic en Salir, el reproductor continúa ejecutando el script hasta que encuentra un punto de corte.

**Pasar** hace avanzar el depurador pasando una línea de código. Este botón desplaza la flecha amarilla hasta la línea siguiente del script y pasa por alto las funciones definidas por el usuario. En el ejemplo anterior, si se ha detenido en la línea 7 y hace clic en Pasar, irá directamente a la línea 8 y `myFunction()` se pasará por alto.

**Continuar** deja la línea en la que se ha detenido el reproductor y prosigue la reproducción hasta llegar a un punto de corte.

**Detener depuración** desactiva el depurador, pero hace que el archivo SWF siga reproduciéndose en Flash Player.


## Utilización del panel Salida

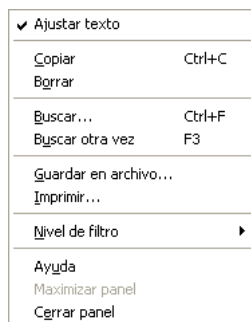
En el modo de prueba, el panel Salida muestra información que ayuda a solucionar problemas relacionados con el archivo SWF. Cierta información, como los errores sintácticos, se muestra automáticamente. Puede accederse a otro tipo de información mediante los comandos Mostrar objetos y Mostrar variables (véase “Lista de los objetos de un archivo SWF” en la página 80 y “Lista de las variables de un archivo SWF” en la página 81).

Si se utiliza la sentencia `trace` en los scripts, puede enviar información específica al panel Salida durante la ejecución del archivo SWF. Por ejemplo, podrían enviarse notas acerca del estado del archivo SWF o el valor de una expresión. Véase “Utilización de la sentencia `trace`” en la página 82.

Para ver el panel Salida, seleccione Ventana > Paneles de desarrollo > Salida o presione F2.

**Nota:** si el script contiene errores de sintaxis, el panel Salida aparece automáticamente cuando se comprueba la sintaxis o se prueba el archivo SWF.

 Para trabajar con el contenido del panel Salida, utilice el menú emergente Opciones situado en la esquina superior derecha.



## Lista de los objetos de un archivo SWF

En el modo de prueba, el comando Mostrar objetos muestra el nivel, el fotograma, el tipo de objeto (forma, clip de película o botón), las rutas de destino y los nombres de instancia de los clips de película, los botones y los campos de texto en una lista jerárquica. Resulta especialmente útil para encontrar el nombre de instancia y la ruta de destino correctas. A diferencia del depurador, la lista no se actualiza automáticamente a medida que se reproduce el archivo SWF; es necesario seleccionar el comando Mostrar objetos cada vez que se desee enviar información al panel Salida.

El comando Mostrar objetos no enumera todos los objetos de datos de ActionScript. En este contexto, el objeto se considera como una forma o símbolo en el escenario.

**Para mostrar una lista de objetos en una película:**

- 1 Si la película no se está ejecutando en el modo de prueba, seleccione Control > Probar película.
- 2 Seleccione Depurar > Mostrar objetos.

En el panel Salida se mostrará una lista de todos los objetos que se encuentren en el escenario, como en el ejemplo siguiente:

```
Level #0: Frame=1 Label="Scene_1"  
  Button: Target="_level0.myButton"  
    Shape:  
      Movie Clip: Frame=1 Target="_level0.myMovieClip"
```



```
Shape:  
Edit Text: Target="_level0.myTextField" Text="Texto de ejemplo."
```

## Lista de las variables de un archivo SWF

En el modo de prueba, el comando Mostrar variables muestra una lista de todas las variables existentes actualmente en el archivo SWF. Resulta especialmente útil para encontrar el nombre de la variable y la ruta de destino correctos. A diferencia del depurador, la lista no se actualiza automáticamente a medida que se reproduce el archivo SWF; es necesario seleccionar el comando Mostrar variables cada vez que se desee enviar la información al panel Salida.

El comando Mostrar variables también muestra las variables globales declaradas con el identificador `_global`. Las variables globales se muestran en la parte superior de la ventana de resultados de Mostrar variables en una sección titulada "Variables globales" y cada variable va precedida de `_global`.

Además, el comando Mostrar variables muestra propiedades del captador/definidor, propiedades creadas con el método `Object.addProperty()` y que invocan métodos `get` (captador) o `set` (definidor). Las propiedades del captador/definidor se muestran junto a las demás propiedades del objeto al que pertenecen. Para distinguir fácilmente estas propiedades de las variables normales, los valores de las propiedades del captador/definidor aparecen precedidas por la cadena `[getter/setter]`. El valor que se muestra para una propiedad del captador/definidor se determina calculando el resultado de la función `get` de la propiedad.

### Para mostrar una lista de las variables de un archivo SWF:

- 1 Si el archivo SWF no se está ejecutando en el modo de prueba, seleccione Control > Probar película.
- 2 Seleccione Depurar > Mostrar variables.

En el panel Salida se mostrará una lista de todas las variables existentes actualmente en el archivo SWF, como en el ejemplo siguiente:

```
Global Variables:  
Variable _global.MyGlobalArray = [object #1] [  
    0:1,  
    1:2,  
    2:3  
]  
Level #0:  
Variable _level0.$version = "WIN 6,0,0,101"  
Variable _level0.RegularVariable = "Gary"  
Variable _level0.AnObject = [object #1] {  
    MyProperty: [getter/setter] 3,14159  
}
```

## Visualización de las propiedades de un campo de texto para la depuración

Para obtener información de depuración sobre los objetos `TextField`, utilice el comando Depurar > Mostrar variables en el modo de probar película. El panel Salida utiliza las siguientes convenciones al mostrar los objetos `TextField`:

- Si no se encuentra una propiedad en el objeto, no se muestra.
- En una línea se muestran como máximo cuatro propiedades.
- Una propiedad con un valor de cadena se muestra en una línea aparte.

- Si hay otras propiedades definidas para el objeto una vez que se han procesado las propiedades incorporadas, éstas se añaden a la pantalla siguiendo las reglas de los puntos segundo y tercero que acabamos de mencionar.
- Las propiedades de color se muestran como números hexadecimales (0x00FF00).
- Las propiedades se muestran en el orden siguiente: variable, text, htmlText, html, textWidth, textHeight, maxChars, borderColor, backgroundColor, textColor, border, background, wordWrap, password, multiline, selectable, scroll, hscroll, maxscroll, maxhscroll, bottomScroll, type, embedFonts, restrict, length, tabIndex, autoSize.

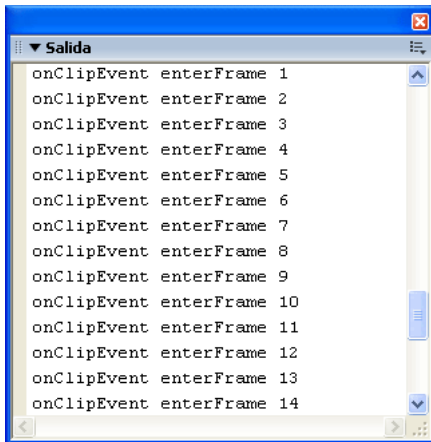
El comando Depurar > Mostrar objetos en modo de prueba enumera los objetos TextField. Si se especifica un nombre de instancia para un campo de texto, en el panel Salida se mostrará la ruta de destino completa, incluido el nombre de instancia, con el formato siguiente:

`Target = "ruta de destino"`

## Utilización de la sentencia trace

Cuando se utiliza la sentencia `trace` en un script, puede enviarse información al panel Salida. Por ejemplo, cuando se prueba una película o una escena, pueden enviarse notas de programación determinadas al panel o hacer que aparezcan resultados concretos cuando se presiona un botón o se reproduce un fotograma. La sentencia `trace` es similar a la sentencia `alert` de JavaScript.

El uso de la sentencia `trace` en un script permite utilizar expresiones como parámetros. El valor de una expresión se muestra en el panel Salida en modo de prueba, como en el ejemplo siguiente:



*La sentencia `trace` devuelve valores que aparecen en el panel Salida.*

```
onClipEvent(enterFrame) {
    trace("onClipEvent enterFrame " + enterFrame++)
}
```

## Actualización de Flash Player para realizar pruebas

Puede descargar la última versión de Flash Player del Centro de soporte de Flash de Macromedia en [www.macromedia.com/go/flash\\_support\\_sp](http://www.macromedia.com/go/flash_support_sp) y utilizarla para probar los archivos SWF con la versión más reciente de Flash Player.

## PARTE II

# Gestión de eventos y creación de interacciones

Los eventos puede generarlos el usuario, por ejemplo, al hacer clic con el ratón o al presionar una tecla, o pueden producirse como consecuencia de otro proceso, como por ejemplo al cargar un archivo XML a través de la red. En el primer capítulo de esta sección se describen los diferentes tipos de eventos de Macromedia Flash y cómo gestionarlos en ActionScript. En el segundo capítulo se muestra cómo aplicar estos principios para crear presentaciones interactivas, aplicaciones y animaciones sencillas.

Capítulo 4: Gestión de eventos ..... 85

Capítulo 5: Creación de interacciones con ActionScript. .... 93



# CAPÍTULO 4

## Gestión de eventos

Un *evento* es un suceso de software o de hardware que requiere una respuesta de una aplicación de Macromedia Flash. Por ejemplo, un evento como hacer clic con el ratón o presionar una tecla se denomina *evento de usuario*, puesto que es una consecuencia directa de una acción del usuario. Un evento generado automáticamente por Flash Player, como la aparición inicial de un clip de película en el escenario, se denomina *evento del sistema*, porque no lo genera directamente el usuario.

A fin de que la aplicación reaccione ante los eventos, debe utilizar *controladores de eventos*, es decir, código ActionScript asociado con un objeto y un evento determinados. Por ejemplo, si un usuario hace clic en un botón del escenario, se podría avanzar la cabeza lectora hasta el siguiente fotograma. O bien, al finalizar la carga de un archivo XML por la red, se podría mostrar el contenido de dicho archivo en un campo de texto.

ActionScript proporciona varias maneras diferentes de controlar los eventos: métodos de controlador de eventos, detectores de eventos y controladores de eventos de clip de película y de botón.

### Utilización de métodos de controlador de eventos

Un método de controlador de eventos es un método de clase que se invoca cuando se produce un evento en una instancia de dicha clase. Por ejemplo, la clase Button define un controlador de eventos `onPress` que se invoca cada vez que se presiona el ratón en un objeto Button. A diferencia del resto de los métodos de una clase, un controlador de eventos no se invoca directamente; Flash Player lo invoca automáticamente cuando se produce el evento pertinente.

De forma predeterminada, los métodos de controlador de eventos no están definidos: cuando se produce un evento determinado, el controlador de eventos correspondiente se invoca, pero la aplicación no responde de ninguna otra forma al evento. Para que la aplicación responda al evento, se debe definir una función con la sentencia `function` y después asignarla al controlador de eventos que corresponda. La función que se asigne al controlador de eventos se invocará automáticamente siempre que se produzca el evento.

Un controlador de eventos se compone de tres partes: el objeto al que se aplica el evento, el nombre del método del controlador de eventos del objeto y la función que se asigna al controlador. En el ejemplo siguiente se muestra la estructura básica de un controlador de eventos.

```
object.eventMethod = function () {  
    // El código se escribe aquí, en respuesta al evento  
}
```

Por ejemplo, imagine que en el escenario hay un botón denominado `next_btn`. El código siguiente asigna una función al controlador de eventos `onPress` del botón; esta función hace que la cabeza lectora avance hasta el siguiente fotograma de la línea de tiempo.

```
next_btn.onPress = function ()
    nextFrame();
}
```

En el código anterior, la función `nextFrame()` se asigna directamente a `onPress`. También se puede asignar una referencia de función (nombre) a un método de controlador de eventos y definir la función posteriormente.

```
// Asignar referencia de función al método de controlador de eventos onPress
del botón
next_btn.onPress = goNextFrame;

// Definir la función doSubmit()
function goNextFrame() {
    nextFrame();
}
```

Observe que se asigna una referencia de función, no el valor devuelto por la función, al controlador de eventos `onPress`.

```
// Incorrecto
next_btn.onPress = goNextFrame();
// Correcto
next_btn.onPress = goNextFrame;
```

Algunos controladores de eventos reciben parámetros que proporcionan información acerca del evento que se ha producido. Por ejemplo, el controlador de eventos `TextField.onSetFocus` se invoca cuando una instancia de campo de texto se selecciona con el teclado. Este controlador de eventos recibe una referencia al objeto de campo de texto que anteriormente estaba seleccionado con el teclado.

Por ejemplo, el código siguiente inserta texto en el campo de texto que antes estaba seleccionado con el teclado.

```
userName_txt.onSetFocus = function(oldFocus_txt) {
    oldFocus_txt.text = "Ya no estoy seleccionado con el teclado";
}
```

Las siguientes clases de `ActionScript` definen controladores de eventos: `Button`, `ContextMenu`, `ContextMenuItem`, `Key`, `LoadVars`, `LocalConnection`, `Mouse`, `MovieClip`, `MovieClipLoader`, `Selection`, `SharedObject`, `Sound`, `Stage`, `TextField`, `XML` y `XMLSocket`. Para más información sobre los controladores de eventos que proporcionan estas clases, consulte las entradas para estas clases que aparecen en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

También se pueden asignar funciones a los controladores de eventos para los objetos que crea el usuario en el tiempo de ejecución. Por ejemplo, el código siguiente crea una nueva instancia de clip de película (`newclip_mc`) y después asigna una función al controlador de eventos `onPress` del clip.

```
_root.attachMovie("symbolID", "newclip_mc", 10);
newclip_mc.onPress = function () {
    trace("Me ha presionado");
}
```

Para más información, consulte [“Creación de clips de película en tiempo de ejecución” en la página 128](#).

## Utilización de detectores de eventos

Los detectores de eventos permiten que un objeto, denominado *objeto detector*, reciba eventos generados por otro objeto, denominado *objeto difusor*. El objeto difusor registra el objeto detector que va a recibir los eventos generados por el difusor. Por ejemplo, se puede registrar un clip de película para que reciba notificaciones `onResize` desde el escenario, o que una instancia de botón reciba notificaciones `onChanged` de un objeto de campo de texto. Se pueden registrar varios objetos detectores para que reciban eventos de un único difusor y se puede registrar un único objeto detector para que reciba eventos de varios difusores.

El modelo de eventos de los detectores de eventos es muy parecido al de los controladores de eventos (véase [“Utilización de métodos de controlador de eventos” en la página 85](#)), pero se diferencian en lo siguiente:

- El objeto al que se asigna el controlador de eventos no es el objeto que emite el evento.
- Se llama a un método especial del objeto difusor, `addListener()`, que registra el objeto detector para que reciba sus eventos.

Para utilizar detectores de eventos, se crea un objeto detector con una propiedad denominada igual que el evento que genera el objeto difusor. A continuación, se asigna una función al detector de eventos que responde de alguna manera al evento. Finalmente, se llama a `addListener()` en el objeto que está difundiendo el evento y se le pasa el nombre del objeto detector. En el código siguiente se aprecia el modelo del detector de eventos.

```
listenerObject.eventName = function(){  
    // el código se escribe aquí  
};  
broadcastObject.addListener(listenerObject);
```

El objeto detector especificado puede ser cualquier objeto, como un clip de película o una instancia de botón del escenario o una instancia de cualquiera de las clases de `ActionScript`. El nombre de evento es un evento que se produce en el `broadcastObject`, que a su vez difunde el evento al `listenerObject`. Se pueden registrar varios detectores de eventos para un único difusor de eventos.

En el ejemplo siguiente se muestra cómo utilizar el detector de eventos `Selection.onSetFocus` para crear un gestor de selección simple para un grupo de campos de introducción de texto. En este caso, el borde del campo de texto que se selecciona con el teclado se activa (aparece) mientras que el borde del texto que deja de estar seleccionado se desactiva.

### Para crear un gestor de selección simple con detectores de eventos:

- 1 Con la herramienta Texto, cree un campo de texto en el escenario.
- 2 Seleccione el campo de texto y, en el inspector de propiedades, seleccione Entrada en el menú emergente Tipo de texto, y seleccione la opción Mostrar borde alrededor del texto.
- 3 Cree otro campo de introducción de texto debajo del primero.  
Compruebe que la opción Mostrar borde alrededor del texto no esté seleccionada para este campo de texto. Cree tantos campos de introducción de texto como le parezca conveniente.
- 4 Seleccione Fotograma 1 en la línea de tiempo y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 5 Para crear un objeto que detecte la notificación de selección de la clase `Selection`, introduzca el código siguiente en el panel Acciones:

```
var focusListener = new Object();  
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
```

```

        oldFocus_txt.border = false;
        newFocus_txt.border = true;
    }

```

Este código crea un objeto de `ActionScript` nuevo (genérico) denominado `focusListener`. Este objeto define para sí mismo una propiedad `onSetFocus`, a la cual asigna una función. La función está formada por dos parámetros: una referencia al campo de texto que ha dejado de estar seleccionado y una referencia al campo de texto que pasa a estar seleccionado. La función define la propiedad `border` del campo de texto que ha dejado de estar seleccionado con el valor `false` y define la propiedad `border` del campo que pasa a estar seleccionado con el valor `true`.

- 6 Para registrar el objeto `focusListener` para que reciba eventos del objeto `Selection`, añada el código siguiente al panel Acciones:

```
Selection.addListener(focusListener);
```

- 7 Pruebe la película (Control > Probar película), haga clic en el primer campo de texto y presione la tecla Tabulador para pasar la selección de un campo a otro.

Para anular el registro de un objeto detector de forma que deje de recibir eventos, debe llamar al método `removeListener()` del objeto difusor y pasarle el nombre del objeto detector.

```
broadcastObject.removeListener(listenerObject);
```

Los detectores de eventos están disponibles para los objetos de las siguientes clases de `ActionScript`: `Key`, `Mouse`, `MovieClipLoader`, `Selection`, `TextField` y `Stage`. Para obtener una lista de los detectores de eventos disponibles para cada una de estas clases, consulte las entradas de las clases en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Utilización de controladores de eventos de botones y de clips de película

Puede asociar controladores de eventos directamente a una instancia de botón o de clip de película mediante los controladores `onClipEvent()` y `on()`. El controlador `onClipEvent()` gestiona los eventos de clip de película y el controlador `on()` gestiona los eventos de botón. También puede utilizar `on()` con clips de película para crear clips de película que reciban eventos de botón. Para más información, consulte [“Creación de clips de película con estados de botón” en la página 89](#).

Para utilizar un controlador `on()` o `onClipEvent()`, asícielo directamente a una instancia de botón o de clip de película del escenario y especifique el evento que desea gestionar para dicha instancia. Por ejemplo, el controlador de eventos `on()` siguiente se ejecuta siempre que el usuario hace clic en el botón al que se ha asociado el controlador.

```

on(press){
    trace("Gracias por seleccionarme.");
}

```

Se pueden especificar dos o más eventos para cada controlador `on()`, separados con comas. El `ActionScript` de un controlador se ejecuta cuando se produce uno de los eventos especificados por el controlador. Por ejemplo, el controlador `on()` siguiente asociado a un botón se ejecuta cada vez que el ratón se desplaza sobre el botón o fuera del botón.



```
on(rollOver, rollOut) {
    trace("Se ha desplazado sobre o fuera");
}
```

También puede asociar más de un controlador a un objeto si quiere que se ejecuten scripts diferentes cuando se produzcan eventos diferentes. Por ejemplo, puede asociar los controladores `onClipEvent()` siguientes a la misma instancia de clip de película. El primero se ejecuta cuando el clip de película se carga por primera vez (o aparece en el escenario); el segundo se ejecuta cuando el clip de película se descarga del escenario.

```
onClipEvent (load) {
    trace("Me he cargado");
}
onClipEvent (unload) {
    trace("Me he descargado");
}
```

Para obtener una lista completa de eventos admitidos por los controladores de eventos `on()` y `onClipEvent()`, véase [on\(\)](#) en la página 585 y [onClipEvent\(\)](#) en la página 586.

La gestión de eventos mediante `on()` y `onClipEvent()` no interfiere con la gestión de eventos mediante los métodos de controlador de eventos que defina el usuario. Por ejemplo, imagine que hay un botón en un archivo SWF. El botón puede tener un controlador `on(press)`, que indica al archivo SWF que debe empezar a reproducirse, y el mismo botón puede tener un método `onPress`, para el que se define una función que indica a un objeto del escenario que gire. Cuando se hace clic en el botón, el archivo SWF empieza a reproducirse y el objeto gira. En función de las preferencias del usuario, se pueden utilizar `on()` y `onClipEvent()`, los métodos de controlador de eventos o ambos tipos de gestión de eventos. No obstante, el ámbito de las variables y objetos en los controladores `on()` y `onClipEvent()` es diferente al de los controladores de eventos y los detectores de eventos. (Véase [“Ámbito del controlador de eventos”](#) en la página 90.)

Sólo puede asociar `onClipEvent()` y `on()` a instancias de clip de película que se hayan colocado en el escenario durante la edición. No pueden asociar `onClipEvent()` y `on()` a las instancias de clip de película creadas en tiempo de ejecución (con el método `attachMovie()`, por ejemplo). Para asociar controladores de eventos a objetos creados durante la ejecución, utilice los métodos de controlador de eventos o los detectores de eventos. (Véase [“Utilización de métodos de controlador de eventos”](#) en la página 85 y [“Utilización de detectores de eventos”](#) en la página 87.)

## Creación de clips de película con estados de botón

Cuando se asocia un controlador `on()` a un clip de película o se asigna una función a uno de los controladores de eventos de ratón `MovieClip` para una instancia de clip de película, el clip de película responde a los eventos de ratón del mismo modo que un botón. También se pueden crear estados de botón automáticos (Arriba, Sobre y Abajo) en un clip de película añadiendo las etiquetas de fotograma `_up`, `_over` y `_down` a la línea de tiempo del clip de película.

Cuando el usuario desplaza el ratón por el clip de película o hace clic en él, la cabeza lectora se desplaza al fotograma que lleva la etiqueta de fotograma pertinente. Para designar la zona activa que utiliza un clip de película, utilice la propiedad `hitArea` de la clase `MovieClip`.

**Para crear estados de botón en un clip de película:**

- 1 Seleccione un fotograma de la línea de tiempo de un clip de película para utilizarlo como estado de botón (Arriba, Sobre o Abajo).
- 2 Introduzca una etiqueta de fotograma en el inspector de propiedades (`_up`, `_over` o `_down`).

- 3 Para agregar más estados de botón, repita los pasos 1 y 2.
- 4 Para que el clip de película responda a eventos de ratón, realice una de las acciones siguientes:
  - Asocie un controlador de eventos `on()` a la instancia de clip de película, como se indica en [“Utilización de controladores de eventos de botones y de clips de película” en la página 88.](#)
  - Asigne una función a uno de los controladores de eventos de ratón del objeto del clip de película (`onPress`, `onRelease`, etc.), tal y como se indica en [“Utilización de métodos de controlador de eventos” en la página 85.](#)

## Ámbito del controlador de eventos

El ámbito, o contexto, de las variables y los comandos que se declaran y se ejecutan con un controlador de eventos depende del tipo de controlador de eventos que se esté utilizando: controladores de eventos o detectores de eventos, o bien los controladores `on()` y `onClipEvent()`.

Las funciones asignadas a los métodos de controlador de eventos (como todas las funciones `ActionScript` que se crean) definen un ámbito de variable local, pero los controladores `on()` y `onClipEvent()` no lo hacen.

Por ejemplo, veamos los dos controladores de eventos siguientes. El primero es un controlador de eventos `onPress` asociado a un clip de película denominado `clip_mc`. El segundo es un controlador `on()` asociado a la misma instancia de clip de película.

```
// Asociado a la línea de tiempo del clip principal de clip_mc:
clip_mc.onPress = function () {
    var color; // variable de función local
    color = "blue";
}
// controlador on() asociado a clip_mc:
on(press){
    var color; // ámbito de variable no local
    color = "blue";
}
```

Aunque los dos controladores de evento tienen el mismo código, los resultados pueden ser diferentes. En el primer caso, la variable `color` es local para la función definida para `onPress`. En el segundo caso, dado que el controlador `on()` no define un ámbito de variable local, el ámbito de la variable pasa a ser la línea de tiempo del clip de película `clip_mc`.

Para los controladores de eventos `on()` asociados a los botones, en lugar de los clips de películas, las variables (así como las llamadas de función y método) se encuentran en el ámbito de la línea de tiempo que contiene la instancia de botón.

Por ejemplo, el controlador de eventos `on()` siguiente producirá un resultado diferente según si se ha asociado a un objeto de clip de película o de botón. En el primer caso, la llamada a la función `play()` pone en marcha la cabeza lectora en la línea de tiempo que contiene el botón; en el segundo caso, la llamada a la función `play()` inicia la línea de tiempo del clip de película al cual está asociado el controlador.

```
// Asociado al botón
on(press){
    play(); // reproduce la línea de tiempo principal
}
// Asociado al clip de película
on(press){
    play(); // reproduce la línea de tiempo del clip de película
}
```

Es decir, cuando está asociado a un objeto de botón, la llamada de método `play()` se aplica a la línea de tiempo que contiene el botón, es decir, la línea de tiempo principal del botón. Pero si el mismo controlador está asociado a un objeto de clip de película, `play()` se aplica al clip de película al que se ha asignado el controlador.

Dentro de una definición de función de controlador de eventos o de detector de eventos, se aplicaría la misma función `play()` a la línea de tiempo que contiene la definición de función. Por ejemplo, supongamos que la función de controlador de eventos `MovieClip.onPress` siguiente se declarase en la línea de tiempo que contiene la instancia de clip de película `myMovieClip`.

```
// Función definida en la línea de tiempo del clip de película:
myMovieClip.onPress = function () {
    play(); // reproduce la línea de tiempo que contiene la definición de la
    función
}
```

Si desea reproducir el clip de película que define el controlador de eventos `onPress`, haga referencia explícita a ese clip mediante la palabra clave `this`, del modo siguiente:

```
myMovieClip.onPress = function () {
    this.play(); // reproduce la línea de tiempo de un clip que define el
    controlador onPress
}
```

## Ámbito de la palabra clave “this”

La palabra clave `this` hace referencia al objeto del ámbito de ejecución actual. Según el tipo de técnica de controlador de eventos que utilice, `this` puede hacer referencia a distintos objetos.

**Dentro de una función de controlador de eventos o de detector de eventos**, `this` hace referencia al objeto que define el método de controlador o detector de eventos. Por ejemplo, en el código siguiente, `this` hace referencia al propio `myClip`.

```
// controlador de eventos onPress() asociado a _level0.myClip:
myClip.onPress = function () {
    trace(this); // muestra '_level0.myClip'
}
```

**Dentro de un controlador `on()` asociado a un clip de película**, `this` hace referencia al clip de película al que está asociado el controlador `on()`.

```
// Asociado al clip de película llamado 'myClip'
on(press){
    trace(this); muestra '_level0.myClip'
}
```

**Dentro de un controlador** `on()` **asociado a un botón**, `this` hace referencia a la línea de tiempo que contiene el botón.

```
// Asociado al botón en la línea de tiempo principal
on(press){
  trace(this); // muestra '_level0'
}
```

# CAPÍTULO 5

## Creación de interacciones con ActionScript

En animaciones sencillas, Macromedia Flash Player reproduce las escenas y los fotogramas de un archivo SWF de forma secuencial. En un archivo SWF interactivo, los usuarios utilizan el teclado y el ratón para ir a distintas partes del archivo SWF, mover objetos, introducir información en formularios y llevar a cabo otras operaciones interactivas.

ActionScript sirve para crear scripts que indican a Flash Player la acción que debe llevar a cabo cuando ocurra un evento. Algunos eventos que desencadenan un script se producen cuando la cabeza lectora llega a un fotograma, cuando se carga o descarga un clip de película, o cuando el usuario hace clic en un botón o presiona una tecla.

Los scripts pueden constar de un solo comando, como indicar a un archivo SWF que se detenga, o bien de una serie de comandos y sentencias, como comprobar primero una condición y, a continuación, realizar una acción. Muchos comandos de ActionScript son sencillos y permiten crear controles básicos para un archivo SWF. Otras requieren un cierto dominio de los lenguajes de programación y están pensadas para operaciones avanzadas.

### Eventos e interacciones

Siempre que un usuario hace clic en el ratón o pulsa una tecla, se genera un evento. Estos tipos de eventos suelen denominarse *eventos de usuario*, ya que se generan como respuesta a alguna acción llevada a cabo por el usuario final. Puede escribir código de ActionScript para responder a estos eventos o *gestionarlos*. Por ejemplo, cuando un usuario hace clic en un botón, puede enviar la cabeza lectora a otro fotograma del archivo SWF o cargar una nueva página Web en el navegador.

En un archivo SWF, los botones, los clips de película y los campos de texto generan eventos a los que se puede responder. ActionScript dispone de tres modos de gestionar eventos: métodos de controlador de eventos, detectores de eventos y controladores `on()` y `onClipEvent()`. Para más información sobre eventos y la gestión de los mismos, consulte el [Capítulo 4, “Gestión de eventos”](#), en la [página 85](#).

### Control de la reproducción de archivos SWF

Las funciones de ActionScript siguientes permiten controlar la cabeza lectora en la línea de tiempo y cargar una página Web en una ventana del navegador:

- Las funciones `gotoAndPlay()` y `gotoAndStop()` envían la cabeza lectora a otro fotograma o escena. Estas son las funciones globales a las que puede llamar desde cualquier script. También puede utilizar los métodos `MovieClip.gotoAndPlay()` y `MovieClip.gotoAndStop()` para desplazarse por la línea de tiempo de un objeto de clip de película específico.
- Las acciones `play()` y `stop()` reproducen y detienen películas.
- La acción `gotoAndPlay()` lleva a otra URL.

## Salto a fotogramas o escenas

Para ir a una escena o un fotograma específicos del archivo SWF, puede utilizar las funciones globales `gotoAndPlay()` y `gotoAndStop()` o los métodos `gotoAndPlay()` y `gotoAndStop()` equivalentes de la clase `MovieClip`. Cada función o método permite especificar un fotograma de la escena actual. Si el documento contiene varias escenas, puede especificar una escena y un fotograma al que desplazarse.

En el ejemplo siguiente se utiliza la función global `gotoAndPlay()` en el controlador de eventos `onRelease` de un objeto de botón para enviar la cabeza lectora de la línea de tiempo que contiene el botón al fotograma 10.

```
jump_btn.onRelease = function () {
    gotoAndPlay(10);
}
```

En el ejemplo siguiente, el método `MovieClip.gotoAndStop()` envía la línea de tiempo de un clip de película llamado `categories_mc` al fotograma 10 y se detiene. Cuando se utilizan los métodos `MovieClip gotoAndPlay()` y `gotoAndStop()`, debe especificar una instancia para cada método.

```
jump_btn.onPress = function () {
    categories_mc.gotoAndStop(10);
}
```

## Cómo reproducir y detener clips de película

A menos que se indique lo contrario, una vez que se inicia un archivo SWF se reproduce en todo el fotograma de la línea de tiempo. Puede detener o iniciar un archivo SWF mediante las funciones globales `play()` y `stop()` o los métodos `MovieClip` equivalentes. Por ejemplo, puede utilizar la función `stop()` para detener un archivo SWF al final de una escena antes de continuar con la escena siguiente. Una vez detenido el archivo, debe volver a iniciarse de forma explícita llamando a la función `play()`.

Puede utilizar las funciones `play()` y `stop()` o los métodos `MovieClip` para controlar la línea de tiempo principal o la línea de tiempo de cualquier clip de película o archivo SWF cargado. El clip de película que desea controlar debe tener un nombre de instancia y, además, debe estar presente en la línea de tiempo.

El controlador `on(press)` siguiente que se encuentra asociado a un botón inicia el movimiento de la cabeza lectora del archivo SWF o del clip de película que contiene el objeto de botón.

```
// Asociado a una instancia de botón
on(press){
    // Reproduce la línea de tiempo que contiene el botón
    play();
}
```

El mismo código del controlador de eventos `on()` tendrá un resultado diferente al asociarlo a un objeto de clip de película, en lugar de a un botón. Al asociar dicho código a un objeto de botón, las sentencias que se creen en un controlador `on()` se aplican, de forma predeterminada, a la línea de tiempo que contiene el botón. Sin embargo, al asociarlas a un objeto de clip de película, las sentencias realizadas en un controlador `on()` se aplican al clip de película al que se encuentra asociado al controlador `on()`.

Por ejemplo, el código del controlador `on()` detiene la línea de tiempo del clip de película al que se encuentra asociado el controlador, no la línea de tiempo que contiene el clip de película.

```
on(press){
    stop();
}
```

Las mismas condiciones se aplican a los controladores `onClipEvent()` que se encuentran asociados a objetos de clip de película. Por ejemplo, el código siguiente detiene la línea de tiempo del clip de película que contiene el controlador `onClipEvent()` cuando el clip se carga por primera vez o cuando aparece en el escenario.

```
onClipEvent (load) {
    stop();
}
```

## Salto a otra URL

Para abrir una página Web en una ventana del navegador o para transferir datos a otra aplicación en una URL definida, puede utilizar la función global `getURL()` o el método `MovieClip.getURL()`. Por ejemplo, puede disponer de un botón que lleve a un nuevo sitio Web, o bien enviar variables de línea de tiempo a un script CGI para procesarlo igual que se procesaría un formulario HTML. También puede especificar la ventana que desea utilizar, del mismo modo que lo haría al elegir una ventana para usar con una etiqueta de anclaje HTML (`<a></a>`).

Por ejemplo, el código siguiente abre la página principal de macromedia.com en una ventana en blanco del navegador cuando el usuario hace clic en la instancia de botón llamada `homepage_btn`.

```
homepage_btn.onRelease = function () {
    getURL("http://www.macromedia.com", "_blank");
}
```

También puede enviar variables junto con la URL, mediante GET o POST. Esto resulta útil si la página que está cargando desde un servidor de aplicaciones, como la página del servidor ColdFusion (CFM), espera recibir variables de formulario. Por ejemplo, suponga que desea cargar una página de CFM llamada `addUser.cfm` que espera obtener dos variables de formulario: `name` y `age`. Para ello, puede crear un clip de película llamado `variables_mc` que defina dichas variables tal como se muestra a continuación.

```
variables_mc.name = "Francois";
variables_mc.age = 32;
```

A continuación, el código siguiente carga `addUser.cfm` en una ventana en blanco del navegador y pasa las variables `variables_mc.name` y `variables_mc.age` del encabezado POST a la página de CFM.

```
variables_mc.getURL("addUser.cfm", "_blank", "POST");
```

Para más información, consulte [getURL\(\) en la página 400](#).

## Creación de interactividad y efectos visuales

Para crear interactividad y otros efectos visuales, debe entender las técnicas siguientes:

- [Creación de un puntero de ratón personalizado](#)
- [Obtención de la posición del ratón](#)
- [Captura de teclas presionadas](#)
- [Configuración de valores de color](#)
- [Creación de controles de sonido](#)
- [Detección de colisiones](#)
- [Creación de una herramienta sencilla de dibujo de líneas](#)

### Creación de un puntero de ratón personalizado

Un puntero de ratón estándar es la representación en la pantalla del sistema de la posición del ratón del usuario. Al reemplazar el puntero de ratón estándar por uno diseñado en Flash, puede integrar el movimiento del ratón del usuario en el archivo SWF con mayor precisión. En el ejemplo de esta sección se utiliza un puntero personalizado que parece una flecha grande. La clave de esta función, sin embargo, radica en la capacidad de convertir el puntero personalizado en cualquier cosa: por ejemplo, un balón que debe llevarse a la portería o una muestra de tela que se coloca sobre un sofá para cambiarle el color.

Para crear un puntero personalizado, diseñe el clip de película de puntero en el escenario. A continuación, en ActionScript, oculte el puntero estándar y realice un seguimiento de su movimiento. Para ocultar el puntero estándar, utilice el método `Mouse.hide()` de la clase `Mouse` incorporada. Para utilizar un clip de película como puntero personalizado, utilice la acción `startDrag()`.

#### Para crear un puntero personalizado:

- 1 Cree un clip de película para utilizarlo como puntero personalizado y coloque una instancia del clip en el escenario.
- 2 Seleccione el clip de película en el escenario.
- 3 Seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en el caso de que no aparezca en pantalla.
- 4 En este panel, escriba lo siguiente:

```
onClipEvent (load) {  
    Mouse.hide();  
    startDrag(this, true);  
}  
onClipEvent(mouseMove){  
    updateAfterEvent();  
}
```

El primer controlador `onClipEvent()` oculta el ratón cuando el clip de película aparece por primera vez en el escenario. El segundo controlador llama a la función `updateAfterEvent` siempre que el usuario mueve el ratón.

La función `updateAfterEvent` actualiza la pantalla inmediatamente después de que se haya producido el evento especificado, en lugar de cuando se dibuja el fotograma siguiente, que es lo que suele ocurrir. (Véase [updateAfterEvent\(\)](#) en la página 744.)

- 5 Seleccione Control > Probar película para probar el puntero personalizado.

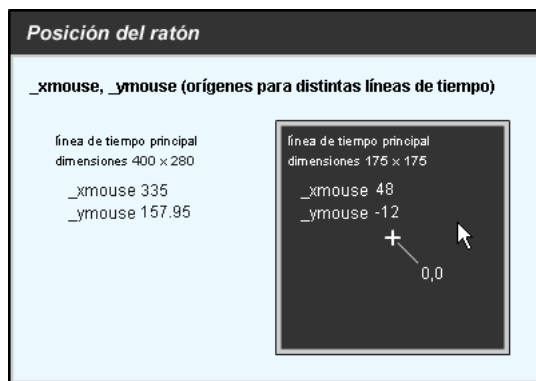


Los botones continúan funcionando al utilizar un puntero personalizado. Se recomienda colocar el puntero personalizado en la capa superior de la línea de tiempo de modo que se mueva por encima de los botones y de otros objetos cuando desplace el ratón por el archivo SWF. Además, la “punta” de un puntero personalizado es el punto de registro del clip de película que se utiliza como puntero personalizado. En consecuencia, si desea que una parte del clip de película actúe como la punta del ratón, configure las coordenadas del punto de registro del clip para que sea ese punto.

Para más información sobre los métodos de la clase `Mouse`, consulte la entrada [Clase Mouse](#) en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Obtención de la posición del ratón

Puede utilizar la propiedad `_xmouse` y la propiedad `_ymouse` para determinar la ubicación del puntero del ratón (cursor) en un archivo SWF. Cada línea de tiempo tiene una propiedad `_xmouse` e `_ymouse` que indican la ubicación del ratón dentro de su sistema de coordenadas. La posición siempre es relativa al punto de registro. Para la línea de tiempo principal (`_level0`), el punto de registro corresponde a la esquina superior izquierda.



*Propiedades `_xmouse` e `_ymouse` de la línea de tiempo principal y de la línea de tiempo de un clip de película*

En los siguientes procedimientos se muestran dos modos para obtener la posición del ratón.

### Para obtener la posición del ratón en la línea de tiempo principal:

- 1 Cree dos cuadros de texto dinámicos y asígneles los nombres `x_pos` e `y_pos`.
- 2 Seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en el caso de que no aparezca en pantalla.
- 3 Para volver a la posición del ratón dentro de la línea de tiempo principal, agregue el código siguiente a cualquier fotograma del archivo SWF `_level0`:

```
x_pos = _root._xmouse;  
y_pos = _root._ymouse;
```

Las variables `x_pos` e `y_pos` se utilizan como contenedores para guardar los valores de las posiciones del ratón. Puede utilizar estas variables en cualquier script del documento. En el controlador `onClipEvent()` siguiente, los valores de `x_pos` e `y_pos` se actualizan cada vez que el usuario mueve el ratón.

```
onClipEvent(mouseMove){
    x_pos = _root._xmouse;
    y_pos = _root._ymouse;
}
```

**Para obtener la posición actual del ratón dentro de un clip de película:**

- 1 Cree un clip de película.
- 2 Seleccione la instancia de clip de película en el escenario. Asígnele el nombre `myMovieClip` con el inspector de propiedades.
- 3 Seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en el caso de que no aparezca en pantalla.
- 4 Utilice el nombre de instancia del clip de película para volver a la posición del ratón dentro de la línea de tiempo principal.

Por ejemplo, la siguiente sentencia podría colocarse en cualquier línea de tiempo del archivo SWF `_level0` para que devuelva la posición `_ymouse` en la instancia `myMovieClip`:

```
x_pos = _root.myMovieClip._xmouse
y_pos = _root.myMovieClip._ymouse
```

El código devuelve las posiciones `_xpos` e `_ypos` del ratón con relación al punto de registro.

- 5 Seleccione Control > Probar película para probar la película.

También puede determinar la posición del ratón en un clip de película utilizando las propiedades `_xmouse` e `_ymouse` de un evento de clip, como se muestra en el código siguiente:

```
onClipEvent(enterFrame) {
    xmousePosition = this._xmouse;
    ymousePosition = this._ymouse;
}
```

Para más información sobre las propiedades `_xmouse` e `_ymouse`, consulte [MovieClip.\\_xmouse en la página 544](#) y [MovieClip.\\_ymouse en la página 546](#).

## Captura de teclas presionadas

Puede utilizar los métodos de la clase incorporada `Key` para detectar la última tecla presionada por el usuario. La clase `Key` no requiere una función constructora; para utilizar sus métodos, simplemente llame a la clase propiamente dicha, como se muestra en el ejemplo siguiente:

```
Key.getCode();
```

Puede obtener códigos de tecla virtuales o valores ASCII (código americano estándar para intercambio de información) de la tecla que haya sido presionada:

- Para obtener el código de tecla virtual de la última tecla presionada, utilice el método `getCode()`.
- Para obtener el valor ASCII de la última tecla presionada, utilice el método `getAscii()`.

Se asigna un código de tecla virtual a cada tecla física del teclado. Por ejemplo, la tecla de flecha izquierda tiene el código de tecla virtual 37. Al utilizar un código de tecla virtual, se garantiza que los controles del archivo SWF sean los mismos en todos los teclados, independientemente del idioma o de la plataforma.

Los valores ASCII se asignan a los primeros 127 caracteres de cada conjunto de caracteres. Los valores ASCII proporcionan información sobre un carácter de la pantalla. Por ejemplo, la letra “A” y la letra “a” tiene diferentes valores ASCII.

Para decidir qué teclas va a utilizar y determinar sus códigos virtuales, utilice alguno de los siguientes métodos:

- Consulte la lista de códigos de tecla del [Apéndice C, “Teclas del teclado y valores de códigos de tecla”, en la página 791](#).
- Utilice una constante de la clase Key. En el cuadro de herramientas Acciones, haga clic en la categoría Clases incorporadas, en Película, en Tecla y en Constantes.
- Asigne el controlador `onClipEvent()` siguiente a un clip de película y, a continuación, seleccione Control > Probar película y presione la tecla que desee.

```
onClipEvent(keyDown) {  
    trace(Key.getCode());  
}
```

El código de la tecla deseada aparece en el panel Salida.

Un lugar habitual para utilizar los métodos de la clase Key es dentro de un controlador de eventos. En el ejemplo siguiente, el usuario mueve el coche mediante las teclas de flecha. El método `Key.isDown()` indica si la tecla que se está presionando es la flecha derecha, izquierda, hacia arriba o hacia abajo. El controlador de eventos, `onEnterFrame`, determina el valor `Key.isDown(keyCode)` de las sentencias `if`. En función del valor, el controlador da instrucciones a Flash Player para que actualice la posición del coche y para que muestre la dirección.



*Al presionar las teclas del teclado, el coche se mueve.*

El procedimiento siguiente muestra cómo se pueden capturar teclas presionadas para mover un clip de película hacia arriba, abajo, la izquierda y la derecha en el escenario, en función de la tecla de flecha correspondiente que se presione (arriba, abajo, izquierda o derecha). El clip de película se limita a un área arbitraria de 400 píxeles de ancho y 300 píxeles de alto. Asimismo, un campo de texto muestra el nombre de la tecla que se presiona.

**Para crear un clip de película activado por teclado:**

- 1 En el escenario, cree un clip de película que se moverá como respuesta a las flechas del teclado. En este ejemplo, el nombre de la instancia de clip de película es `car`.
- 2 En el escenario, cree un cuadro de texto dinámico que se actualizará con la dirección del coche. Asígnele el nombre de instancia `display_txt` mediante el inspector de propiedades.

**Nota:** no confunda los nombres de variable con los nombres de instancia. Para más información, consulte [“Nombres de instancia y de variable de los campos de texto” en la página 140](#).

- 3 Seleccione el fotograma 1 en la línea de tiempo; a continuación, seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en caso de que todavía no aparezca en pantalla.

- 4 Para establecer cuánto avanza el coche en la pantalla cada vez que se presiona una tecla, defina una variable `distance` y establezca el valor inicial de la misma en 10.

```
var distance = 10;
```

- 5 Para crear un controlador de eventos para el clip de película del coche que compruebe qué tecla de flecha (izquierda, derecha, arriba o abajo) se encuentra presionada, añada el código siguiente en el panel Acciones:

```
car.onEnterFrame = function() {  
  
}
```

- 6 Añada una sentencia `with` al cuerpo del controlador `onEnterFrame` y especifique `car` como objeto de la sentencia `with`.

El código deberá ser similar al siguiente:

```
var distance = 10;  
car.onEnterFrame = function() {  
    with (car) {  
    }  
}
```

- 7 Para comprobar que está presionando la tecla de flecha derecha y para mover el clip de película del coche en esa dirección, añada el código al cuerpo de la sentencia `with`. El código deberá ser similar al siguiente:

```
distance = 10;  
car.onEnterFrame = function() {  
    with (car) {  
        if (Key.isDown(Key.RIGHT)) {  
            _x += distance;  
            if (_x >= 400) {  
                _x = 400;  
            }  
            _root.display_txt.text = "Right";  
        }  
    }  
}
```

Si la tecla de flecha derecha está presionada, la propiedad `_x` del coche aumentará según la cantidad especificada por la variable `distance`. La sentencia `if` siguiente comprueba si el valor de la propiedad `_x` del clip es mayor o igual que 400 (`if(_x >=400)`); en caso de que esto ocurra, su posición se fija en 400. Asimismo, la palabra *Right* debe aparecer en el archivo SWF.

- 8 Utilice un código similar para comprobar si se está presionando la tecla de flecha izquierda, derecha, arriba o abajo. El código deberá ser similar al siguiente:

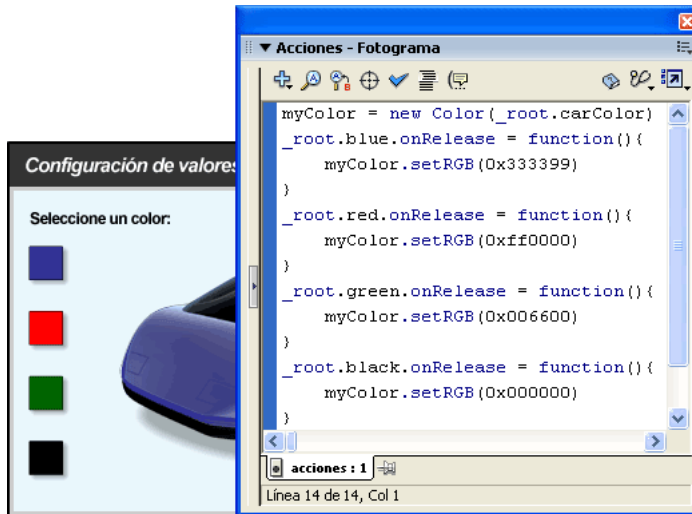
```
var distance = 10;
car.onEnterFrame = function() {
    with (car) {
        if (Key.isDown(Key.RIGHT)) {
            _x += distance;
            if (_x >= 400) {
                _x = 400;
            }
            _root.display_txt.text = "Right";
        } else if (Key.isDown(Key.LEFT)) {
            _x -= distance;
            if (_x < 0) {
                _x = 0;
            }
            _root.display_txt.text = "Left";
        } else if (Key.isDown(Key.UP)) {
            _y -= distance;
            if (_y < 0) {
                _y = 0 ;
            }
            _root.display_txt.text = "Up";
        } else if (Key.isDown(Key.DOWN)) {
            _y += distance;
            if (_y > 300) {
                _y = 300;
            }
            _root.display_txt.text = "Down";
        }
    }
}
```

- 9 Seleccione Control > Probar película para probar el archivo.

Para más información sobre los métodos de la clase `Key`, consulte la entrada [Clase Key](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

## Configuración de valores de color

Puede utilizar los métodos de la clase incorporada `Color` para ajustar el color de un clip de película. El método `setRGB()` asigna valores hexadecimales RGB (rojo, verde, azul) al clip de película. En el siguiente ejemplo se utiliza `setRGB` para cambiar el color de un objeto en función de la información introducida por el usuario.



*La acción del botón crea un objeto `Color` y cambia el color del coche en función de la información introducida por el usuario.*

### Para establecer el valor de color de un clip de película:

- 1 Seleccione un clip de película en el escenario.
- 2 En el inspector de propiedades, introduzca `carColor` como nombre de instancia.
- 3 Cree un botón denominado `color chip`, coloque cuatro instancias del botón en el escenario y asígneles los nombres `red`, `green`, `blue` y `black`.
- 4 Seleccione el fotograma 1 en la línea de tiempo principal y seleccione **Ventana > Paneles de desarrollo > Acciones**.
- 5 Para crear un objeto `Color` que se centre en el clip de película `carColor`, añada el código siguiente en el panel Acciones:  

```
myColor = new Color(_root.carColor);
```
- 6 Para hacer que el botón de color azul cambie el color de clip de película `carColor` a azul, añada el código siguiente al panel Acciones:  

```
_root.blue.onRelease = function(){
    myColor.setRGB(0x0000ff)
}
```

El valor hexadecimal `0x0000ff` es azul. En la tabla siguiente se muestran los demás colores que utilizará y sus valores hexadecimales.

- 7 Repita el paso 6 para los demás botones (red, green y black) para cambiar el color del clip de película al color que corresponda. El código debe parecerse al siguiente:

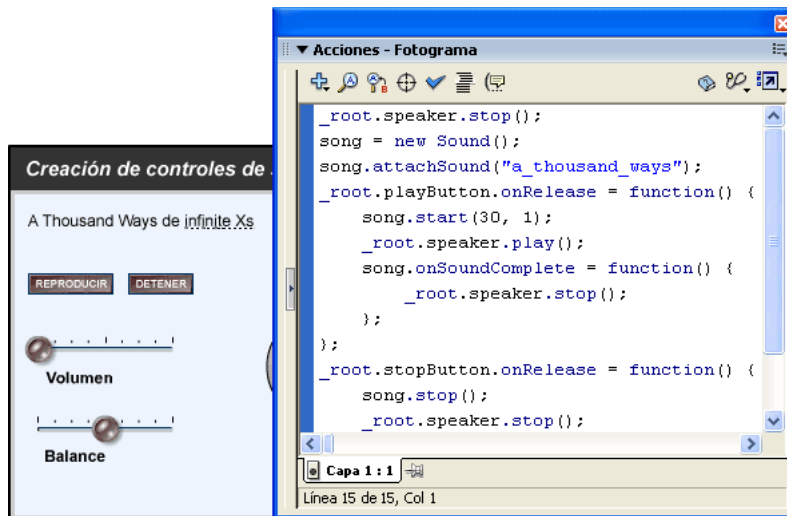
```
myColor = new Color(_root.carColor)
_root.blue.onRelease = function(){
    myColor.setRGB(0x0000ff)
}
_root.red.onRelease = function(){
    myColor.setRGB(0xff0000)
}
_root.green.onRelease = function(){
    myColor.setRGB(0x00ff00)
}
_root.black.onRelease = function(){
    myColor.setRGB(0x000000)
}
```

- 8 Seleccione Control > Probar película para cambiar el color del clip de película.

Para más información sobre los métodos de la clase Color, consulte la entrada [Clase Color](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

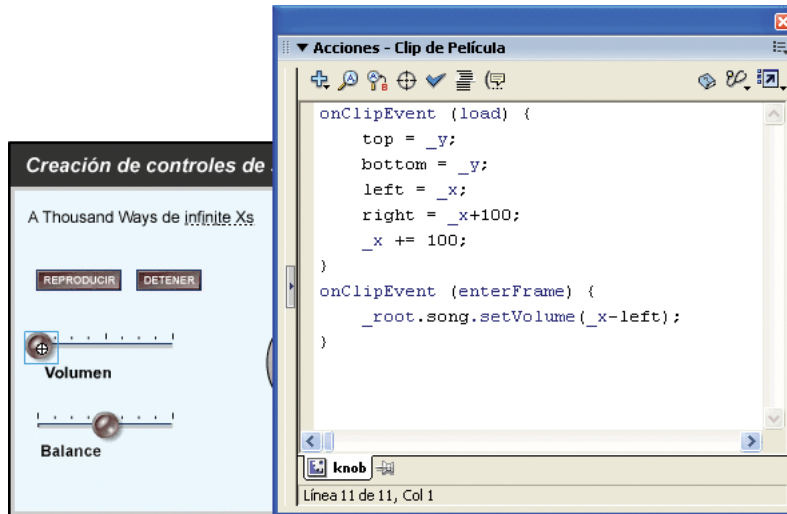
## Creación de controles de sonido

Utilice la clase incorporada Sound para controlar los sonidos de un archivo SWF. Para utilizar los métodos de la clase Sound, deberá crear primero un nuevo objeto Sound. A continuación, puede utilizar el método `attachSound()` para insertar un sonido de la biblioteca en un archivo SWF mientras el archivo SWF se está reproduciendo.



*Cuando el usuario suelta el botón Reproducir, se reproduce una canción por el altavoz.*

El método `setVolume()` de la clase `Sound` controla el volumen y el método `setPan()` ajusta el balance izquierdo y derecho de un sonido.



*Cuando el usuario arrastra el deslizador de volumen, se llama al método `setVolume()`.*

En los procedimientos siguientes se muestra cómo crear controles de sonido como los que se muestran más arriba.

#### **Asignación de un sonido a una línea de tiempo:**

- 1 Seleccione Archivo > Importar para importar un sonido.
- 2 Seleccione el sonido en la biblioteca, haga clic con el botón derecho del ratón (Windows) o con la tecla Control (Macintosh) y seleccione Vinculación.
- 3 Seleccione Exportar para ActionScript y Exportar en primer fotograma; a continuación, asígnele el identificador `a_thousand_ways`.
- 4 Agregue un botón al escenario y asígnele el nombre `playButton`.
- 5 Agregue un botón al escenario y asígnele el nombre `stopButton`.
- 6 Agregue un clip de película al escenario y asígnele el nombre `speaker`.
- 7 Seleccione el fotograma 1 en la línea de tiempo principal y seleccione Ventana > Paneles de desarrollo > Acciones. Añada el código siguiente al panel Acciones:

```
speaker.stop();
song = new Sound();
song.onSoundComplete = function() {
    speaker.stop();
};
song.attachSound("a_thousand_ways");
playButton.onRelease = function() {
    song.start();
    speaker.play();
};
stopButton.onRelease = function () {
    song.stop();
};
```



```

    speaker.stop();
}

```

Este código primero detiene el clip de película `speaker`. A continuación, crea un objeto `Sound` (`song`) nuevo y lo asocia al sonido cuyo identificador de vínculo es `a_thousand_ways`. Después, define un controlador `onSoundComplete` para el objeto `song`, el cual detiene el clip de película `speaker` una vez que el sonido ha finalizado. Por último, los controladores `onRelease` asociados con los objetos `playButton` y `stopButton` inician y detienen el sonido mediante los métodos `Sound.start()` y `Sound.stop()`. Asimismo, reproducen y detienen el clip de película `speaker`.

- 8 Seleccione Control > Probar película para oír el sonido.

**Para crear un control deslizable de volumen:**

- 1 Arrastre un botón al escenario.
- 2 Seleccione el botón y luego Modificar > Convertir en símbolo. Asegúrese de elegir el comportamiento del clip de película.

Esta acción crea un clip de película con el botón en su primer fotograma.

- 3 Seleccione el clip de película y, a continuación, Edición > Editar seleccionado.
- 4 Seleccione el botón y, a continuación, Ventana > Paneles de desarrollo > Acciones.
- 5 Introduzca las siguientes acciones:

```

on(press){
    startDrag(this, false, left, top, right, bottom);
}
on(release) {
    stopDrag();
}

```

Los parámetros de `startDrag()` `left`, `top`, `right` y `bottom` son variables definidas en una acción de clip.

- 6 Seleccione Edición > Editar documento para volver a la línea de tiempo principal.
- 7 Seleccione el clip de película en el escenario.
- 8 Introduzca las siguientes acciones:

```

onClipEvent (load) {
    top = _y;
    bottom = _y;
    left = _x;
    right = _x+100;
    _x += 100;
}
onClipEvent (enterFrame) {
    _parent.song.setVolume(_x-left);
}

```

- 9 Seleccione Control > Probar película para utilizar el deslizador de volumen.

**Para crear un control de deslizador de balance:**

- 1 Arrastre un botón al escenario.
- 2 Seleccione el botón y, a continuación, Modificar > Convertir en símbolo. Seleccione una propiedad de clip de película.
- 3 Seleccione el clip de película y, a continuación, Edición > Editar símbolos.
- 4 Seleccione el botón y, a continuación, Ventana > Paneles de desarrollo > Acciones.

5 Introduzca las siguientes acciones:

```
on(press){
    startDrag ("", false, left, top, right, bottom);
    dragging = true;
}
on (release, releaseOutside) {
    stopDrag ();
    dragging = false;
}
```

Los parámetros de `startDrag()` `left`, `top`, `right` y `bottom` son variables definidas en una acción de clip.

6 Seleccione Edición > Editar documento para volver a la línea de tiempo principal.

7 Seleccione el clip de película en el escenario.

8 Introduzca las siguientes acciones:

```
onClipEvent (load) {
    top=_y;
    bottom=_y;
    left=_x-50;
    right=_x+50;
    center=_x;
}

onClipEvent(enterFrame) {
    if (dragging==true){
        _parent.setPan((_x-center)*2);
    }
}
```

9 Seleccione Control > Probar película para utilizar el deslizador de balance.

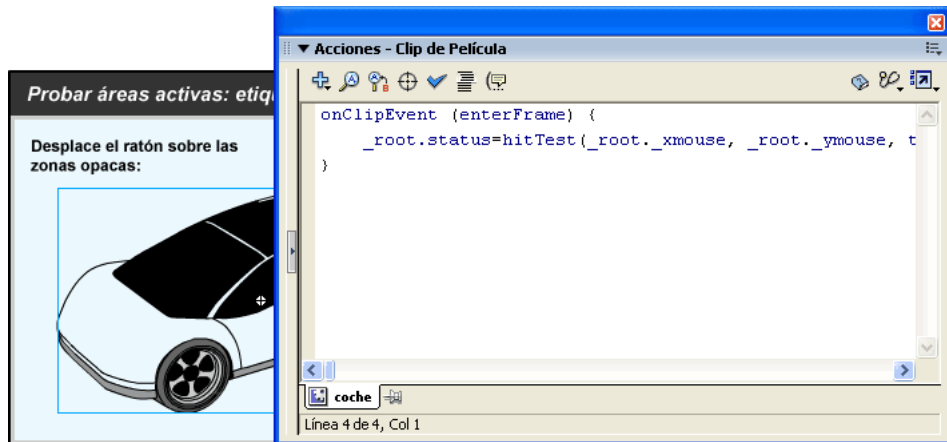
Para más información sobre los métodos de la clase `Sound`, consulte la entrada [Clase Sound](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

## Detección de colisiones

El método `hitTest()` de la clase `MovieClip` detecta las colisiones dentro de un archivo SWF. Comprueba si el objeto ha colisionado con un clip de película y devuelve un valor booleano (`true` o `false`).

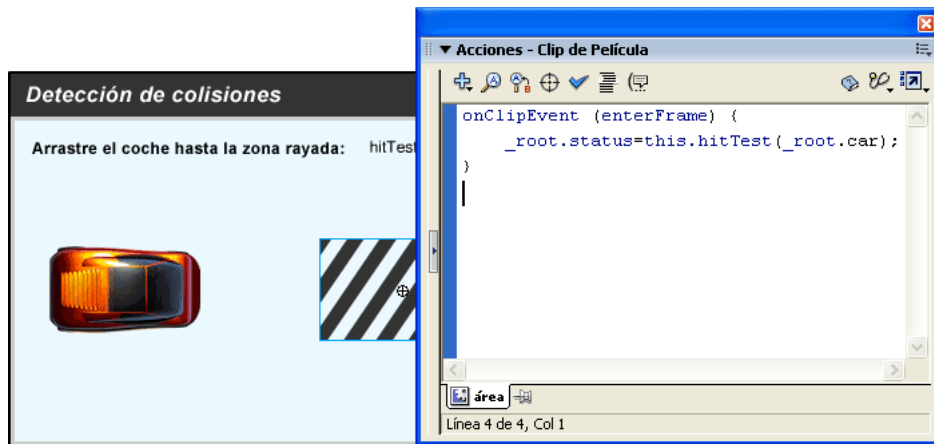
Existen dos casos en los que debe saber si se ha producido una colisión: para probar si el usuario ha llegado a una cierta área estática del escenario y para determinar cuándo un clip de película ha alcanzado a otro. Con el método `hitTest()`, puede determinar estos resultados.

Puede utilizar los parámetros del método `hitTest()` para especificar las coordenadas  $x$  e  $y$  de una zona activa en el escenario o utilizar la ruta de destino de otro clip de película como zona activa. Al especificar  $x$  e  $y$ , `hitTest()` devuelve `true` si el punto identificado por  $(x, y)$  no es un punto transparente. Cuando se pasa un destino a `hitTest()`, se comparan los recuadros de delimitación de los dos clips de película. Si se solapan, `hitTest()` devuelve `true`. Si los dos cuadros no tienen ningún punto de intersección, `hitTest()` devuelve `false`.



*El campo de texto muestra "True" siempre que el puntero del ratón se encuentra sobre el coche*

También puede utilizar el método `hitTest()` para probar la colisión entre dos clips de película.



*El campo de texto muestra "True" siempre que un clip de película está en contacto con el otro.*

En los siguientes procedimientos se muestra cómo se detecta la colisión mediante el ejemplo del coche.

### Para detectar una colisión entre un clip de película y un punto del escenario:

- 1 Cree un clip de película nuevo en el escenario e introduzca `box` como nombre de instancia en el inspector de propiedades.
- 2 Cree un cuadro de texto dinámico en el escenario e introduzca `status` como nombre de instancia en el inspector de propiedades.
- 3 Seleccione el primer fotograma de la Capa 1 en la línea de tiempo.
- 4 Seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en el caso de que no aparezca en pantalla.
- 5 Introduzca el código siguiente en el panel Acciones:

```
box.onEnterFrame = function () {  
    status.text = this.hitTest(_xmouse, _ymouse, true);  
}
```

- 6 Seleccione Control > Probar película y mueva el ratón sobre el clip de película para comprobar la colisión.

El valor `true` se visualiza siempre que el ratón se encuentra sobre un píxel que no es transparente.

### Para detectar una colisión en dos clips de película:

- 1 Arrastre dos clips de película al escenario y asígneles los nombres de instancia `car` y `area`.
- 2 Cree un cuadro de texto dinámico en el escenario e introduzca `status` como nombre de instancia en el inspector de propiedades.
- 3 Seleccione el primer fotograma de la Capa 1 en la línea de tiempo.
- 4 Seleccione Ventana > Paneles de desarrollo > Acciones para abrir el panel Acciones en el caso de que no aparezca en pantalla.
- 5 Introduzca los códigos siguientes en el panel Acciones:

```
area.onEnterFrame = function () {  
    status.text=this.hitTest(car);  
}  
car.onPress = function () {  
    this.startDrag(false);  
    updateAfterEvent();  
}  
car.onRelease = function () {  
    this.stopDrag();  
}
```

- 6 Seleccione Control > Probar película y arrastre el clip de película para comprobar la detección de una colisión.

Siempre que el recuadro de delimitación del coche tenga un punto de intersección con el recuadro de delimitación del área, el estado es `true`.

Para más información, consulte [MovieClip.hitTest\(\)](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

## Creación de una herramienta sencilla de dibujo de líneas

Puede utilizar los métodos de la clase `MovieClip` para dibujar líneas y rellenos en el escenario mientras se reproduce el archivo SWF. Esto permite crear herramientas de dibujo para los usuarios y dibujar formas en el archivo SWF como respuesta a los eventos. Los métodos de dibujo son `beginFill()`, `beginGradientFill()`, `clear()`, `curveTo()`, `endFill()`, `lineTo()`, `lineStyle()` y `moveTo()`. Puede aplicar estos métodos a cualquier instancia de clip de película (por ejemplo, `myClip.lineTo()`) o a un nivel (`_root.curveTo()`).

Los métodos `lineTo()` y `curveTo()` permiten dibujar líneas y curvas, respectivamente. Utilice el método `lineStyle()` para especificar el color de línea, el grosor y el parámetro alfa para una línea o curva. El método de dibujo `moveTo()` establece la posición del dibujo actual en las coordenadas de escenario *x* e *y* que especifique.

Los métodos `beginFill()` y `beginGradientFill()` rellenan un trazado cerrado con un relleno degradado o sólido respectivamente, y `endFill()` aplica el relleno especificado en la última llamada a `beginFill()` o a `beginGradientFill()`. El método `clear()` elimina lo que se haya dibujado en el objeto del clip de película especificado.

Para más información, consulte `MovieClip.beginFill()` en la página 494, `MovieClip.beginGradientFill()` en la página 495, `MovieClip.clear()` en la página 498, `MovieClip.curveTo()` en la página 501, `MovieClip.endFill()` en la página 504, `MovieClip.lineTo()` en la página 516, `MovieClip.lineStyle()` en la página 515 y `MovieClip.moveTo()` en la página 521.

### Para crear una herramienta sencilla de trazo de líneas:

- 1 En un documento nuevo, cree un nuevo botón en el escenario y escriba `clear_btn` como nombre de instancia en el inspector de propiedades.
- 2 Seleccione el fotograma 1 en la línea de tiempo; a continuación, seleccione **Ventana > Paneles de desarrollo > Acciones** para abrir el panel Acciones en caso de que no aparezca ya en pantalla.
- 3 En este panel, introduzca los códigos siguientes:

```
_root.onMouseDown = function() {
    _root.lineStyle(5, 0xFF0000, 100);
    _root.moveTo(_root._xmouse, _root._ymouse);
    isDrawing = true;
};
_root.onMouseMove = function() {
    if (isDrawing == true) {
        _root.lineTo(_root._xmouse, _root._ymouse);
        updateAfterEvent();
    }
};
_root.onMouseUp = function() {
    isDrawing = false;
};
clear_btn.onRelease = function() {
    _root.clear();
};
```

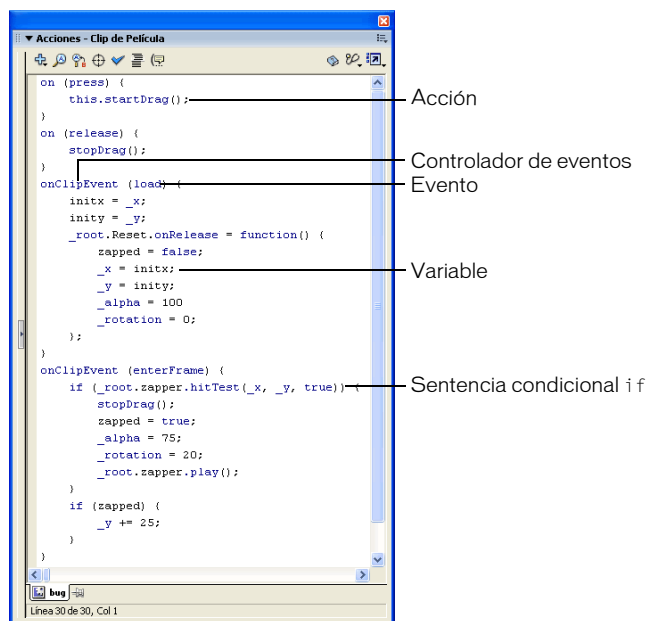
- 4 Seleccione **Control > Probar película** para probar la película. Haga clic y arrastre con el ratón para dibujar una línea en el escenario. Haga clic en el botón para borrar lo que ha dibujado.

## Análisis de un script de ejemplo

En el archivo SWF de muestra `zapper.swf` (que puede visualizarse en el apartado Utilización de Flash de la Ayuda), cuando un usuario arrastra la mariquita hasta la toma eléctrica, la mariquita cae y la toma empieza a vibrar. La línea de tiempo principal sólo tiene un fotograma y contiene tres objetos: la mariquita, la toma eléctrica y un botón de restablecimiento. Cada uno de estos objetos es una instancia de clip de película.



Hay un script en el archivo SWF; está asociado a la instancia `bug`, como se muestra a continuación en el panel Acciones:



*Panel Acciones con el script asociado a la instancia `bug`*

El nombre de instancia de la mariquita es `bug` y el de la toma es `zapper`. En el script se hace referencia a la mariquita como `this` porque el script está asociado a la mariquita y la palabra reservada `this` hace referencia al objeto que la contiene.

Existen dos controladores `onClipEvent()` con dos eventos diferentes: `load` y `enterFrame`. Las acciones de la sentencia `onClipEvent(load)` se ejecutan una sola vez, al cargarse el archivo SWF. Las acciones de la sentencia `onClipEvent(enterFrame)` se ejecutan cada vez que la cabeza lectora accede a un fotograma. Aunque se trate de archivos SWF de un solo fotograma, la cabeza lectora accederá al fotograma reiteradamente y el script se ejecutará de la misma forma. Dentro de cada controlador `onClipEvent()` se llevan a cabo las acciones siguientes:

**onClipEvent(load)** Se definen dos variables, `initx` e `inity`, para almacenar las posiciones *x* e *y* iniciales de la instancia de clip de película `bug`. Se define una función y se asigna al evento `onRelease` de la instancia `Restablecer`. Se llama a esta función cada vez que se presiona y suelta el botón del ratón en el botón `Restablecer`. La función coloca la mariquita de nuevo en su posición inicial en el escenario, restablece sus valores de rotación y alfa, y restablece la variable `zapped` en `false`.

**onClipEvent(enterFrame)** Una sentencia `if` condicional utiliza el método `hitTest()` para comprobar si la instancia del insecto está tocando la instancia de la toma eléctrica (`_root.zapper`). La comprobación puede generar dos resultados, `true` o `false`:

```
onClipEvent (load) {
    initx = _x;
    inity = _y;
    _root.Reset.onRelease = function() {
        zapped = false;
        _x = initx;
        _y = inity;
        _alpha = 100
        _rotation = 0;
    };
}
```

Si el método `hitTest()` devuelve el valor `true`, se llama al método `stopDrag()`, la variable `zapper` se establece en `true`, se cambia el valor de las propiedades alfa y de rotación y se indica a la instancia `zapped` que debe reproducirse.

Si el método `hitTest()` devuelve `false`, no se ejecuta ningún código especificado entre `{}` que aparezca inmediatamente después de la sentencia `if`.

Existen dos controladores `on()` asociados a la instancia `bug` con dos eventos distintos: `press` y `release`. Las acciones de la sentencia `on(press)` se ejecutan al presionar el botón del ratón sobre la instancia `bug`. Las acciones de la sentencia `on(release)` se ejecutan al liberar el botón del ratón sobre la instancia `bug`. Dentro de cada controlador `onClipEvent()` se llevan a cabo las siguientes acciones:

**on(press)** Una acción `startDrag()` permite arrastrar la mariquita. Puesto que el script está asociado a la instancia `bug`, la palabra clave `this` indica que se puede arrastrar la instancia `bug`:

```
on(press){
    this.startDrag();
}
```

**on(release)** Una acción `stopDrag()` detiene la acción de arrastre:

```
on(release) {
    stopDrag();
}
```

Para ver cómo se reproduce el archivo SWF, consulte la Ayuda de la guía de referencia de `ActionScript`.





## PARTE III

### Trabajo con objetos y clases

En esta sección se describe el modelo de objetos de tiempo de ejecución de Macromedia Flash y sus funciones, y se centra la atención en la manipulación de clips de película y de texto. Asimismo, se describe cómo crear clases e interfaces mediante ActionScript 2.0.

Capítulo 6: Utilización de clases incorporadas .....	115
Capítulo 7: Trabajo con clips de película .....	123
Capítulo 8: Trabajo con texto. ....	139
Capítulo 9: Creación de clases con ActionScript 2.0. ....	161



# CAPÍTULO 6

## Utilización de clases incorporadas

Además de los principales elementos de lenguaje y construcciones de ActionScript (como los bucles `for` y `while`) y los tipos de datos primitivos (números, cadenas y matrices) descritos anteriormente en [“Conceptos básicos de ActionScript” en la página 27](#), ActionScript proporciona una serie de clases incorporadas o *tipos de datos complejos*. Estas clases proporcionan varias funciones para la creación de scripts.

Algunas de estas clases están basadas en la especificación ECMAScript y se conocen como *clases principales de ActionScript*. Entre ellas se incluyen las clases `Array`, `Boolean`, `Date` y `Math`. Para más información, consulte [“Clases principales” en la página 117](#).

El resto de las clases incorporadas de ActionScript son específicas de Macromedia Flash y del modelo de objetos de Flash Player. Para comprender la distinción que existe entre las clases principales de ActionScript y las clases específicas de Flash, tenga en cuenta la distinción entre la clases JavaScript principales y las de la parte del cliente: del mismo modo que las clases JavaScript de la parte del cliente controlan el entorno del cliente (el contenido de la página Web y el navegador Web), las clases específicas de Flash controlan el aspecto y el comportamiento de una aplicación Flash en tiempo de ejecución.

Este capítulo presenta las clases incorporadas de ActionScript, describe tareas comunes que pueden realizarse con dichas clases y proporciona ejemplos de código. Para ver información general sobre estas clases, consulte [“Información general sobre las clases incorporadas” en la página 116](#). Para obtener información general sobre cómo utilizar las clases y los objetos en un entorno de programación orientada a objetos, consulte [“Clases e instancias” en la página 115](#).

### Clases e instancias

En un entorno de programación orientada a objetos, una *clase* define una categoría de objetos. Una clase describe las propiedades (datos) y el comportamiento (métodos) de un objeto, del mismo modo que un plano arquitectónico describe las características de un edificio. Para utilizar las propiedades y los métodos definidos por una clase, primero debe crear una *instancia* de dicha clase. La relación entre una instancia y su clase es similar a la relación entre una casa y sus planos arquitectónicos.

### Creación de un objeto nuevo

Para crear una instancia de una clase de ActionScript, utilice el operador `new` para invocar la función constructora de la clase. La función constructora tiene siempre el mismo nombre que la clase, y devuelve una instancia de la clase, que normalmente se asigna a una variable.

Por ejemplo, el código siguiente crea un nuevo objeto Sound.

```
var song:Sound= new Sound();
```

En algunos casos, no es necesario crear una instancia de una clase para utilizarla. Para más información, consulte [“Miembros de clase \(estáticos\)” en la página 116](#).

## Acceso a las propiedades del objeto

Utilice el operador de punto (.) para acceder al valor de una propiedad en un objeto. Ponga el nombre del objeto a la izquierda del punto y el nombre de la propiedad, en el lado derecho. Por ejemplo, en la siguiente declaración, myObject es el objeto y name es la propiedad:

```
myObject.name
```

El código siguiente crea un objeto TextField nuevo y luego define su propiedad autoSize con el valor true.

```
var my_text = new TextField();  
my_text.autoSize = true;
```

También puede utilizar el operador de acceso a una matriz ([ ]) para acceder a las propiedades de un objeto. Véase [“Operadores de punto y de acceso a una matriz” en la página 52](#).

## Llamada a métodos de objeto

Puede llamar al método de un objeto mediante el operador de punto (.) seguido por el método. Por ejemplo, el código siguiente crea un nuevo objeto Sound y llama a su método setVolume().

```
mySound = new Sound(this);  
mySound.setVolume(50);
```

## Miembros de clase (estáticos)

Algunas clases de ActionScript incorporadas tienen lo que se denomina *miembros de clase* (o *miembros estáticos*). Se accede o se invoca a los miembros de clase (propiedades y métodos) desde el propio nombre de la clase, no desde una instancia de la misma. Es decir, no se crea una instancia de la clase para utilizar estas propiedades y métodos.

Por ejemplo, todas las propiedades de la clase Math son estáticas. El código siguiente invoca el método max() de la clase Math para determinar entre dos números cuál es el mayor.

```
var largerNumber = Math.max(10, 20);
```

## Información general sobre las clases incorporadas

En esta sección se enumeran todas las clases de ActionScript y se incluyen una breve descripción de cada una y referencias a otras secciones relevantes de la documentación.

## Clases principales

Las clases principales de ActionScript son las clases que se toman directamente de ECMAScript. En la caja de herramientas Acciones, estas clases se encuentran en Clases incorporadas > subcarpeta Principal.

Clase	Descripción
Arguments	Matriz que contiene los valores que se han pasado como parámetros a una función. Véase la entrada <a href="#">Clase Arguments</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Array	La clase Array contiene métodos y propiedades para trabajar con objetos de matriz. Véase la entrada <a href="#">Clase Array</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Boolean	Este clase es un envoltorio para los valores booleanos ( <code>true</code> o <code>false</code> ). Véase la entrada <a href="#">Clase Boolean</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Button	Esta clase proporciona métodos y propiedades para trabajar con objetos de botón. Véase la entrada <a href="#">Clase Button</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Date	La clase Date proporciona acceso a los valores de fecha y hora relativos a la hora universal (hora de Greenwich) o al sistema operativo en el que se ejecuta Flash Player. Véase la entrada <a href="#">Clase Date</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Error	Esta clase contiene información sobre los errores que se detectan en los scripts. Normalmente se utiliza la sentencia <code>throw</code> para generar una condición de error, la cual puede gestionarse utilizando una sentencia <code>try...catch...finally</code> . Véase <a href="#">try...catch...finally</a> y las entradas <a href="#">Clase Error</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Function	Es la representación en clase de todas las funciones de ActionScript, tanto las originales de ActionScript como las que defina el usuario. Véase la entrada <a href="#">Clase Function</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Math	Esta clase sirve para acceder a constantes y funciones matemáticas y manipularlas. Todas las propiedades y métodos de la clase Math son estáticos y deben llamarse mediante la sintaxis <code>Math.method(parameter)</code> o <code>Math.constant</code> . Véase la entrada <a href="#">Clase Math</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Number	Es un envoltorio para el tipo de datos primitivo de número. Véase la entrada <a href="#">Clase Number</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Object	Esta clase se encuentra en la raíz de la jerarquía de clases de ActionScript; el resto de las clases heredan sus métodos y sus propiedades. Véase la entrada <a href="#">Clase Object</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
String	Es un envoltorio para el tipo de datos primitivo de cadena, que permite utilizar los métodos y las propiedades del objeto String para manipular tipos de valores de cadena primitivos. Véase la entrada <a href="#">Clase String</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.

## Clases específicas de Flash Player

En las tablas siguientes se enumeran las clases específicas de Flash Player y del modelo de tiempo de ejecución de Flash. Estas clases se dividen en cuatro categorías: clases de película (que proporcionan un control general de los archivos SWF y Flash Player), clases multimedia (para trabajar con sonido y vídeo), clases de cliente/servidor (para trabajar con XML y otras fuentes de datos externas) y clases de edición (que controlan el entorno de edición de Flash).

**Nota:** esta clasificación afecta a las ubicaciones de las clases en la caja de herramientas Acciones, pero no a la manera de utilizar las clases.

### Clases de película

Las clases de película controlan la mayor parte de los elementos visuales de un archivo SWF, incluidos los clips de película, los campos de texto y los botones. Las clases de película se encuentran en la caja de herramientas Acciones, en Clases incorporadas > subcarpeta Película.

Clase	Descripción
Accessibility	La clase Accessibility gestiona la comunicación entre los archivos SWF y las aplicaciones de lectura en pantalla. Los métodos de esta clase se utilizan con la propiedad <code>_accProps</code> global para controlar las propiedades accesibles de los clips de película, los botones y los campos de texto en tiempo de ejecución. Véase <code>_accProps</code> y las entradas <a href="#">Clase Accessibility</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
Button	Cada botón de un archivo SWF es una instancia de la clase Button. Esta clase proporciona métodos, propiedades y controladores de eventos para trabajar con botones. Véase la entrada <a href="#">Clase Button</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
Color	La clase Color permite obtener y establecer valores de colores RGB para los objetos de clip de película. Para más información, consulte la entrada <a href="#">Clase Color</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para ver un ejemplo de cómo utilizar la clase Color para cambiar el color de los clips de película, consulte <a href="#">“Configuración de valores de color” en la página 102</a> .
ContextMenu	La clase ContextMenu permite controlar el contenido del menú contextual de Flash Player. Puede asociar distintos objetos ContextMenu con objetos MovieClip, Button o TextField utilizando la propiedad <code>menu</code> disponible para dichas clases. También puede añadir elementos de menú personalizados a un objeto ContextMenu utilizando la clase ContextMenuitem. Véanse las entradas <a href="#">Clase ContextMenu</a> y <a href="#">Clase ContextMenuitem</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
ContextMenuitem	La clase ContextMenuitem permite crear nuevos elementos de menú que aparecen en el menú contextual de Flash Player. Los nuevos elementos de menú que se crean con esta clase se añaden al menú contextual de Flash Player utilizando la clase ContextMenu. Véanse las entradas <a href="#">Clase ContextMenu</a> y <a href="#">Clase ContextMenuitem</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .

Clase	Descripción
Key	Esta clase proporciona métodos y propiedades para obtener información sobre el teclado y las teclas que se presionan. Para más información, consulte la entrada <a href="#">Clase Key</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para ver un ejemplo de cómo capturar teclas presionadas para crear un archivo SWF interactivo, consulte <a href="#">“Captura de teclas presionadas” en la página 98</a> .
LocalConnection	La clase LocalConnection permite que se comuniquen dos archivos SWF que se ejecuten en el mismo equipo. Véase la entrada <a href="#">Clase LocalConnection</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
Mouse	La clase Mouse controla el ratón en un archivo SWF; por ejemplo, esta clase permite ocultar o mostrar el puntero del ratón. Para más información, consulte la entrada <a href="#">Clase Mouse</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para ver un ejemplo de cómo utilizar la clase Mouse, consulte <a href="#">“Creación de un puntero de ratón personalizado” en la página 96</a> .
MovieClip	Cada uno de los clips de película de una película Flash es una instancia de la clase MovieClip. Los métodos y propiedades de esta clase sirven para controlar los objetos de clip de película. Véanse el <a href="#">Capítulo 7, “Trabajo con clips de película”, en la página 123</a> y la entrada <a href="#">Clase MovieClip</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
MovieClipLoader	Esta clase permite seguir el progreso de la descarga de los archivos SWF y JPEG con un mecanismo de detector de eventos. Véanse el <a href="#">“Precarga de archivos SWF y JPEG” en la página 207</a> y la entrada <a href="#">Clase MovieClipLoader</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
PrintJob	La clase PrintJob permite imprimir contenido que se presenta dinámicamente y documentos de varias páginas. Véase la entrada <a href="#">Clase PrintJob</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> y <a href="#">“Utilización de la clase PrintJob de ActionScript” en el apartado Utilización de Flash de la Ayuda</a> .
Selection	La clase Selection permite obtener y establecer los campos de texto que aparecerán activos, los espacios de selección de campo de texto y los puntos de inserción de campo de texto. Véase la entrada <a href="#">Clase Selection</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
SharedObject	La clase SharedObject proporciona almacenamiento de datos local en el equipo cliente. Véase la entrada <a href="#">Clase SharedObject</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
Stage	Esta clase proporciona información sobre las dimensiones, la alineación y el modo de escala de un archivo SWF, y notifica los eventos de cambio de tamaño del escenario. Véase la entrada <a href="#">Clase Stage</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
System	La clase System proporciona información sobre Flash Player y el sistema en el que se ejecuta Flash Player (por ejemplo, la resolución de pantalla y el lenguaje actual del sistema). También permite mostrar u ocultar el panel de configuración de Flash Player y modificar la configuración de seguridad del archivo SWF. Véase la entrada <a href="#">Clase System</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .

Clase	Descripción
TextField	La clase TextField controla los campos de texto dinámico y de introducción de texto. Véanse el <a href="#">Capítulo 8, “Trabajo con texto”, en la página 139</a> y la entrada <a href="#">Clase TextField</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
TextField.StyleSheet	La clase TextField.StyleSheet (una clase incluida en la clase TextField) permite crear y aplicar estilos de texto CSS a texto en formato HTML o XML. Véanse el <a href="#">“Aplicación de formato al texto con hojas de estilos en cascada” en la página 143</a> y la entrada <a href="#">Clase TextField.StyleSheet</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
TextFormat	La clase TextFormat permite aplicar estilos de formato a caracteres o párrafos de un objeto TextField. Véanse el <a href="#">“Utilización de la clase TextFormat” en la página 141</a> y la entrada <a href="#">Clase TextFormat</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .

## Clases multimedia

Las clases multimedia controlan la reproducción de sonido y vídeo en un archivo SWF, así como el acceso al micrófono y cámara del usuario, si es que están instalados. Estas clases se encuentran en Clases incorporadas > subcarpeta Media de la caja de herramientas Acciones.

Clase	Descripción
Camera	La clase Camera proporciona acceso a la cámara del usuario, si es que está instalada. Cuando se utiliza con Flash Communication Server MX, el archivo SWF puede capturar, difundir y registrar imágenes y vídeo desde la cámara de un usuario. Véase la entrada <a href="#">Clase Camera</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
Microphone	La clase Microphone proporciona acceso al micrófono del usuario, si es que hay uno instalado. Cuando se utiliza con Flash Communication Server MX, el archivo SWF puede difundir y registrar audio desde el micrófono de un usuario. Véase la entrada <a href="#">Clase Microphone</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
NetConnection	La clase NetConnection sirve para establecer una conexión de transmisión local para reproducir un archivo Flash Video (FLV) desde una dirección HTTP o desde un sistema de archivos local. Para más información, consulte la entrada <a href="#">Clase NetConnection</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para más información sobre cómo reproducir archivos FLV en Internet, consulte <a href="#">“Reproducción dinámica de archivos FLV externos” en la página 205</a> .
NetStream	La clase NetStream sirve para controlar la reproducción de archivos FLV. Para más información, consulte la entrada <a href="#">Clase NetStream</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para más información sobre cómo reproducir archivos FLV en Internet, consulte <a href="#">“Reproducción dinámica de archivos FLV externos” en la página 205</a> .



Clase	Descripción
Sound	La clase Sound controla los sonidos de un archivo SWF. Para más información, consulte la entrada <a href="#">Clase Sound</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> . Para ver un ejemplo de cómo utilizar la clase Sound para crear controladores de volumen y balance, consulte <a href="#">“Creación de controles de sonido” en la página 103</a> .
Video	La clase Video sirve para mostrar objetos de vídeo en un archivo SWF. Véase la entrada <a href="#">Clase Video</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .

## Clases de cliente/servidor

En la tabla siguiente se enumeran las clases que sirven para enviar datos a fuentes externas y recibirlos de ellas, o para comunicarse con servidores de aplicaciones mediante FTP, HTTP o HTTPS.

**Nota:** en Flash Player 7, un archivo SWF sólo puede cargar datos del dominio del que procede. Para más información, consulte [“Funciones de seguridad de Flash Player” en la página 196](#) y [“Carga de datos de varios dominios” en la página 198](#).

Estas clases se encuentran en Clases incorporadas > subcarpeta Cliente/Servidor del panel Acciones.

Clase	Descripción
LoadVars	La clase LoadVars es una alternativa a la acción <code>loadVariables()</code> para transferir variables entre un archivo SWF y un servidor en pares de nombre/valor. Véanse <a href="#">“Utilización de la clase LoadVars” en la página 188</a> y la entrada <a href="#">Clase LoadVars</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
XML	La clase XML se amplía a la clase XMLNode y proporciona métodos, propiedades y controladores de eventos para trabajar con datos con formato XML, incluidos la carga y el análisis de XML externos, la creación de documentos XML y el desplazamiento por árboles de documentos XML. Véanse <a href="#">“Utilización de la clase XML” en la página 189</a> y la entrada <a href="#">Clase XML</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
XMLNode	Esta clase representa un único nodo de un árbol de documentos XML. Es la superclase de la clase XML. Véase la entrada <a href="#">Clase XMLNode</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .
XMLSocket	La clase XMLSocket permite crear una conexión de socket permanente con otro equipo para la transferencia de datos de baja latencia, como la que necesitan las aplicaciones de chat en tiempo real. Véanse <a href="#">“Utilización de la clase XMLSocket” en la página 192</a> y la entrada <a href="#">Clase XMLSocket</a> en el <a href="#">Capítulo 12, “Diccionario de ActionScript”, en la página 213</a> .

## Clases de edición

Las clases de edición sólo están disponibles en el entorno de edición de Flash. Estas clases se encuentran en Clases incorporadas > subcarpeta Edición de la caja de herramientas Acciones.

Clase	Descripción
CustomActions	La clase CustomActions permite gestionar las acciones personalizadas que se registran con la herramienta de edición. Véase la entrada <a href="#">Clase CustomActions</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.
Vista previa dinámica	La función Vista previa dinámica (que figura, aunque no sea una clase, bajo Clases incorporadas en la caja de herramientas Acciones) proporciona una sola función denominada <code>onUpdate</code> , utilizada por los desarrolladores de componentes. Véase <a href="#">onUpdate</a> en el <a href="#">Capítulo 12, "Diccionario de ActionScript"</a> , en la página 213.

# CAPÍTULO 7

## Trabajo con clips de película

Los clips de película son archivos SWF en miniatura con contenido propio que se ejecutan de manera independiente y al margen de la línea de tiempo en la que se encuentran. Por ejemplo, si la línea de tiempo principal sólo tiene un fotograma y un clip de película de dicho fotograma tiene diez fotogramas, cuando se reproduzca el archivo SWF principal se reproducirán todos los fotogramas del clip de película. Un clip de película puede, a su vez, contener otros clips de película o *clips anidados*. Los clips de película anidados de esta manera tienen una relación jerárquica, en la cual el *clip principal* contiene uno o varios *clips secundarios*.

Cada instancia de clip de película tiene un nombre, denominado *nombre de instancia*, que identifica al clip de manera unívoca como un objeto que puede controlarse con ActionScript. Concretamente, el nombre de instancia lo identifica como si fuera un objeto del tipo de clase MovieClip. Para controlar el aspecto y el comportamiento de los clips de película en tiempo de ejecución, utilice las propiedades y los métodos de la clase MovieClip.

Los clips de película vienen a ser objetos autónomos que pueden responder a eventos, enviar mensajes a otros objetos de clip de película, mantener su estado y controlar sus clips secundarios. De este modo, los clips de película proporcionan la base de una *arquitectura basada en componentes* en Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004. De hecho, los componentes disponibles en el panel Componentes (Ventana > Paneles de desarrollo > Componentes) son clips de película sofisticados que han sido diseñados y programados para tener un aspecto y un comportamiento concretos. Para más información sobre la creación de componentes, consulte *Utilización de componentes*.

### Control de clips de película con ActionScript

Para realizar tareas en clips de película, puede utilizar las funciones globales ActionScript o los métodos de la clase MovieClip. Algunos métodos MovieClip realizan las mismas tareas que las funciones del mismo nombre; otros métodos MovieClip, tales como `hitTest()` y `swapDepths()`, no tienen nombres de función correspondientes.

En el ejemplo siguiente se muestra la diferencia entre el uso de un método y de una función. Ambas sentencias duplican la instancia `my_mc`, asignan el nombre `newClip` al nuevo clip y lo colocan a una profundidad de 5.

```
my_mc.duplicateMovieClip("newClip", 5);  
duplicateMovieClip("my_mc", "newClip", 5);
```

Si una función y un método tienen comportamientos similares, puede utilizar cualquiera de los dos para controlar los clips de película. La elección depende de lo que prefiera y de lo familiarizado que esté con la creación de scripts en ActionScript. Tanto si utiliza una función como un método, la línea de tiempo de destino debe cargarse en Flash Player al llamar a la función o al método.

Para utilizar un método, debe invocarlo indicando la ruta de destino del nombre de instancia seguida de un punto y, después, del nombre del método y los parámetros, como en las sentencias siguientes:

```
myMovieClip.play();
parentClip.childClip.gotoAndPlay(3);
```

En la primera sentencia, `play()` mueve la cabeza lectora de la instancia `myMovieClip`. En la segunda sentencia, `gotoAndPlay` envía la cabeza lectora de `childClip` (que depende de la instancia `parentClip`) al fotograma 3 y continúa moviendo la cabeza lectora.

Las acciones globales que controlan una línea de tiempo tienen un parámetro de *destino* que permite especificar la ruta de destino a la instancia que se desea controlar. Por ejemplo, en el script siguiente `startDrag()` utiliza la instancia `customCursor` y hace que se pueda arrastrar:

```
on(press){
    startDrag("customCursor");
}
```

Las funciones siguientes utilizan clips de película: `loadMovie()`, `unloadMovie()`, `loadVariables()`, `setProperty()`, `startDrag()`, `duplicateMovieClip()` y `removeMovieClip()`. Para utilizar estas funciones, es necesario introducir una ruta de destino para el parámetro *target* de la función, a fin de indicar el destino de la misma.

Los métodos `MovieClip` siguientes pueden controlar clips de película o niveles cargados y no tienen funciones equivalentes: `MovieClip.attachMovie()`, `MovieClip.createEmptyMovieClip()`, `MovieClip.createTextField()`, `MovieClip.getBounds()`, `MovieClip.getBytesLoaded()`, `MovieClip.getBytesTotal()`, `MovieClip.getDepth()`, `MovieClip.getInstanceAtDepth()`, `MovieClip.getNextHighestDepth()`, `MovieClip.globalToLocal()`, `MovieClip.localToGlobal()`, `MovieClip.hitTest()`, `MovieClip.setMask()`, `MovieClip.swapDepths()`.

Para más información sobre estas funciones y métodos, consulte el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

## Llamada a varios métodos en un solo clip de película

Puede utilizar la sentencia `with` para referirse a un clip de película una vez y después ejecutar una serie de métodos en ese clip. La sentencia `with` funciona en todos los objetos de ActionScript (por ejemplo, `Array`, `Color` y `Sound`), no solamente en clips de película.

La sentencia `with` toma un objeto como parámetro. El objeto que especifique se agregará al final de la ruta de destino actual. Todas las acciones anidadas dentro de una sentencia `with` se llevan a cabo dentro de la nueva ruta de destino o ámbito. Por ejemplo, en el siguiente script, a la sentencia `with` se le pasa el objeto `donut.hole` para cambiar las propiedades de `hole`:

```
with (donut.hole){
    _alpha = 20;
    _xscale = 150;
    _yscale = 150;
}
```

El script se comporta como si a las sentencias que hay en la sentencia `with` se las llamara desde la línea de tiempo de la instancia `hole`. El código anterior equivale al siguiente:

```
donut.hole._alpha = 20;
donut.hole._xscale = 150;
donut.hole._yscale = 150;
```

El código anterior también equivale al siguiente:

```
with (donut){
    hole._alpha = 20;
    hole._xscale = 150;
    hole._yscale = 150;
}
```

## Carga y descarga de archivos SWF adicionales

Para reproducir películas SWF adicionales sin cerrar Flash Player o pasar de una película a otra sin cargar otra página HTML, puede utilizar la función global `loadMovie()` o el método `loadMovie()` de la clase `MovieClip`. Puede utilizar `loadMovie()` para enviar variables a un script CGI que genera un archivo SWF como resultado del CGI. Cuando carga un archivo SWF, puede especificar el nivel o clip de película en el que se cargará el archivo SWF. Si carga un archivo SWF en un destino, el archivo SWF cargado heredará las propiedades del clip de película de destino. Una vez que se ha cargado la película, puede cambiar dichas propiedades.

El método `unloadMovie()` elimina un archivo SWF cargado previamente con `loadMovie()`. La descarga de modo explícito de archivos SWF con `unloadMovie()` garantiza una transición gradual entre los archivos SWF y puede reducir la memoria requerida por Flash Player.

Utilice `loadMovie()` para seguir uno de estos procedimientos:

- Reproducir una secuencia de anuncios publicitarios en forma de archivos SWF colocando una función `loadMovie()` al final de cada archivo SWF para cargar el archivo SWF siguiente.
- Desarrollar una interfaz de bifurcación que permita al usuario elegir entre distintos archivos SWF.
- Crear una interfaz de navegación con controles de navegación en el nivel 0 que carguen otros niveles. Cargar niveles produce transiciones más suaves que cargar nuevas páginas HTML en un navegador.

Para más información sobre cómo cargar películas, consulte [“Carga de archivos SWF y JPEG externos” en la página 202](#).

## Especificación de una línea de tiempo raíz para archivos SWF cargados

La propiedad de `ActionScript _root` especifica o devuelve una referencia a la línea de tiempo raíz de un archivo SWF. Si un archivo SWF tiene varios niveles, la línea de tiempo raíz está en el nivel que contiene el script que se está ejecutando. Por ejemplo, si un script del nivel 1 consulta el valor de `_root`, se devuelve `_level1`. No obstante, la línea de tiempo especificada mediante `_root` puede cambiar en función de si un archivo SWF se ejecuta de forma independiente (en su propio nivel) o de si se ha cargado en una instancia de clip de película mediante una llamada `loadMovie()`.

Por ejemplo, pongamos por caso un archivo denominado `container.swf` que tenga una instancia de clip de película denominada `target_mc` en su línea de tiempo principal. El archivo `container.swf` declara una variable denominada `userName` en su línea de tiempo principal; a continuación, el mismo script carga otro archivo denominado `contents.swf` en el clip de película `target_mc`.

```
// En container.swf:  
_root.userName = "Tim";  
target_mc.loadMovie("contents.swf");
```

El archivo SWF cargado, `contents.swf`, también declara una variable denominada `userName` en su línea de tiempo raíz.

```
// En content.swf:  
_root.userName = "Mary";
```

Cuando `contents.swf` se carga en el clip de película de `container.swf`, el valor de `userName`, que está asociado a la línea de tiempo raíz del archivo SWF que aloja (`container.swf`), se establece en "Mary". Esto puede ocasionar que el código de `container.swf` (así como `contents.swf`) no funcione correctamente.

Para hacer que `_root` indique siempre el valor de línea de tiempo del archivo SWF cargado, en lugar de la línea de tiempo raíz, utilice la propiedad `_lockroot`. Esta propiedad puede establecerla el archivo SWF que ejecuta la carga o el archivo SWF que se está cargando. Cuando `_lockroot` se establece en `true` en una instancia de clip de película, dicho clip de película actuará como `_root` para cualquier SWF que esté cargado en él. Cuando `_lockroot` se establece en `true` en un archivo SWF, dicho archivo SWF actuará como su propia raíz al margen de qué otro archivo SWF la cargue. Puede establecerse `_lockroot` en `true` en cualquier clip de película y en tantos como se desee. De forma predeterminada, esta propiedad es `false`.

Por ejemplo, el autor de `container.swf` podría asociar el código siguiente al clip de película `target_mc`:

```
// Asociado al clip de película target_mc:  
onClipEvent (load) {  
    this._lockroot = true;  
}
```

Esto garantizaría que las referencias a `_root` de `contents.swf`, o cualquier archivo SWF cargado en `target_mc`, se aplicarían a su propia línea de tiempo, no a la línea de tiempo raíz de `container.swf`.

Del mismo modo, el autor de `contents.swf` podría añadir el código siguiente a su línea de tiempo principal.

```
// En contents.swf:  
this._lockroot = true;
```

Esto garantizaría que, al margen de donde estuviera cargada `contents.swf`, las referencias que hiciera a `_root` se aplicarían a su propia línea de tiempo principal, no a la línea de tiempo del archivo SWF que aloja.

Para más información, consulte [MovieClip.\\_lockroot en la página 519](#).

## Cargar archivos JPEG en clips de película

Puede utilizar la función `loadMovie()`, o el método `MovieClip` del mismo nombre, para cargar archivos de imagen JPEG en una instancia de clip de película. También puede utilizar la función `loadMovieNum()` para cargar un archivo JPEG en un nivel.

Al cargar una imagen en un clip de película, la esquina superior izquierda de la imagen se coloca en el punto de registro del clip de película. Dado que este punto suele ser el centro del clip de película, la imagen cargada no aparecerá en el centro. Además, cuando se carga una imagen en una línea de tiempo raíz, la esquina superior izquierda de la imagen se sitúa en la esquina superior izquierda del escenario. La imagen cargada hereda la rotación y la escala del clip de película, pero el contenido original del clip de película se elimina.

Para más información, consulte [“Carga de archivos SWF y JPEG externos” en la página 202](#), [loadMovie\(\) en la página 426](#), [MovieClip.loadMovie\(\) en la página 517](#) y [loadMovieNum\(\) en la página 427](#).

## Modificación de la posición y el aspecto de un clip de película

Para cambiar las propiedades de un clip de película a medida que se reproduce, escriba una sentencia que asigne un valor a una propiedad o utilice la función `setProperty()`. Por ejemplo, en el código siguiente se establece la rotación de la instancia `mc` en 45:

```
mc._rotation = 45;
```

Esto es equivalente al código siguiente, en el que se utiliza la función `setProperty()`:

```
setProperty("mc", _rotation, 45);
```

Algunas propiedades, llamadas *propiedades de sólo lectura*, tienen valores que se pueden leer pero no establecer. Estas propiedades se especifican como de sólo lectura en las entradas de diccionario de `ActionScript` correspondientes. A continuación, se muestran propiedades de sólo lectura: `_currentframe`, `_droptarget`, `_framesloaded`, `_parent`, `_target`, `_totalframes`, `_url`, `_xmouse` e `_ymouse`.

Puede escribir sentencias para establecer cualquier propiedad que no sea de sólo lectura. La sentencia siguiente establece la propiedad `_alpha` de la instancia de clip de película `wheel`, que es secundaria de la instancia `car`:

```
car.wheel._alpha = 50;
```

Además, puede escribir sentencias que obtienen el valor de una propiedad de clip de película. Por ejemplo, la sentencia siguiente obtiene el valor de la propiedad `_xmouse` de la línea de tiempo del nivel actual y establece la propiedad `_x` de la instancia `customCursor` en ese valor:

```
onClipEvent(enterFrame) {
    customCursor._x = _root._xmouse;
}
```

Esto es equivalente al código siguiente, en el que se utiliza la función `getProperty()`:

```
onClipEvent(enterFrame) {
    customCursor._x = getProperty(_root, _xmouse);
}
```

Las propiedades `_x`, `_y`, `_rotation`, `_xscale`, `_yscale`, `_height`, `_width`, `_alpha` y `_visible` se ven afectadas por las transformaciones del elemento principal del clip de película y modifican el clip de película y cualquiera de los elementos secundarios del clip. Las propiedades `_focusrect`, `_highquality`, `_quality` y `_soundbuftime` son globales, solamente pertenecen al nivel 0 de la línea de tiempo principal. Todas las demás propiedades pertenecen a cada clip de película o nivel cargado.

Para obtener una lista de propiedades de clips de película, consulte [“Resumen de propiedades para la clase MovieClip” en la página 490](#).

## Clips de película que se pueden arrastrar

Puede utilizar la función `startDrag()` global o el método `MovieClip.startDrag()` para que sea posible arrastrar un clip de película. Por ejemplo, puede crear un clip de película que se pueda arrastrar para juegos, funciones de arrastrar y soltar, interfaces personalizables, barras de desplazamiento y controles deslizantes.

Un clip de película se puede arrastrar hasta que esta acción se detenga explícitamente con `stopDrag()`, o hasta que se utilice otro clip de película con `startDrag`. Sólo un clip de película puede arrastrarse al mismo tiempo.

Para crear comportamientos de arrastrar y soltar más complejos, puede probar la propiedad `_droptarget` del clip de película que se está arrastrando. Por ejemplo, puede examinar la propiedad `_droptarget` para ver si el clip de película se arrastró a un clip de película específico (como un clip de película de tipo “papelera”) y, a continuación, desencadenar otra acción. Para información detallada, consulte [startDrag\(\) en la página 647](#) o [MovieClip.startDrag\(\) en la página 537](#).

## Creación de clips de película en tiempo de ejecución

No sólo es posible crear instancias de clips de película en el entorno de creación de Flash, sino que además se pueden crear en tiempo de ejecución. ActionScript proporciona tres maneras de crear nuevos clips de película en tiempo de ejecución:

- Crear una nueva instancia de clip de película vacía
- Duplicar una instancia de clip de película existente
- Asociar al escenario la instancia de un símbolo de biblioteca de clips de película

Cada instancia de clip de película que cree en tiempo de ejecución deberá tener un nombre de instancia y un valor de profundidad (apilamiento u orden *z*). La profundidad especificada determina la manera en que el nuevo clip se solapa con otros clips en la misma línea de tiempo. (Véase [“Gestión de las profundidades de los clips de película” en la página 132](#).)



## Creación de un clip de película vacío

Para crear un clip de película vacío en el escenario, utilice el método `createEmptyMovieClip()` de la clase `MovieClip`. Este método crea un clip de película como elemento secundario del clip que llama al método. El punto de registro de un clip de película vacío recién creado se encuentra en la esquina superior izquierda.

Por ejemplo, en el código siguiente se crea un nuevo clip de película secundario, denominado `new_mc`, a una profundidad de 10 pulgadas en el clip de película denominado `parent_mc`.

```
parent_mc.createEmptyMovieClip("new_mc", 10);
```

Con el código siguiente se crea un nuevo clip de película, denominado `canvas_mc`, en la línea de tiempo raíz del archivo SWF en el que se ejecuta el script y, a continuación, se invoca la acción `loadMovie()` para cargar en ella un archivo JPEG externo.

```
_root.createEmptyMovieClip("canvas_mc", 10);  
canvas_mc.loadMovie("flowers.jpg");
```

Para más información, consulte [MovieClip.createEmptyMovieClip\(\)](#) en la página 499.

## Duplicación o eliminación de un clip de película

Para duplicar o eliminar instancias de clip de película, utilice las funciones globales `duplicateMovieClip()` o `removeMovieClip()`, o bien los métodos de la clase `MovieClip` que llevan el mismo nombre. Con el método `duplicateMovieClip()` se crea una nueva instancia de una instancia de clip de película existente, se le asigna un nuevo nombre de instancia y se le da una profundidad o un orden *z*. Un clip de película duplicado siempre comienza en el fotograma 1, aunque el clip de película original se encontrase en otro fotograma cuando se duplicó, y siempre se encuentra delante de todos los clips de película definidos anteriormente situados en la línea de tiempo.

Para borrar un clip de película creado con `duplicateMovieClip()`, utilice `removeMovieClip()`. Los clips de película duplicados también se eliminan si se borra el clip de película principal.

Para más información, consulte [duplicateMovieClip\(\)](#) en la página 379 y [removeMovieClip\(\)](#) en la página 608.

## Asociación de un símbolo de clip de película al escenario

Una nueva manera de crear instancias de clip de película en tiempo de ejecución consiste en utilizar `attachMovie()`. El método `attachMovie()` asocia al escenario una instancia de un símbolo de clip de película de la biblioteca del archivo SWF. El nuevo clip pasa a ser un clip secundario del clip que lo ha asociado.

Para utilizar `ActionScript` a fin de asociar un símbolo de clip de película de la biblioteca, es necesario exportar el símbolo para `ActionScript` y asignarle un identificador de vínculo exclusivo. Para ello, deberá utilizar el cuadro de diálogo Propiedades de vinculación.

De forma predeterminada, todos los clips de película que se exportan para usarlos con `ActionScript` se cargan antes del primer fotograma del archivo SWF que los contiene. Esto puede producir cierta demora en la reproducción del primer fotograma. Cuando asigne un identificador de vínculo a un elemento, también puede especificar si este contenido debe agregarse antes del primer fotograma. Si no se añade en el primer fotograma, debe incluir una instancia de éste en algún otro fotograma del archivo SWF; de lo contrario, el elemento no se exportará al archivo SWF.

### Para asignar un identificador de vínculo a un clip de película:

- 1 Seleccione Ventana > Biblioteca, para abrir el panel Biblioteca.
- 2 Seleccione un clip de película del panel Biblioteca.
- 3 En el panel Biblioteca, elija Vinculación en el menú de opciones del panel Biblioteca.  
Aparecerá el cuadro de diálogo Propiedades de vinculación.
- 4 En Vinculación, seleccione Exportar para ActionScript.
- 5 En Identificador, introduzca un ID para el clip de película.  
De forma predeterminada, el identificador y el símbolo tienen el mismo nombre.
- 6 También puede asignar una clase ActionScript 2.0 al símbolo del clip de película. (Véase [“Asignación de una clase a un símbolo de clip de película” en la página 135.](#))
- 7 Si no desea que el clip de película se cargue antes que el primer fotograma, deseleccione la opción Exportar en primer fotograma.  
Si deselecciona esta opción, coloque una instancia del clip de película en el fotograma de la línea de tiempo donde desee que esté disponible. Por ejemplo, si el script que está escribiendo no hace referencia al clip de película hasta el fotograma 10, coloque una instancia del símbolo en dicho fotograma de la línea de tiempo, o bien antes del mismo.
- 8 Haga clic en Aceptar.

Cuando haya asignado un identificador de vínculo a un clip de película, podrá asociar una instancia del símbolo al escenario en tiempo de ejecución utilizando `attachMovie()`.

### Para adjuntar un clip de película a otro clip de película:

- 1 Asigne un identificador de vínculo a un símbolo de biblioteca de clips de película, tal como se ha descrito anteriormente.
- 2 Con el panel Acciones abierto (Ventana > Paneles de desarrollo > Acciones), seleccione un fotograma en la línea de tiempo.
- 3 En el panel Script del panel Acciones, escriba el nombre del clip de película o el nivel al que desea asociar el nuevo clip de película. Por ejemplo, para asociar el clip de película a la línea de tiempo raíz, escriba `_root`.
- 4 En la caja de herramientas Acciones (situada a la izquierda del panel Acciones), haga clic en la categoría Built-in Classes, la categoría Movie y la categoría MovieClip y, a continuación, haga doble clic en `attachMovie()`.
- 5 Utilizando las sugerencias para el código que aparecen como guía, introduzca valores para los parámetros siguientes:
  - En `idName`, especifique el identificador introducido en el cuadro de diálogo Propiedades de vinculación.
  - En `newName`, introduzca un nombre de instancia para el clip adjuntado para poder utilizarlo.
  - En `depth`, introduzca el nivel en el que el clip de película duplicado se asociará al clip de película. Cada clip de película asociado tiene su propio orden de apilamiento, siendo el nivel 0 el nivel del clip de película que lo originó. Los clips de película adjuntados siempre están encima del clip de película original. A continuación, se muestra un ejemplo:  
`myMovieClip.attachMovie("calif", "california", 10);`

Para más información, consulte [MovieClip.attachMovie\(\)](#) en la página 493.

## Adición de parámetros a clips de película creados de forma dinámica

Al crear o duplicar un clip de película de forma dinámica utilizando `MovieClip.attachMovie()` y `MovieClip.duplicateMovieClip()`, puede rellenar el clip de película con parámetros de otro objeto. El parámetro `initObject` de `attachMovie()` y `duplicateMovieClip()` permite que los clips de película creados de forma dinámica reciban parámetros de clip. El parámetro `initObject` es opcional.

Para más información, consulte [MovieClip.attachMovie\(\)](#) en la página 493 y [MovieClip.duplicateMovieClip\(\)](#) en la página 503.

**Para rellenar un clip de película creado de forma dinámica con parámetros de un objeto especificado, siga uno de estos procedimientos:**

- Utilice la sintaxis siguiente con `attachMovie()`:  
`myMovieClip.attachMovie(idName, newName, depth [, initObject])`
- Utilice la sintaxis siguiente con `duplicateMovieClip()`:  
`myMovieClip.duplicateMovieClip(idName, newName, depth [, initObject])`

Con el parámetro `initObject` se especifica el nombre del objeto cuyos parámetros se van a utilizar para rellenar el clip de película creado de forma dinámica.

**Para rellenar un clip de película con parámetros utilizando `attachMovie()`:**

- 1 En un nuevo documento de Flash, cree un nuevo símbolo de clip de película seleccionando Insertar > Nuevo símbolo. Escriba `dynamic` en el cuadro de texto Nombre y seleccione el comportamiento del clip de película.
- 2 Dentro del símbolo, cree un campo de texto dinámico en el escenario, con el nombre de instancia `name_txt`.
- 3 Seleccione el primer fotograma de la línea de tiempo del clip de película y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 4 Cree una nueva variable llamada `name` y asigne su valor a la propiedad `text` de `name_txt`, tal como se muestra a continuación:  

```
var name:String;  
name_txt.text = name;
```
- 5 Seleccione Edición > Editar documento para volver a la línea de tiempo principal.
- 6 Seleccione el símbolo de clip de película de la biblioteca y, a continuación, seleccione Propiedades de vinculación en el menú de opciones del panel Biblioteca.  
Aparecerá el cuadro de diálogo Propiedades de vinculación.
- 7 Seleccione la opción Exportar para ActionScript y haga clic en Aceptar.
- 8 Seleccione el primer fotograma de la línea de tiempo principal y añada el código siguiente al panel Script del panel Acciones:  

```
_root.attachMovie("dynamic", "newClipName", 10, {name:"Erick"});
```
- 9 Pruebe la película (Control > Probar película). El nombre especificado en la llamada a `attachMovie()` aparece dentro del campo de texto del nuevo clip de película.

## Gestión de las profundidades de los clips de película

Cada clip de película tiene su propio espacio de orden  $z$  que determina la forma en la que los objetos se solapan en el archivo SWF o clip de película principal. Cada clip de película tiene asociado un valor de profundidad que determina si dicho clip se presentará delante o detrás de otros clips de película dentro de la misma línea de tiempo del clip de película. Cuando se crea un nuevo clip de película en tiempo de ejecución utilizando `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()` o `MovieClip.createEmptyMovieClip()`, debe especificarse siempre la profundidad del nuevo clip como parámetro de método. Por ejemplo, el código siguiente asocia un nuevo clip de película a la línea de tiempo de un clip de película denominado `container_mc` con el valor de profundidad 10.

```
container_mc.attachMovie("symbolID", "clip_1", 10);
```

Con esto se crea un nuevo clip de película con la profundidad 10 en el espacio de orden  $z$  de `container_mc`.

Por ejemplo, el código siguiente asocia dos nuevos clips de película a `container_mc`. El primer clip, llamado `clip_1`, se representará detrás de `clip_2`, puesto que se le asignó un valor inferior de profundidad.

```
container_mc.attachMovie("symbolID", "clip_1", 10);
container_mc.attachMovie("symbolID", "clip_2", 15);
```

Los valores de profundidad para los clips de película pueden oscilar entre -16384 y 1048575.

La clase `MovieClip` proporciona varios métodos para gestionar las profundidades de los clips de película: Véase `MovieClip.getNextHighestDepth()` en la página 509, `MovieClip.getInstanceAtDepth()` en la página 508, `MovieClip.getDepth()` en la página 508 y `MovieClip.swapDepths()` en la página 539.

## Determinación del siguiente valor superior de profundidad disponible

Para determinar el siguiente valor superior de profundidad disponible de un clip de película, consulte `MovieClip.getNextHighestDepth()`. El valor entero devuelto por este método indica la próxima profundidad disponible que se representará delante del resto de los objetos del clip de película.

Con el código siguiente se crea un nuevo clip de película, con el valor de profundidad 10, en la línea de tiempo del clip de película denominado `menus_mc`. A continuación, determina la siguiente profundidad superior disponible de ese mismo clip de película y crea un nuevo clip de película en esa profundidad.

```
menus_mc.attachMovie("menuClip","file_menu", 10);
var nextDepth = menus_mc.getNextHighestDepth();
menus_mc.attachMovie("menuClip", "edit_menu", nextDepth);
```

En este caso, la variable denominada `nextDepth` contiene el valor 11, puesto que es el siguiente valor de profundidad superior disponible para el clip de película `menus_mc`.

Para obtener el valor actual más alto de profundidad ocupado, reste 1 al valor devuelto por `getNextHighestDepth()`, tal como se muestra en la sección siguiente (véase “[Determinación de la instancia a una profundidad determinada](#)” en la página 133).

## Determinación de la instancia a una profundidad determinada

Para determinar la instancia a una profundidad determinada, utilice `MovieClip.getInstanceAtDepth()`. Este método devuelve una referencia a la instancia a la profundidad especificada.

El código siguiente combina `getNextHighestDepth()` y `getInstanceAtDepth()` para determinar el clip de película en el valor de profundidad (actual) más alto ocupado de la línea de tiempo raíz.

```
var highestOccupiedDepth = _root.getNextHighestDepth() - 1;  
var instanceAtHighestDepth = _root.getInstanceAtDepth(highestOccupiedDepth);
```

Para más información, consulte [MovieClip.getInstanceAtDepth\(\)](#) en la página 508.

## Determinación de la profundidad de una instancia

Para determinar la profundidad de una instancia de clip de película, utilice `MovieClip.getDepth()`.

El código siguiente se repite en todos los clips de película de la línea de tiempo principal de un archivo SWF y muestra el nombre de instancia y el valor de profundidad correspondiente de cada clip en el panel Salida.

```
for(each in _root) {  
    var obj = _root[each];  
    if(obj instanceof MovieClip) {  
        var objDepth = obj.getDepth();  
        trace(obj._name + ":" + objDepth)  
    }  
}
```

Para más información, consulte [MovieClip.getDepth\(\)](#) en la página 508.

## Intercambio de profundidades de clip de película

Para intercambiar las profundidades de dos clips de película en la misma línea de tiempo, utilice `MovieClip.swapDepths()`. Para más información, consulte [MovieClip.swapDepths\(\)](#) en la página 539.

## Dibujo de formas con ActionScript

Puede utilizar métodos de la clase `MovieClip` para trazar líneas y rellenos en el escenario. Esto permite crear herramientas de dibujo para los usuarios y dibujar formas en la película como respuesta a los eventos. Los métodos de dibujo son `beginFill()`, `beginGradientFill()`, `clear()`, `curveTo()`, `endFill()`, `lineTo()`, `lineStyle()` y `moveTo()`.

Puede utilizar estos métodos de dibujo con cualquier clip de película. Sin embargo, si utiliza los métodos de dibujo con un clip de película que se ha creado en modo de edición, los métodos se ejecutarán antes de dibujar el clip. Es decir, el contenido creado en modo de edición se dibuja encima del contenido dibujado con los métodos de dibujo.

Puede utilizar clips de película con métodos de dibujo como máscaras; sin embargo, al igual que en todas las máscaras de clip de película, los trazos se pasan por alto.

### Para dibujar una forma:

- 1 Utilice `createEmptyMovieClip()` para crear un clip de película vacío en el escenario.  
El nuevo clip de película es un elemento secundario de un clip de película existente o de la línea de tiempo principal, tal como se muestra a continuación:

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

- 2 Utilice el clip de película vacío para llamar a los métodos de dibujo.

En el ejemplo siguiente se dibuja un triángulo con líneas de color magenta de 5 puntos y sin relleno:

```
with (_root.triangle) {  
    lineStyle (5, 0xff00ff, 100);  
    moveTo (200, 200);  
    lineTo (300, 300);  
    lineTo (100, 300);  
    lineTo (200, 200);  
}
```

Para más información sobre estos métodos, consulte las entradas correspondientes en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Utilización de clips de película como máscaras

Puede utilizar un clip de película como una máscara para crear un agujero a través del cual se ve el contenido de otro clip de película. El clip de película de máscara reproduce todos los fotogramas de su línea de tiempo, igual que un clip de película normal. Puede hacer que el clip de película de máscara se pueda arrastrar, animarlo a lo largo de una guía de movimiento, utilizar formas separadas en una sola máscara o cambiar el tamaño de una máscara de forma dinámica. También puede utilizar ActionScript para activar y desactivar una máscara.

No puede utilizar una máscara para enmascarar otra máscara ni establecer la propiedad `_alpha` de un clip de película de máscara. En un clip de película que se usa como máscara, sólo se pueden utilizar los rellenos; los trazos se pasan por alto.

### Para crear una máscara:

- 1 En el escenario, elija el clip de película que desea enmascarar.
- 2 En el inspector de propiedades, introduzca un nombre de instancia para el clip de película, como por ejemplo `image`.
- 3 Cree el clip de película que se va a enmascarar. Asígnele un nombre de instancia en el inspector de propiedades, como por ejemplo `mask`.  
El clip de película con máscara se revelará bajo todas las zonas opacas (no transparentes) del clip de película que actúa como máscara.
- 4 Seleccione el fotograma 1 de la línea de tiempo.
- 5 Abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones), si todavía no está abierto.
- 6 En este panel, introduzca los códigos siguientes:

```
image.setMask(mask);
```

Para más información, consulte [MovieClip.setMask\(\)](#) en la página 536.

## Enmascaramiento de fuentes de dispositivo

Puede utilizar un clip de película para enmascarar un texto configurado en una fuente de dispositivo. Para que la máscara de un clip de película de una fuente de dispositivo funcione correctamente, el usuario deberá tener Flash Player 6 versión 40 o posterior.

Cuando se utiliza un clip de película para enmascarar texto configurado 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 clip de película que no sea rectangular para el texto de fuente de dispositivo en el entorno de edición de Macromedia Flash, la máscara que aparecerá en la película SWF tendrá la forma del recuadro de delimitación rectangular de la máscara, y no la de la máscara en sí.

Para enmascarar fuentes de dispositivo, sólo puede utilizarse un clip de película como máscara. No se puede enmascarar fuentes de dispositivo utilizando una capa de máscara en el escenario.

## Gestión de eventos de clip de película

Los clips de película pueden responder a eventos de usuario, tales como presionar los botones del ratón o teclas, así como a eventos a nivel de sistema, tales como la carga inicial de un clip de película en el escenario. ActionScript proporciona dos maneras de gestionar eventos de clip de película: a través de los métodos de controlador de eventos y a través de los controladores de eventos `onClipEvent()` y `on()`. Para más información, consulte [Capítulo 4, “Gestión de eventos”](#), en la [página 85](#).

## Asignación de una clase a un símbolo de clip de película

Mediante ActionScript 2.0 puede crear su propia clase, que amplía el comportamiento de la clase `MovieClip` incorporada y, después, puede asignar dicha clase a un símbolo de biblioteca de clip de película con el cuadro de diálogo Propiedades de vinculación. Siempre que se crea una instancia del clip de película al que se asignó la clase, ésta adopta las propiedades y los comportamientos definidos por la clase asignada al mismo. Para más información sobre ActionScript 2.0, consulte el [Capítulo 9, “Creación de clases con ActionScript 2.0”](#), en la [página 161](#).

En una subclase de la clase `MovieClip`, puede proporcionar definiciones de método para los métodos `MovieClip` incorporados y los controladores de eventos, como `onEnterFrame` y `onRelease`. Con el procedimiento siguiente se crea una clase denominada `MoveRight` que se amplía a la clase `MovieClip`; `MoveRight` define un controlador `onPress` que mueve el clip 20 píxeles hacia la derecha cuando el usuario hace clic en el clip de película. El segundo procedimiento consiste en crear un símbolo de clip de película en un nuevo documento de Flash (FLA) y en asignar la clase `MoveRight` a dicho símbolo.

**Para crear una subclase de clip de película:**

- 1 Cree un nuevo directorio llamado `BallTest`.
- 2 Cree un nuevo archivo ActionScript utilizando uno de los procedimientos siguientes:
  - En Flash MX Professional 2004, seleccione Archivo > Nuevo y, a continuación, seleccione archivo ActionScript de la lista de tipos de documento.
  - En Flash MX 2004, cree un archivo de texto en el editor de texto que prefiera.

### 3 Escriba el código siguiente en el script:

```
// La clase MoveRight mueve el clip hacia la derecha 5 píxeles cada fotograma
class MoveRight extends MovieClip {
    function onPress() {
        this._x += 20;
    }
}
```

### 4 Guarde el documento como MoveRight.as en el directorio BallTest.

#### Para asignar la clase a un símbolo de clip de película:

- 1 En Flash, seleccione Archivo > Nuevo y, en la lista de tipos de archivo, seleccione Documento de Flash y pulse Aceptar.
- 2 Dibuje un círculo en el escenario con la herramienta Óvalo.
- 3 Seleccione el círculo y, a continuación, seleccione Modificar > Convertir en símbolo. En el cuadro de diálogo Convertir en símbolo, seleccione Clip de película como comportamiento del símbolo y escriba Ball en el cuadro de texto Nombre.
- 4 Abra el panel Biblioteca (Ventana > Biblioteca) y seleccione el símbolo Ball.
- 5 Seleccione Vinculación, en el menú de opciones del panel Biblioteca, para abrir el cuadro de diálogo Propiedades de vinculación.
- 6 En el cuadro de diálogo Propiedades de vinculación, seleccione la opción Exportar para ActionScript y escriba MoveRight en el cuadro de texto Clase de AS 2.0. Haga clic en Aceptar.
- 7 Guarde el archivo como Ball fla en el directorio BallTest (el directorio que contiene el archivo MoveRight.as).
- 8 Pruebe la película (Control > Probar película).

Cada vez que haga clic en el clip de película, éste se moverá 20 píxeles a la derecha.

## Inicialización de las propiedades de clase

En el ejemplo anterior, se ha añadido la instancia del símbolo Ball al escenario de forma manual, es decir, durante el proceso de edición. Como se ha dicho con anterioridad (véase [“Adición de parámetros a clips de película creados de forma dinámica” en la página 131](#)), puede asignar parámetros a los clips que cree en tiempo de ejecución utilizando el parámetro *initObject* de *attachMovie()* y *duplicateMovie()*. Puede utilizar esta función para inicializar las propiedades de la clase que vaya a asignar a un clip de película.

Por ejemplo, la clase siguiente, denominada MoveRightDistance, es una variación de la clase MoveRight, explicada anteriormente (véase [“Asignación de una clase a un símbolo de clip de película” en la página 135](#)). La diferencia es una nueva propiedad denominada *distance*, cuyo valor determina el número de píxeles que se desplaza un clip de película cada vez que se hace clic en él.

```
// La clase MoveRightDistance mueve el clip hacia la derecha 5 píxeles cada
// fotograma
class MoveRightDistance extends MovieClip {
    // la propiedad distance determina cuántos
    // píxeles debe desplazarse el clip cada vez que se presiona el botón del
    // ratón
    var distance:Number;
    function onPress() {
        this._x += distance;
    }
}
```



Suponiendo que esta clase se haya asignado a un símbolo cuyo identificador de vínculo sea `Ball`, con el código siguiente se crean dos nuevas instancias del símbolo en la línea de tiempo raíz del archivo SWF. La primera instancia, denominada `ball_50`, se mueve 50 píxeles cada vez que se hace clic en ella; la segunda, denominada `ball_125`, se mueve 125 píxeles cada vez que se hace clic en ella.

```
_root.attachMovie("Ball", "ball_50", 10, {distance:50});  
_root.attachMovie("Ball", "ball_125", 20, {distance:125});
```



# CAPÍTULO 8

## Trabajo con texto

Un campo de texto dinámico o de entrada es un objeto `TextField` (una instancia de la clase `TextField`). Al crear un campo de texto, puede asignarle un nombre de instancia en el inspector de propiedades. Puede utilizar el nombre de instancia en sentencias de `ActionScript` para establecer, modificar y dar formato al campo de texto y a su contenido mediante las clases `TextField` y `TextFormat`.

Los métodos de la clase `TextField` permiten establecer, seleccionar y manipular texto de un campo de texto dinámico o de entrada que se cree durante la edición o la ejecución. Para más información, consulte [“Utilización de la clase `TextField`” en la página 140](#). Para más información sobre depuración de campos de texto en tiempo de ejecución, consulte [“Visualización de las propiedades de un campo de texto para la depuración” en la página 81](#).

`ActionScript` también proporciona diversas maneras de dar formato a los textos durante la ejecución. La clase `TextFormat` permite definir el formato de carácter y de párrafo para los objetos `TextField` (véase [“Utilización de la clase `TextFormat`” en la página 141](#)). Flash Player también admite un subconjunto de etiquetas `HTML` que puede utilizar para dar formato al texto (véase [“Utilización de texto en formato `HTML`” en la página 151](#)). Flash Player 7 y las versiones posteriores admiten la etiqueta `HTML` `<img>`, que permite incorporar no sólo imágenes externas, sino también archivos `SWF` externos, así como clips de película que residen en la biblioteca (véase [“Etiqueta de imagen \(`<img>`\)” en la página 153](#)).

En Flash Player 7 y versiones posteriores, puede aplicar estilos `CSS` (Cascading Style Sheets, hojas de estilos en cascada) a los campos de texto mediante la clase `TextField.StyleSheet`. Puede utilizar `CSS` para aplicar un estilo a las etiquetas `HTML` incorporadas, definir nuevas etiquetas de formato o aplicar estilos. Para más información sobre el uso de `CSS`, consulte [“Aplicación de formato al texto con hojas de estilos en cascada” en la página 143](#).

También puede asignar texto con formato `HTML`, que opcionalmente puede utilizar estilos `CSS`, directamente a un campo de texto. En Flash Player 7 y versiones posteriores, el texto `HTML` que asigna a un campo de texto puede contener elementos multimedia incorporados (clips de película, archivos `SWF` y archivos `JPEG`). El texto se ajustará alrededor del elemento multimedia incorporado, igual que los navegadores Web ajustan texto alrededor del elemento multimedia incorporado en un documento `HTML`. Para más información, consulte [“Etiqueta de imagen \(`<img>`\)” en la página 153](#).

## Utilización de la clase TextField

La clase TextField representa cualquier campo de texto dinámico o seleccionable (editable) que se crea mediante la herramienta Texto de Flash. Utilice los métodos y propiedades de esta clase para controlar los campos de texto durante la ejecución. Los objetos TextField admiten las mismas propiedades que los objetos MovieClip, a excepción de las propiedades `_currentframe`, `_droptarget`, `_framesloaded` y `_totalframes`. Puede obtener y establecer propiedades e invocar métodos para campos de texto de forma dinámica.

Para controlar un campo de texto dinámico o de entrada mediante ActionScript, debe asignarle un nombre de instancia en el inspector de propiedades. Con ello, podrá hacer referencia al campo de texto por el nombre de instancia y utilizar los métodos y propiedades de la clase TextField para controlar el contenido o el aspecto básico del campo de texto. También puede crear objetos TextField durante la ejecución y asignarles nombres de instancia mediante el método `MovieClip.createTextField()`. Para más información, consulte [“Creación de campos de texto durante la ejecución” en la página 141](#).

### Asignación de texto a un campo de texto durante la ejecución

Para asignar texto a un campo de texto, utilice la propiedad `TextField.text`.

**Para asignar texto a un campo de texto durante la ejecución:**

- 1 Con la herramienta Texto, cree un campo de texto en el escenario.
- 2 Con el campo de texto seleccionado, en el inspector de propiedades (Ventana > Propiedades), especifique `headline_txt` en el cuadro de texto Nombre de instancia, que se encuentra debajo del menú emergente Tipo de texto en la parte izquierda del inspector.

Los nombres de instancia pueden constar únicamente de letras, caracteres de subrayado (`_`) y símbolos de dólar (`$`).

- 3 En la línea de tiempo, seleccione el primer fotograma de la capa 1 y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 4 Introduzca el código siguiente en el panel Acciones:  

```
headline_txt.text = "Brasil gana el Mundial";
```
- 5 Seleccione Control > Probar película para probar la película.

### Nombres de instancia y de variable de los campos de texto

En el inspector de propiedades también puede asignar un nombre de variable a un campo de texto dinámico o de entrada, así como un nombre de instancia. Con ello, podrá hacer referencia al nombre de variable del campo de texto en ActionScript, cuyo valor determina el contenido del campo de texto. Tenga en cuenta que el nombre de instancia y el nombre de variable de un campo de texto son distintos.

Utilice el nombre de instancia asignado a un campo de texto para invocar métodos y obtener y establecer las propiedades de dicho campo de texto. El nombre de variable de un campo de texto es sencillamente una referencia de variable al texto contenido en dicho campo, no una referencia a un objeto.

Por ejemplo, si ha asignado el nombre de variable `mytextVar` a un campo de texto, puede establecer el contenido del campo de texto mediante el código siguiente:

```
var mytextVar = "Esto es lo que aparecerá en el campo de texto";
```

No obstante, no puede utilizar la variable `myTextVar` para establecer la misma propiedad de texto del campo de texto en otro texto.

```
//Esto no funcionará
myTextVar.text = "Una variable de campo de texto no es una referencia de
objeto";
```

Como norma general, utilice la propiedad `TextField.text` para controlar el contenido de un campo de texto, a menos que se vaya a utilizar en una versión de Flash Player que no admita la clase `TextField`. Esto reducirá las probabilidades de que se produzcan conflictos de nombres de variables, que podrían provocar un comportamiento inesperado durante la ejecución.

## Creación de campos de texto durante la ejecución

Puede utilizar el método `createTextField()` de la clase `MovieClip` para crear un campo de texto vacío en el escenario durante la ejecución. Este campo de texto nuevo se asocia a la línea de tiempo del clip de película que llama al método. El método `createTextField()` utiliza la sintaxis siguiente:

```
movieClip.createTextField(instanceName, depth, x, y, width, height)
```

Por ejemplo, el código siguiente crea un campo de texto de 300 x 100 píxeles denominado `test_txt` en el punto (0,0) y una profundidad (orden *z*) de 10.

```
_root.createTextField("test_txt", 10, 0, 0, 300, 100);
```

Utilice el nombre de instancia especificado en la llamada `createTextField()` para acceder a los métodos y propiedades de la clase `TextField`. Por ejemplo, el código siguiente crea un campo de texto denominado `test_txt` y, a continuación, modifica sus propiedades para que sea un campo de texto multilínea con ajuste de texto que se expanda para adaptarse al texto insertado.

Finalmente, asigna texto a la propiedad `text` del campo de texto.

```
_root.createTextField("test_txt", 10, 0, 0, 100, 50);
test_txt.multiline = true;
test_txt.wordWrap = true;
test_txt.autoSize = true;
test_txt.text = "Crear nuevos campos de texto con el método
MovieClip.createTextField.";
```

Puede utilizar el método `TextField.removeTextField()` para eliminar un campo de texto creado con `createTextField()`. El método `removeTextField()` no funciona en los campos de texto colocados por la línea de tiempo durante la edición.

Para más información, consulte [MovieClip.createTextField\(\) en la página 499](#) y [TextField.removeTextField\(\) en la página 700](#).

## Utilización de la clase TextFormat

Puede utilizar la clase `TextFormat` de `ActionScript` para establecer propiedades de formato de un campo de texto. La clase `TextFormat` incorpora información de formato de caracteres y párrafos. La información de formato de caracteres describe el aspecto de caracteres individuales: nombre de fuente, tamaño en puntos, color y una URL asociada. La información de formato de párrafo describe el aspecto de un párrafo: margen izquierdo, margen derecho, sangría de la primera línea, y alineación a la izquierda, a la derecha o centrado.

Para utilizar la clase `TextFormat`, primero debe crear un objeto `TextFormat` y establecer los estilos de formato de carácter y de párrafo. A continuación, aplique el objeto `TextFormat` a un campo de texto mediante el método `TextField.setTextFormat()` o `TextField.setNewTextFormat()`.

El método `setTextFormat()` cambia el formato de texto aplicado a caracteres individuales, a grupos de caracteres, o a todo el cuerpo del texto de un campo de texto. No obstante, el texto nuevo insertado (como el texto especificado por un usuario o insertado con `ActionScript`) no toma el formato especificado por una llamada `setTextFormat()`. Para especificar el formato predeterminado para el texto que se acaba de insertar, utilice `TextField.setNewTextFormat()`. Para más información, consulte [TextField.setTextFormat\(\) en la página 704](#) y [TextField.setNewTextFormat\(\) en la página 703](#).

#### Para dar formato a un campo de texto con la clase `TextFormat`:

- 1 En un documento nuevo de Flash, cree un campo de texto en el escenario mediante la herramienta Texto. Escriba un texto en el campo de texto en el escenario, como “Texto en negrita y cursiva de 24 puntos”.
- 2 En el inspector de propiedades, escriba `myText_txt` en el cuadro de texto Nombre de instancia, seleccione Dinámico en el menú emergente Tipo de texto y elija Multilínea en el menú emergente Tipo de línea.
- 3 En la línea de tiempo, seleccione el primer fotograma de la capa 1 y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 4 Especifique el código siguiente en el panel Acciones para crear un objeto `TextFormat`, y establezca las propiedades `bold` e `italic` en `true`, y `size` en 24.

```
// Crear objeto TextFormat
var txtfmt_fmt = new TextFormat();
// Especificar formato de párrafo y caracteres
txtfmt_fmt.bold = "true";
txtfmt_fmt.italic = "true";
txtfmt_fmt.size = "24"
```

- 5 Aplique el objeto `TextFormat` al campo de texto que ha creado en el paso 1 utilizando `TextField.setTextFormat()`.

```
myText_txt.setTextFormat(txtfmt_fmt);
```

Esta versión de `setTextFormat()` aplica el formato especificado a todo el campo de texto. Hay otras dos versiones de este método que permiten aplicar formato a caracteres individuales o a grupos de caracteres. Por ejemplo, el código siguiente aplica el formato de negrita, cursiva y 24 puntos a los cuatro primeros caracteres especificados en el campo de texto.

```
myText_txt.setTextFormat(txtfmt_fmt, 0, 3);
```

Para más información, consulte [TextField.setTextFormat\(\) en la página 704](#).

- 6 Seleccione Control > Probar película para probar la película.

## Propiedades predeterminadas de los nuevos campos de texto

Los campos de texto creados durante la ejecución mediante `createTextField()` reciben un objeto `TextFormat` predeterminado con las propiedades siguientes:

```
font = "Times New Roman"
size = 12
cvtColor = 0x000000
bold = false
italic = false
underline = false
```

```
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
bullet = false
tabStops = [] (matriz vacía)
```

Para ver una lista completa de los métodos `TextFormat` y sus descripciones, véase la entrada [Clase `TextFormat`](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

## Obtención de información de medidas del texto

Puede utilizar el método `TextFormat.getTextExtent()` para obtener medidas detalladas del texto para una cadena que tiene un determinado formato. Por ejemplo, imaginemos que debe crear, durante la ejecución, un objeto `TextField` nuevo que contenga una cantidad arbitraria de texto en fuente Arial en negrita de 24 puntos y con una sangría de 5 píxeles. Debe determinar la anchura o la altura que debe tener el nuevo objeto `TextField` para que se muestre todo el texto. El método `getTextExtent()` proporciona medidas como ascendente, descendente, altura y anchura.

Para más información, consulte [TextFormat.getTextExtent\(\)](#) en la página 722.

## Aplicación de formato al texto con hojas de estilos en cascada

Las hojas de estilos en cascada son un mecanismo para crear estilos de texto que pueden aplicarse a documentos HTML o XML. Una hoja de estilos es una recopilación de reglas de formato que especifican cómo se debe dar formato a elementos HTML o XML. Cada regla asocia un nombre de estilo, o *selector*, con una o varias propiedades de estilo y sus valores. Por ejemplo, el estilo siguiente define un selector denominado `bodyText`.

```
bodyText { text-align: left}
```

Puede crear estilos que redefinan etiquetas de formato HTML incorporadas utilizadas por Flash Player (como `<p>` y `<li>`), crear “clases” de estilos que puedan aplicarse a elementos HTML específicos mediante el atributo `class` de la etiqueta `<p>` o `<span>`, o definir etiquetas nuevas.

Utilice la clase `TextField.StyleSheet` para trabajar con hojas de estilos de texto. Puede cargar estilos desde un archivo CSS externo o crearlos de forma nativa con ActionScript. Para aplicar una hoja de estilos a un campo de texto que contenga texto con formato HTML o XML, utilice la propiedad `TextField.styleSheet`. Los estilos definidos en la hoja de estilos se asignan automáticamente a las etiquetas definidas en el documento HTML o XML.

La utilización de hojas de estilos conlleva tres pasos básicos:

- Crear un objeto de hoja de estilos desde la clase `TextField.StyleSheet`. Véase [“Creación de un objeto de hoja de estilos”](#) en la página 145.
- Añadir estilos al objeto de hoja de estilos, ya sea importándolos desde un archivo CSS externo o bien definiéndolos con ActionScript. Véase [“Carga de archivos CSS externos”](#) en la página 145 y [“Creación de estilos con ActionScript”](#) en la página 146.

- Asignar el objeto de hoja de estilos a un campo de texto que contenga texto con formato XML o HTML. Véase “Aplicación de estilos a un objeto TextField” en la página 146, “Ejemplo de utilización de estilos con HTML” en la página 148 y “Ejemplo de utilización de estilos con XML” en la página 150.

## Propiedades CSS admitidas

Flash Player admite un subconjunto de propiedades en la especificación CSS1 original ([www.w3.org/TR/REC-CSS1](http://www.w3.org/TR/REC-CSS1)). En la tabla siguiente se muestran las propiedades CSS y los valores admitidos, así como los nombres de propiedad de ActionScript correspondientes. Cada nombre de propiedad de ActionScript se deriva del nombre de propiedad CSS correspondiente; el guión se omite y el carácter siguiente va en mayúscula.

Propiedad CSS	Propiedad de ActionScript	Uso y valores admitidos
text-align	textAlign	Los valores reconocidos son <code>left</code> , <code>center</code> y <code>right</code> .
font-size	fontSize	Sólo se utiliza la parte numérica del valor; las unidades (px, pt) no se analizan; los píxeles y los puntos son equivalentes.
text-decoration	textDecoration	Los valores reconocidos son <code>none</code> y <code>underline</code> .
margin-left	marginLeft	Sólo se utiliza la parte numérica del valor. Las unidades (px, pt) no se analizan; los píxeles y los puntos son equivalentes.
margin-right	marginRight	Sólo se utiliza la parte numérica del valor. Las unidades (px, pt) no se analizan; los píxeles y los puntos son equivalentes.
font-weight	fontWeight	Los valores reconocidos son <code>normal</code> y <code>bold</code> .
font-style	fontStyle	Los valores reconocidos son <code>normal</code> e <code>italic</code> .
text-indent	textIndent	Sólo se utiliza la parte numérica del valor. Las unidades (px, pt) no se analizan; los píxeles y los puntos son equivalentes.
font-family	fontFamily	Lista de fuentes que se deben utilizar, separadas por comas, en orden descendente de conveniencia. Se puede utilizar cualquier nombre de familia de fuentes. Si especifica un nombre de fuente genérico, se convertirá a una fuente de dispositivo adecuada. Hay disponibles las siguientes conversiones de fuentes: <code>mono</code> se convierte en <code>_typewriter</code> , <code>sans-serif</code> se convierte en <code>_sans</code> y <code>serif</code> se convierte en <code>_serif</code> .
color	color	Sólo se admiten valores de color hexadecimales. No se admiten los nombres de los colores (como <code>blue</code> ).
display	display	Los valores admitidos son <code>inline</code> , <code>block</code> y <code>none</code> .



## Creación de un objeto de hoja de estilos

Las hojas de estilos CSS están representadas en ActionScript mediante la clase `TextField.StyleSheet`. Esta clase sólo está disponible para los archivos SWF que se vayan a utilizar en Flash Player 7 o versiones posteriores. Para crear un objeto de hoja de estilos, se llama a la función constructora de la clase `TextField.StyleSheet`.

```
var newStyle = new TextField.StyleSheet();
```

Para añadir estilos a un objeto de hoja de estilos, puede cargar un archivo CSS externo en el objeto o definir los estilos en ActionScript. Véase [“Carga de archivos CSS externos” en la página 145](#) y [“Creación de estilos con ActionScript” en la página 146](#).

## Carga de archivos CSS externos

Puede definir estilos en un archivo CSS externo y después cargar dicho archivo en un objeto de hoja de estilos. Los estilos definidos en el archivo CSS se añaden al objeto de hoja de estilos. Para cargar un archivo CSS externo, utilice el método `load()` de la clase `TextField.StyleSheet`. Para determinar cuándo ha finalizado la carga del archivo CSS, utilice el controlador de eventos `onLoad` del objeto de hoja de estilos.

En el ejemplo siguiente, se creará y cargará un archivo CSS externo y se utilizará el método `TextField.StyleSheet.getStyleNames()` para recuperar los nombres de los estilos cargados.

### Para cargar una hoja de estilos externa:

- 1 En el editor XML o de texto que prefiera, cree un nuevo archivo.

- 2 Añada al archivo las siguientes definiciones de estilo:

```
// Nombre de archivo: styles.css
bodyText {
    font-family: Arial,Helvetica,sans-serif;
    font-size: 12px;
}

headline {
    font-family: Arial,Helvetica,sans-serif;
    font-size: 24px;
}
```

- 3 Guarde el archivo CSS como `styles.css`.

- 4 En Flash, cree un documento FLA.

- 5 En la línea de tiempo (Ventana > Línea de tiempo), seleccione Capa 1.

- 6 Abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).

- 7 Añada el código siguiente al panel Acciones:

```
var css_styles = new TextField.StyleSheet();
css_styles.load("styles.css");
css_styles.onLoad = function(ok) {
    if(ok) {
        // mostrar nombres de estilos
        trace(this.getStyleNames());
    } else {
        trace("Error al cargar archivo CSS.");
    }
}
```

- 8 Guarde el archivo en el mismo directorio que contiene el archivo `styles.css`.

## 9 Pruebe la película (Control > Probar película).

Se mostrarán los nombres de los dos estilos en el panel Salida:

```
body  
headLine
```

Si aparece “Error al cargar archivo CSS” en el panel Salida, asegúrese de que el archivo FLA y el archivo CSS se encuentren en el mismo directorio y de que ha escrito correctamente el nombre del archivo CSS.

Al igual que ocurre con los demás métodos de ActionScript que cargan datos a través de la red, el archivo CSS debe residir en el mismo dominio que el archivo SWF que está cargando el archivo. (Véase “Carga de datos de varios dominios” en la [página 198](#).)

## Creación de estilos con ActionScript

Puede crear estilos de texto con ActionScript mediante el método `setStyle()` de la clase `TextField.StyleSheet`. Este método toma dos parámetros: el nombre del estilo y un objeto que define las propiedades de dicho estilo.

Por ejemplo, el código siguiente crea un objeto de hoja de estilos denominado `styles` que define dos estilos idénticos a los que importó anteriormente (véase “Carga de archivos CSS externos” en la [página 145](#)).

```
var styles = new TextField.StyleSheet();  
styles.setStyle("bodyText",  
    {fontFamily: 'Arial,Helvetica,sans-serif',  
      fontSize: '12px'}  
);  
styles.setStyle("headline",  
    {fontFamily: 'Arial,Helvetica,sans-serif',  
      fontSize: '24px'}  
);
```

## Aplicación de estilos a un objeto TextField

Para aplicar un objeto de hoja de estilos a un campo de texto, asigne el objeto de hoja de estilos a la propiedad `styleSheet` del campo de texto.

```
textObj_txt.styleSheet = styleSheetObj;
```

**Nota:** no confunda la *propiedad* `TextField.styleSheet` con la *clase* `TextField.StyleSheet`. La combinación de mayúsculas y minúsculas indica la diferencia.

Cuando asigna un objeto de hoja de estilos a un objeto `TextField`, se producen los cambios siguientes en el comportamiento habitual del campo de texto:

- Las propiedades `text` y `htmlText` del campo de texto, así como las variables asociadas al campo de texto, siempre contienen el mismo valor y se comportan de forma idéntica.
- El campo de texto pasa a ser de sólo lectura y el usuario no puede editarlo.
- Los métodos `setTextFormat()` y `replaceSel()` de la clase `TextField` ya no funcionan con el campo de texto. La única manera de cambiar el campo es modificando las propiedades `text` o `htmlText` del campo de texto o cambiando la variable asociada al campo de texto.
- Todo el texto asignado a la propiedad `text` o a la propiedad `htmlText` del campo de texto o toda variable asociada se almacena literalmente; todo lo escrito en estas propiedades puede recuperarse en el formato de texto original.

## Combinación de estilos

Los estilos CSS en Flash Player son aditivos; es decir, cuando los estilos están anidados, cada nivel de anidación puede aportar información de estilo adicional, que se añade para dar como resultado el formato final.

Por ejemplo, a continuación se muestran datos XML asignados a un campo de texto:

```
<sectionHeading>Esto es una sección</sectionHeading>
<mainBody>Esta es parte del texto del cuerpo, con una palabra
<emphasized>enfaticizada</emphasized>.</mainBody>
```

Para la palabra *enfaticizada* de este ejemplo, el estilo `emphasized` está anidado dentro del estilo `mainBody`. El estilo `mainBody` aporta las reglas de color, tamaño de fuente y decoración. El estilo `emphasized` añade una regla de grosor de fuente a dichas reglas. Se aplicará el formato a la palabra *enfaticizada* mediante una combinación de las reglas especificadas por `mainBody` y `emphasized`.

## Utilización de las clases de estilos

Puede crear “clases” de estilos que puede aplicar a una etiqueta `<p>` o `<span>` utilizando el atributo `class` de cualquiera de las dos etiquetas. Cuando se aplica a una etiqueta `<p>`, el estilo afecta a todo el párrafo. También puede aplicar estilos a un espacio de texto que utiliza una clase de estilo mediante la etiqueta `<span>`.

Por ejemplo, la siguiente hoja de estilos define dos clases de estilos: `mainBody` y `emphasis`.

```
.mainBody {
    font-family: Arial,Helvetica,sans-serif;
    font-size: 24px;
}
.emphasis {
    color: #666666;
    font-style: italic;
}
```

En el texto HTML que asigna a un campo de texto, puede aplicar estos estilos a etiquetas `<p>` y `<span>`, como se indica a continuación.

```
<p class="mainBody">Esto es <span class="emphasis">muy emocionante</span></p>
```

## Estilos para etiquetas HTML incorporadas

Flash Player admite un subconjunto de etiquetas HTML. Para más información, consulte [“Utilización de texto en formato HTML” en la página 151](#). Puede asignar un estilo CSS a cada instancia de una etiqueta HTML incorporada que aparezca en un campo de texto. Por ejemplo, a continuación se define un estilo para la etiqueta HTML incorporada `<p>`. Todas las instancias de dicha etiqueta tendrán el estilo especificado por la regla de estilo.

```
p {
    font-family: Arial,Helvetica,sans-serif;
    font-size: 12px;
    display: inline;
}
```

En la tabla siguiente se muestra a qué etiquetas HTML incorporadas se puede aplicar un estilo y cómo se aplica cada estilo:

Nombre del estilo	Cómo se aplica el estilo
p	Afecta a todas las etiquetas <p>.
body	Afecta a todas las etiquetas <body>. El estilo p, si se especifica, tiene prioridad sobre el estilo body.
li	Afecta a todas las etiquetas de viñeta <li>.
a	Afecta a todas las etiquetas de anclaje <a>.
a:link	Afecta a todas las etiquetas de anclaje <a>. Este estilo se aplica después del estilo a.
a:hover	Se aplica a una etiqueta de anclaje <a> cuando se coloca el puntero del ratón sobre el vínculo. Este estilo se aplica después de los estilos a y a:link. Cuando el puntero del ratón se desplaza fuera del vínculo, el estilo a:hover desaparece del vínculo.
a:active	Se aplica a una etiqueta de anclaje <a> cuando el usuario hace clic en el vínculo. Este estilo se aplica después de los estilos a y a:link. Cuando se suelta el botón del ratón, el estilo a:active desaparece del vínculo.

## Ejemplo de utilización de estilos con HTML

En esta sección se ofrece un ejemplo de utilización de estilos con las etiquetas HTML. Creará una hoja de estilos que contendrá estilos para algunas etiquetas incorporadas y definirá algunas clases de estilos. A continuación, aplicará dicha hoja de estilos a un objeto TextField que contenga texto en formato HTML.

**Para aplicar formato a HTML con una hoja de estilos, haga lo siguiente:**

- 1 En el editor de texto que prefiera, cree un archivo.
- 2 Añada al archivo la siguiente definición de hoja de estilos:

```
p {
  color: #000000;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 12px;
  display: inline;
}

a:link {
  color: #FF0000;
}

a:hover{
  text-decoration: underline;
}

.headline {
  color: #000000;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 18px;
  font-weight: bold;
  display: block;
}
```

```
.byline {
    color: #666600;
    font-style: italic;
    font-weight: bold;
    display: inline;
}
```

Esta hoja de estilos define los estilos para dos etiquetas HTML incorporadas (<p> y <a>) que se aplicarán a todas las instancias de dichas etiquetas. También define dos clases de estilos (.headline y .byline) que se aplicarán a párrafos y espacios de texto específicos.

- 3 Guarde el archivo como `html_styles.css`.
- 4 En Flash, cree un archivo FLA.
- 5 Con la herramienta Texto, cree un campo de texto de unos 400 píxeles de ancho y 300 píxeles de alto.
- 6 Abra el inspector de propiedades (Ventana > Propiedades) y seleccione el campo de texto.
- 7 En el inspector de propiedades, seleccione Texto dinámico en el menú Tipo de texto, seleccione Multilínea en el menú Tipo de línea y elija la opción Generar texto como HTML; a continuación, escriba `news_txt` en el cuadro de texto Nombre de instancia.
- 8 Seleccione el primer fotograma de la Capa 1 de la línea de tiempo (Ventana > Línea de tiempo).
- 9 Abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones) y añada el siguiente código al panel Acciones:

```
// Crear objeto de hoja de estilos
var style_sheet = new TextField.StyleSheet();
// Ubicación del archivo CSS que define los estilos
var css_url = "html_styles.css";
// Crear texto HTML para visualizar
var storyText:String = "<p class='headline'>Flash Player ahora admite hojas
de estilos en cascada</p><p><span class='byline'>San Francisco, CA</span>-
-Macromedia Inc. ha anunciado hoy una nueva versión de Flash Player que
admite estilos de texto CSS (Cascading Style Sheet, hoja de estilos en
cascada). Para más información, visite el <a href='http://
www.macromedia.com'>sitio Web de Macromedia Flash.</a></p>";
// Cargar archivo CSS y definir controlador onLoad:
style_sheet.load(css_url);
style_sheet.onLoad = function(ok) {
    if (ok) {
        // Si la hoja de estilos se ha cargado sin errores,
        // asígnela al objeto de texto,
        // y asigne el texto HTML al campo de texto.
        news_txt.styleSheet = style_sheet;
        news_txt.text = storyText;
    }
};
```

**Nota:** para simplificar este ejemplo, el texto HTML al que se aplica el estilo está incorporado en el script; en una aplicación real, seguramente cargaría el texto desde un archivo externo. Para más información sobre cómo cargar datos externos, consulte el [Capítulo 10, "Trabajo con datos externos"](#), en la página 185.

- 10 Guarde el archivo como `news_html.fla` en el mismo directorio que contiene el archivo CSS creado anteriormente.
- 11 Ejecute la película (Control > Probar película) para ver los estilos aplicados al texto HTML automáticamente.

## Utilización de estilos para definir etiquetas nuevas

Si define un estilo nuevo en una hoja de estilos, dicho estilo puede utilizarse como etiqueta, igual que utilizaría una etiqueta HTML incorporada. Por ejemplo, si una hoja de estilos define un estilo CSS denominado `sectionHeading`, puede utilizar `<sectionHeading>` como un elemento en cualquier campo de texto asociado con la hoja de estilos. Esto permite asignar texto con formato XML arbitrario directamente a un campo de texto, de modo que se aplicará formato al texto de manera automática siguiendo las reglas de la hoja de estilos.

Por ejemplo, la siguiente hoja de estilos crea los estilos `sectionHeading`, `mainBody` y `emphasized`.

```
sectionHeading {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: 18px; display: block
}
mainBody {
    color: #000099;
    text-decoration: underline;
    font-size: 12px; display: block
}
emphasized {
    font-weight: bold; display: inline
}
```

A continuación, puede rellenar un campo de texto asociado con dicha hoja de estilos con el siguiente texto en formato XML:

```
<sectionHeading>Esta es una sección</sectionHeading>
<mainBody>Esto es parte del texto,
con una palabra <emphasized>enfaticada</emphasized>.
</mainBody>
```

## Ejemplo de utilización de estilos con XML

En esta sección, creará el mismo archivo FLA que creó anteriormente (véase [“Ejemplo de utilización de estilos con HTML” en la página 148](#)) pero con texto en formato XML. En este ejemplo, creará la hoja de estilos con `ActionScript`, en lugar de importar los estilos desde un archivo CSS.

### Para aplicar formato a XML con una hoja de estilos:

- 1 En Flash, cree un archivo FLA.
- 2 Con la herramienta Texto, cree un campo de texto de unos 400 píxeles de ancho y 300 píxeles de alto.
- 3 Abra el inspector de propiedades (Ventana > Propiedades) y seleccione el campo de texto.
- 4 En el inspector de propiedades, seleccione Texto dinámico en el menú Tipo de texto, seleccione Multilínea en el menú Tipo de línea y elija la opción Generar texto como HTML; a continuación, escriba `news_txt` en el cuadro de texto Nombre de instancia.
- 5 En la Capa 1 de la línea de tiempo (Ventana > Línea de tiempo), seleccione el primer fotograma.
- 6 Para crear el objeto de hoja de estilos, abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones) y añada el siguiente código al panel Acciones:

```
var xml_styles = new TextField.StyleSheet();
xml_styles.setStyle("mainBody", {
    color:'#000000',
    fontFamily:'Arial,Helvetica,sans-serif',
```

```

        fontSize:'12',
        display:'block'
    });
xml_styles.setStyle("title", {
    color:'#000000',
    fontFamily:'Arial,Helvetica,sans-serif',
    fontSize:'18',
    display:'block',
    fontWeight:'bold'
});
xml_styles.setStyle("byline", {
    color:'#666666',
    fontWeight:'bold',
    fontStyle:'italic',
    display:'inline'
});
xml_styles.setStyle("a:link", {
    color:'#FF0000'
});
xml_styles.setStyle("a:hover", {
    textDecoration:'underline'
});

```

Este código crea un nuevo objeto de hoja de estilos denominado `xml_styles` que define estilos mediante el método `setStyle()`. Los estilos son los mismos que los creados en un archivo CSS externo anteriormente en este capítulo.

- 7 Para crear el texto XML que se debe asignar al campo de texto, añada el código siguiente al panel Acciones:

```

var storyText = "<title>Flash Player ahora admite CSS</title><mainBody><byline>San Francisco, CA</byline>--Macromedia Inc. ha anunciado hoy una nueva versión de Flash Player que admite estilos de texto CSS (Cascading Style Sheet, hoja de estilos en cascada). Para más información, visite el <a href=\"http://www.macromedia.com\">sitio Web de Macromedia Flash</a></mainBody>";

```

- 8 Por último, debe añadir el código siguiente para aplicar el objeto de hoja de estilos a la propiedad `styleSheet` del campo de texto y asignar el texto XML al campo de texto.

```

news_txt.styleSheet = xml_styles;
news_txt.text = storyText;

```

- 9 Guarde el archivo como `news_xml fla`.

- 10 Ejecute la película (Control > Probar película) para ver los estilos aplicados automáticamente al texto del campo.

## Utilización de texto en formato HTML

Flash Player admite un subconjunto de etiquetas HTML estándar, como `<p>` y `<li>`, que puede utilizar para aplicar estilo al texto de cualquier campo de texto dinámico o de introducción de texto. Los campos de texto de Flash Player 7 y versiones posteriores también admiten la etiqueta `<img>`, que permite incorporar archivos JPEG, archivos SWF y clips de película a un campo de texto. Flash Player distribuye automáticamente el texto alrededor de las imágenes incorporadas a los campos de texto, de forma muy similar a como los navegadores Web distribuyen el texto alrededor de las imágenes incorporadas en las páginas HTML. Para más información, consulte [“Incorporación de imágenes, archivos SWF y clips de película en campos de texto” en la página 156](#).

Flash Player también admite la etiqueta `<textformat>`, que permite aplicar estilos de formato de párrafo de la clase `TextFormat` a los campos de texto compatibles con HTML. Para más información, consulte [“Utilización de la clase `TextFormat`” en la página 141](#).

## Información general sobre la utilización del texto con formato HTML

Para utilizar HTML en un campo de texto, debe activar el formato HTML del campo de texto seleccionando la opción **Generar texto como HTML** en el inspector de propiedades o bien estableciendo en la propiedad `html` del campo de texto el valor `true`. Para insertar HTML en un campo de texto, utilice la propiedad `TextField.htmlText`.

Por ejemplo, el código siguiente activa la aplicación de formato HTML para un campo de texto denominado `headline_txt` y, a continuación, asigna HTML al campo de texto.

```
headline_txt.html = true;
headline_txt.htmlText = "<font face='Times New Roman' size='24'>Así es como se
    asigna texto HTML a un campo de texto.</font>";
```

Los atributos de las etiquetas HTML deben escribirse entre comillas simples o dobles. Los valores de los atributos que no estén entre comillas pueden provocar resultados imprevisibles, como una presentación incorrecta del texto. Por ejemplo, Flash Player no generará correctamente el fragmento de código HTML siguiente porque el valor asignado al atributo `align` (`left`) no está entre comillas:

```
textField.htmlText = "<p align=left>Esto es texto alineado a la izquierda</p>";
```

Si escribe los valores de los atributos entre comillas dobles, debe anular el valor de las comillas (`\`). Por ejemplo, estas dos alternativas son correctas:

```
textField.htmlText = "<p align='left'>Aquí se utilizan comillas simples</p>";
textField.htmlText = "<p align=\"left\">Aquí se anula el valor de las comillas
    dobles</p>";
```

No es necesario anular el valor de las comillas dobles si carga texto de un archivo externo; sólo es necesario si asigna una cadena de texto en `ActionScript`.

## Etiquetas HTML admitidas

En esta sección se enumeran las etiquetas HTML incorporadas admitidas por Flash Player. También puede crear nuevos estilos y etiquetas mediante las hojas de estilos en cascada; véase [“Aplicación de formato al texto con hojas de estilos en cascada” en la página 143](#).

### Etiqueta de anclaje (<a>)

La etiqueta `<a>` crea un hipervínculo y admite los atributos siguientes:

- `href` Especifica la URL de la página que se cargará en el navegador. La URL puede ser absoluta o bien relativa a la ubicación del archivo SWF que carga la página.
- `target` Especifica el nombre de la ventana de destino en la que se cargará la página.

Por ejemplo, el fragmento de código HTML siguiente crea el vínculo “Ir a página principal”, que abre [www.macromedia.com](http://www.macromedia.com) en una ventana de navegador nueva.

```
<a href="../home.htm" target="_blank">Ir a página principal</a>
```

También puede definir los estilos `a:link`, `a:hover` y `a:active` para las etiquetas de anclaje mediante las hojas de estilos. Véase [“Estilos para etiquetas HTML incorporadas” en la página 147](#).



## Etiqueta de negrita (<b>)

La etiqueta <b> genera el texto en negrita. Debe haber un tipo de letra en negrita disponible para la fuente utilizada para mostrar el texto.

```
<b>Esto es texto en negrita.</b>
```

## Etiqueta de salto de línea (<br>)

La etiqueta <br> crea un salto de línea en el campo de texto, como se muestra en este ejemplo:

```
Una línea de texto<br>Otra línea de texto<br>
```

## Etiqueta de fuente (<font>)

La etiqueta <font> especifica una fuente o una lista de fuentes para mostrar el texto.

La etiqueta de fuente admite los atributos siguientes:

- **color** Sólo se admiten los valores de colores hexadecimales (#FFFFFF). Por ejemplo, el código HTML siguiente crea texto de color rojo.  

```
<font color="#FF0000">Esto es texto de color rojo</font>
```
- **face** Especifica el nombre de la fuente que se utiliza. También puede especificar una lista de nombres de fuentes separados por comas; en este caso, Flash Player elegirá la primera fuente disponible. Si la fuente especificada no está instalada en el sistema de reproducción o no está incorporada en el archivo SWF, Flash Player elegirá una fuente alternativa.

Ejemplo:

```
<font face="Times, Times New Roman">Esto es Times o bien Times New Roman..</font>
```

Para más información sobre la incorporación de fuentes en las aplicaciones Flash, consulte [TextField.embedFonts en la página 689](#) y “Establecimiento de opciones de texto dinámico y de entrada” en el apartado Utilización de Flash de la Ayuda.

- **size** Especifica el tamaño de la fuente en píxeles. También puede utilizar tamaños en puntos relativos (+2 o -4).  

```
<font size="24" color="#0000FF">Esto es texto de color verde de 24 puntos</font>
```

## Etiqueta de imagen (<img>)

La etiqueta <img> permite incorporar archivos JPEG y SWF externos y clips de película en los campos de texto. El texto fluye automáticamente alrededor de las imágenes incorporadas en los campos de texto. Esta etiqueta está admitida únicamente en los campos de texto dinámicos y de introducción de texto multilinea y con ajuste de texto.

**Para crear un campo de texto multilinea con ajuste de texto, siga uno de estos procedimientos:**

- En el entorno de edición de Flash, seleccione un campo de texto en el escenario y, a continuación, en el inspector de propiedades, seleccione Multilinea en el menú emergente Tipo de texto.
- En el caso de un campo de texto creado durante el tiempo de ejecución con `MovieClip.createTextField()`, defina las propiedades `TextField.multiline` y `TextField.wordWrap` de la instancia del campo de texto nuevo con el valor `true`.

La etiqueta `<img>` tiene un solo atributo necesario, `src`, que especifica la ruta a un archivo JPEG, a un archivo SWF o al identificador de vínculo de un símbolo de clip de película. Todos los demás atributos son opcionales.

La etiqueta `<img>` admite los atributos siguientes:

- `src` Especifica la URL de un archivo JPEG o SWF o el identificador de vínculo de un símbolo de clip de película de la biblioteca. Este atributo es necesario; todos los demás son opcionales. Los archivos externos (JPEG y SWF) no se muestran hasta que no se han descargado completamente.

**Nota:** Flash Player no admite archivos JPEG progresivos.

- `id` Especifica el nombre de la instancia de clip de película (creada por Flash Player) que contiene el archivo JPEG, SWF o el clip de película incorporado. Resulta de utilidad si desea controlar el contenido incorporado con `ActionScript`.
- `width` Anchura, en píxeles, de la imagen, el archivo SWF o el clip de película.
- `height` Altura, en píxeles, de la imagen, archivo SWF o clip de película que se va a insertar.
- `align` Especifica la alineación horizontal de la imagen incorporada en el campo de texto. Los valores válidos son `left` y `right`. El valor predeterminado es `left`.
- `hspace` Especifica la cantidad de espacio horizontal que rodea a la imagen si no aparece texto alguno. El valor predeterminado es 8.
- `vspace` Especifica la cantidad de espacio vertical que rodea a la imagen si no aparece texto alguno. El valor predeterminado es 8.

Para más información y ejemplos sobre la utilización de la etiqueta `<img>`, consulte [“Incorporación de imágenes, archivos SWF y clips de película en campos de texto” en la página 156](#).

## Etiqueta de cursiva (`<i>`)

La etiqueta `<i>` muestra el texto en cursiva. Debe haber un tipo de letra en cursiva disponible para la fuente utilizada.

Eso es muy `<i>interesante</i>`.

El código anterior generaría lo siguiente:

Eso es muy *interesante*.

## Etiqueta de elemento de lista (`<li>`)

La etiqueta `<li>` coloca una viñeta delante del texto incluido en la etiqueta.

Lista de la compra:

```
<li>Manzanas</li>
<li>Naranjas</li>
<li>Limones</li>
```

El código anterior generaría lo siguiente:

Lista de la compra:

- Manzanas
- Naranjas
- Limones

## Etiqueta de párrafo (<p>)

La etiqueta <p> crea un párrafo nuevo. Admite los atributos siguientes:

- **align** Especifica la alineación del texto dentro del párrafo; los valores válidos son `left`, `right` y `center`.
- **class** Especifica una clase de estilos CSS definida por un objeto `TextField.StyleSheet`. Para más información, consulte [“Utilización de las clases de estilos” en la página 147](#).

En el ejemplo siguiente se utiliza el atributo `align` para alinear el texto a la derecha de un campo de texto.

```
textField.htmlText = "<p align='right'>Este texto se alinea a la derecha del  
campo de texto</p>";
```

En el ejemplo siguiente se utiliza el atributo `class` para asignar una clase de estilo de texto a una etiqueta <p>.

```
var myStyleSheet = new TextField.StyleSheet();  
myStyleSheet.secreateTextField("test", 10, 0,0, 300,100);  
createTextField("test", 10, 0,0, 300,100);  
test.styleSheet = myStyleSheet;  
test.htmlText = "<p class='body'>Esto es texto con estilo de cuerpo.</p>";
```

## Etiqueta de espacio (<span>)

La etiqueta <span> sólo se puede utilizar con los estilos de texto CSS. Para más información, consulte [“Aplicación de formato al texto con hojas de estilos en cascada” en la página 143](#). Admite el atributo siguiente:

- **class** Especifica una clase de estilos CSS definida por un objeto `TextField.StyleSheet`. Para más información sobre la creación de clases de estilos de texto, consulte [“Utilización de las clases de estilos” en la página 147](#).

## Etiqueta de formato de texto (<textformat>)

La etiqueta <textformat> permite utilizar un subconjunto de las propiedades de formato de párrafo de la clase `TextFormat` en los campos de texto HTML, incluidos el interlineado, la sangría, los márgenes y las tabulaciones. Puede combinar las etiquetas <textformat> con las etiquetas HTML incorporadas.

La etiqueta <textformat> tiene los atributos siguientes:

- **blockindent** Especifica la sangría de bloque en puntos; corresponde a `TextFormat.blockIndent`. (Véase [TextFormat.blockIndent en la página 720](#).)
- **indent** Especifica la sangría desde el margen izquierdo hasta el primer carácter del párrafo; corresponde a `TextFormat.indent`. (Véase [TextFormat.indent en la página 724](#).)
- **leading** Especifica la cantidad de espacio vertical entre las líneas (interlineado); corresponde a `TextFormat.leading`. (Véase [TextFormat.leading en la página 725](#).)
- **leftmargin** Especifica el margen izquierdo del párrafo en puntos; corresponde a `TextFormat.leftMargin`. (Véase [TextFormat.leftMargin en la página 725](#).)
- **rightmargin** Especifica el margen derecho del párrafo en puntos; corresponde a `TextFormat.rightMargin`. (Véase [TextFormat.rightMargin en la página 725](#).)

- **tabstops** Especifica las tabulaciones personalizadas como una matriz de enteros no negativos; corresponde a `TextFormat.tabStops`. (Véase [TextFormat.tabStops en la página 726](#).)

En el ejemplo de código siguiente se utiliza el atributo `tabstops` de la etiqueta `<textformat>` para crear una tabla de datos con encabezados de fila en negrita, como se muestra a continuación:

Nombre	Edad	Departamento
Tomás	32	IMD
Eduardo	46	Ingeniería

#### Para crear una tabla de datos con formato utilizando tabulaciones:

- 1 Con la herramienta Texto, cree un campo de texto dinámico de unos 300 píxeles de ancho por 100 píxeles de alto.
- 2 En el inspector de propiedades, introduzca `table_txt` en el cuadro de texto Nombre de instancia, seleccione Multilínea en el menú Tipo de línea y seleccione la opción Generar texto como HTML.
- 3 En la línea de tiempo, seleccione el primer fotograma de la capa 1.
- 4 Abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones) e introduzca el código siguiente en ese panel:

```
var rowHeaders = "<b>Nombre\t</b><b>Edad\t</b><b>Departamento";
var row_1 = "Tomás\t31\tIMD";
var row_2 = "Eduardo\t42\tQA";
table_txt.htmlText = "<textformat tabstops='[100, 200]'\t";
table_txt.htmlText += rowHeaders;
table_txt.htmlText += row_1;
table_txt.htmlText += row_2 ;
table_txt.htmlText += "</textformat>";
```

Observe cómo se utiliza la secuencia de escape del carácter de tabulación (`\t`) para añadir tabulaciones entre cada “columna” de la tabla.

- 5 Seleccione Control > Probar película para probar la película.

## Etiqueta de subrayado (<u>)

La etiqueta `<u>` subraya el texto incluido en la etiqueta.

Este texto está `<u>`subrayado`</u>`.

El código anterior generaría lo siguiente:

Este texto está subrayado.

## Incorporación de imágenes, archivos SWF y clips de película en campos de texto

En Flash Player 7 y versiones posteriores, puede utilizar la etiqueta `<img>` para incorporar archivos JPEG, SWF y clips de película en campos de texto dinámicos y de introducción de texto. Para consultar la lista completa de los atributos de la etiqueta `<img>`, véase “[Etiqueta de imagen \(<img>\)](#)” en la página 153.

De forma predeterminada, Flash muestra los medios incorporados en un campo de texto con su tamaño completo. Para especificar las dimensiones de los medios incorporados, utilice los atributos `height` y `width` de la etiqueta `<img>`. (Véase [“Especificación de los valores de altura y anchura” en la página 157.](#))

En general, las imágenes incorporadas a un campo de texto aparecen en la línea que sigue a la etiqueta `<img>`. Sin embargo, si la etiqueta `<img>` es el primer carácter de un campo de texto, la imagen aparece en la primera línea del campo de texto.

## Incorporación de archivos SWF y JPEG

Para incorporar un archivo JPEG o SWF en un campo de texto, especifique la ruta absoluta o relativa del archivo JPEG o SWF en el atributo `src` de la etiqueta `<img>`. Por ejemplo, el código siguiente inserta un archivo JPEG que se encuentra en el mismo directorio que el archivo SWF.

```
textField_txt.htmlText = "<p>Esta es una foto de mis últimas vacaciones:<img  
src='beach.jpg'>";
```

## Incorporación de símbolos de clip de película

Para incorporar un símbolo de clip de película a un campo de texto, especifique el identificador de vínculo del símbolo para el atributo `src` de la etiqueta `<img>`. Para obtener información sobre la definición de un identificador de vínculo, consulte [“Asociación de un símbolo de clip de película al escenario” en la página 129.](#)

Por ejemplo, el código siguiente inserta un símbolo de clip de película con el identificador de vínculo `symbol_ID`.

```
textField_txt.htmlText = "<p>Esto es un símbolo de clip de película:<img  
src='symbol_ID'>";
```

Para que un clip de película incorporado se muestre correctamente y en su totalidad, el punto de registro del símbolo correspondiente debe estar en el punto (0,0).

## Especificación de los valores de altura y anchura

Si especifica los atributos `width` y `height` para una etiqueta `<img>`, en el campo de texto se reserva espacio para el archivo JPEG, el archivo SWF o el clip de película. Una vez que un archivo JPEG o SWF se ha descargado completamente, se mostrará en el espacio reservado. Flash aumenta o reduce la escala de los medios de acuerdo con los valores de `height` y `width`.

Si no especifica los valores de `height` y `width`, no se reserva espacio para los medios incorporados. Una vez que un archivo JPEG o SWF se ha descargado completamente, Flash lo inserta en el campo de texto con su tamaño completo y redistribuye el texto alrededor.

## Control de los medios incorporados con ActionScript

Flash Player crea un clip de película nuevo para cada etiqueta `<img>` y lo incorpora al objeto `TextField`. El atributo `id` de la etiqueta `<img>` permite asignar un nombre de instancia al clip de película creado. Esto permite controlar ese clip de película con ActionScript.

El clip de película creado por Flash Player se añade como clip de película secundario en el campo de texto que contiene la imagen.

Por ejemplo, el código siguiente incorpora un archivo SWF denominado `animation.swf` en el campo de texto `textField_txt` en el nivel 0 y asigna el nombre de instancia `animation_mc` al clip de película que contiene el archivo SWF.

```
_level0.textField_txt.htmlText = "Aquí tenemos una animación interesante: <img  
src='animation.swf' id='animation_mc'>
```

En este caso, la ruta completa del clip de película recién creado es

`_level0.textField_txt.animation_mc`. Por ejemplo, puede asociar el código siguiente a un botón (en la misma línea de tiempo que `textField_txt`) que detenga la cabeza lectora del archivo SWF incorporado.

```
on(press){  
    textField_txt.animation_mc.stop();  
}
```

## Creación de hipervínculos a partir de medios incorporados

Para crear un hipervínculo a partir de un archivo JPEG, un archivo SWF o un clip de película incorporado, coloque la etiqueta `<img>` dentro de una etiqueta `<a>`:

```
textField.htmlText = "Haga clic en la imagen para regresar a la página  
principal<a href='home.htm'><img src='home.jpg'></a>";
```

Cuando el ratón pasa sobre una imagen, un archivo SWF o un clip de película que está entre etiquetas `<a>`, el puntero del ratón se convierte en un icono en forma de mano, al igual que ocurre con los hipervínculos estándar. Las acciones interactivas, como hacer clic con el ratón y presionar teclas, no se registran en los archivos SWF y clips de película que están entre etiquetas `<a>`.

## Creación de texto desplazable

Existen varias maneras de crear texto desplazable en Flash. Puede permitir el desplazamiento en los campos de texto dinámico y de introducción de texto seleccionando la opción Desplazamiento permitido del menú Texto o del menú contextual, o haciendo doble clic en el selector del bloque de texto con la tecla Mayús presionada.

Puede utilizar las propiedades `scroll` y `maxscroll` del objeto `TextField` para controlar el desplazamiento vertical, y las propiedades `hscroll` y `maxhscroll` para controlar el desplazamiento horizontal en un bloque de texto. Las propiedades `scroll` y `hscroll` especifican las posiciones de desplazamiento vertical y horizontal actuales respectivamente; puede leer y escribir estas propiedades. Las propiedades `maxscroll` y `maxhscroll` especifican respectivamente las posiciones de desplazamiento vertical y horizontal máximas; sólo está autorizado a leer estas propiedades.

El componente `TextArea` de Flash MX 2004 proporciona un método sencillo para crear campos de texto desplazable sin apenas crear scripts. Para más información, consulte la entrada Componente `TextArea` en el apartado Utilización de componentes de la Ayuda.

**Para crear un bloque de texto dinámico desplazable, realice una de las acciones siguientes:**

- Haga doble clic con la tecla Mayús presionada en el selector del bloque de texto dinámico.
- Seleccione el bloque de texto dinámico con la herramienta Flecha y seleccione Texto > Desplazamiento permitido.
- Seleccione el bloque de texto dinámico con la herramienta Flecha. Haga clic con el botón derecho del ratón (Windows) o con la tecla Control presionada (Macintosh) en el bloque de texto dinámico y seleccione Texto > Desplazamiento permitido.

### Para utilizar la propiedad scroll para crear texto desplazable:

- 1 Realice uno de los siguientes pasos:
  - Con la herramienta Texto, arrastre un campo de texto en el escenario. Asigne al campo de texto el nombre de instancia `textField` en el inspector de propiedades.
  - Utilice `ActionScript` para crear un campo de texto de forma dinámica con el método `MovieClip.createTextField()`. Asigne al campo de texto el nombre de instancia `textField` como parámetro del método.
- 2 Cree un botón Arriba y un botón Abajo, o seleccione Ventana > Otros paneles > Bibliotecas comunes > Botones y arrastre los botones al escenario.  
Estos botones servirán para desplazar el texto hacia arriba y hacia abajo.
- 3 Seleccione el botón Abajo en el escenario.
- 4 En el panel Acciones (Ventana > Paneles de desarrollo > Acciones), introduzca el código siguiente para desplazar el texto hacia abajo en el campo de texto:

```
on(press){
    textField.scroll += 1;
}
```
- 5 Seleccione el botón Arriba en el escenario.
- 6 En el panel Acciones, introduzca el código siguiente para desplazar el texto hacia arriba:

```
on(press){
    textField.scroll -= 1;
}
```





# CAPÍTULO 9

## Creación de clases con ActionScript 2.0

ActionScript 2.0 es una reorganización del lenguaje ActionScript que proporciona varias funciones de programación nuevas y potentes que poseen otros lenguajes de programación, como Java. ActionScript 2.0 prefiere utilizar estructuras de programa reutilizables, escalables, consistentes y sostenibles. También disminuye el tiempo de desarrollo al proporcionar a los usuarios ayuda para la codificación e información de depuración detallada. ActionScript 2.0 se ajusta a los estándares existentes y se basa en la propuesta ECMAScript 4 ([www.mozilla.org/js/language/es4/](http://www.mozilla.org/js/language/es4/)). ActionScript 2.0 está disponible en Macromedia Flash MX 2004 y en Macromedia Flash MX Professional 2004.

A continuación se describen las características de ActionScript 2.0.

**Modelo común de programación orientada a objetos (OOP)** La principal función de ActionScript 2.0 es un modelo común para crear programas orientados a objetos. ActionScript 2.0 introduce varios nuevos conceptos y palabras clave de programación orientada a objetos, como por ejemplo *clase*, *interfaz* y *paquetes*, con los que estará familiarizado si ha programado alguna vez en código Java.

El modelo OOP que proporciona ActionScript 2.0 es una “formalización sintáctica” del método de cadenas prototipo utilizado en versiones anteriores de Macromedia Flash para crear objetos y establecer herencia.

**Strict data typing** ActionScript 2.0 también permite especificar de forma explícita tipos de datos para variables, parámetros de función y tipos de devolución de funciones. Por ejemplo, en el siguiente código se declara una variable denominada `userName` de tipo `String` (una clase o un tipo de datos de ActionScript incorporado).

```
var userName:String = "";
```

**Advertencias y errores del compilador** Las dos funciones anteriores permiten a la herramienta de edición y al compilador proporcionar advertencias y mensajes de error que le ayudan a encontrar los fallos en las aplicaciones con más rapidez que anteriormente en Flash.

**Atención:** si va a utilizar la sintaxis de ActionScript 2.0, asegúrese de que la configuración de publicación del archivo FLA especifique ActionScript 2.0. Ésta es la configuración predeterminada para los archivos creados con Flash MX 2004. No obstante, si abre un archivo FLA más antiguo que utiliza ActionScript 1 y lo reescribe utilizando ActionScript 2.0, debe cambiar la configuración de publicación del archivo FLA a ActionScript 2.0. Si no lo hace, el archivo FLA no se compilará correctamente, pero no se generarán errores.

# Principios de la programación orientada a objetos

En esta sección se ofrece una breve introducción a los principios relacionados con el desarrollo de programas orientados a objetos. Estos principios se describen más a fondo en el resto de este capítulo junto con detalles sobre su implementación en Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004.

## Objetos

Piense en un objeto del mundo real, como por ejemplo un gato. Podría decirse que un gato tiene propiedades (o estados) como nombre, edad y color; también tiene comportamientos como dormir, comer y ronronear. En el mundo de la programación orientada a objetos, los objetos tienen también propiedades y comportamientos. Al utilizar las técnicas orientadas a objetos, se puede tomar como modelo un objeto del mundo real (como un gato) o un objeto más abstracto (como un proceso químico).

## Clases y miembros de clases

Siguiendo con la analogía del mundo real, piense que hay gatos de distintos colores, edades y nombres con maneras diferentes de comer y ronronear. Pero todos los gatos pertenecen a una determinada clase de objeto, un objeto del tipo “gato”. Cada gato específico (del mundo real) es una instancia del tipo de clase “gato”.

Del mismo modo, en programación orientada a objetos, una *clase* define un diseño para un tipo de objeto. Las características y los comportamientos que pertenecen a una clase se denominan *miembros* de dicha clase. Las características (en el ejemplo del gato, nombre, edad y color) se denominan *propiedades* de la clase, y se representan como variables; los comportamientos (comer, dormir) se denominan *métodos* de la clase, y se representan como funciones.

Por ejemplo, podría crear una clase Person y luego crear una persona específica que sería una instancia de dicha clase, denominada también objeto Person. Este objeto incluiría todas las propiedades y todos los métodos de la clase Person.

En ActionScript se define una clase con la sentencia `class` (véase [“Creación y utilización de clases” en la página 167](#)). ActionScript incluye una serie de clases incorporadas, como MovieClip, TextField y String. Para más información, consulte [Capítulo 6, “Utilización de clases incorporadas”](#), en la página 115.

## Herencia

Una de las principales ventajas de la programación orientada a objetos es que se pueden crear *subclases* de una clase; la subclase *heredará* todas las propiedades y los métodos de la *superclase*. La subclase normalmente define métodos y propiedades adicionales o se *amplía* a la superclase. Las subclases también pueden *suplantar* a (proporcionar sus propias definiciones para) los métodos heredados de una superclase.

Por ejemplo, podría crear una clase Mamífero que defina determinadas propiedades y comportamientos comunes a todos los mamíferos. Luego podría crear una clase Gato que se ampliara a la clase Mamífero. De este modo, la herencia puede impulsar la reutilización del código: en lugar de volver a crear el código común a las dos clases, simplemente puede ampliar a una clase existente. A su vez, otra subclase podría ampliar la clase Gato y así sucesivamente. En una aplicación compleja, determinar la estructura jerárquica de las clases constituye una gran parte del proceso de diseño.

En ActionScript, debe utilizar la palabra clave `extends` para establecer la herencia entre una clase y su superclase. Para más información, consulte [“Creación de subclases” en la página 169](#).

## Interfaces

Las interfaces de la programación orientada a objetos pueden describirse como clases cuyos métodos no se han implementado (definido). Otra clase puede implementar los métodos declarados por la interfaz.

Una interfaz también puede considerarse como un “contrato de programación” que puede utilizarse para aplicar relaciones entre clases que de otro modo no estarían relacionadas. Por ejemplo, suponga que está trabajando con un equipo de programadores y que cada uno de ellos está trabajando en una parte (clase) distinta de la misma aplicación. Al diseñar la aplicación, se acordaron una serie de métodos que utilizarán las distintas clases para comunicarse. Por lo tanto, puede crear una interfaz que declare estos métodos, sus parámetros y sus tipos de devolución. Todas las clases que implementen esta interfaz deberán proporcionar definiciones de dichos métodos; de lo contrario, se producirá un error en el compilador.

También puede utilizar interfaces para proporcionar una forma de “herencia múltiple” limitada, que no se permite en ActionScript 2.0. En la herencia múltiple, una clase se amplía a más de una clase. Por ejemplo, en C++ la clase Gato podría ampliar la clase Mamífero, así como una clase Juguetón, que tiene los métodos PerseguirCola y ComerFriskies. ActionScript 2.0, al igual que Java, no permite que una clase se amplíe directamente a varias clases. No obstante, podría crear una interfaz Juguetón que declare los métodos PerseguirCola y ComerFriskies. Una clase Gato, o cualquier otra clase, podría implementar esta interfaz y proporcionar definiciones para dichos métodos.

Para más información, consulte [“Creación de una interfaz” en la página 173](#).

## Utilización de clases: un ejemplo sencillo

Para aquellos que no están familiarizados con la programación orientada a objetos, esta sección ofrece una visión general del flujo de trabajo que implica la creación y el uso de clases en Flash. Como mínimo, este flujo de trabajo incluye los siguientes pasos:

- 1 Definir una clase en un archivo de clase ActionScript externo.
- 2 Guardar el archivo de clase en un directorio de rutas de clases designado (una ubicación en donde Flash busque las clases).
- 3 Crear una instancia de la clase en otro script, sea en un documento Flash (FLA) o en un archivo de script externo, o crear una subclase basada en la clase original.

En esta sección también se trata una nueva función de ActionScript 2.0 denominada *strict data typing*, que permite especificar el tipo de datos para una variable, un parámetro de función o un tipo de devolución de función.

Aunque esta sección sólo trata las clases, el flujo de trabajo general es el mismo que para utilizar interfaces. Para más información, consulte [“Creación y utilización de interfaces” en la página 173](#).

## Creación de un archivo de clase

Para crear una clase, primero debe crear un archivo de ActionScript (AS) externo. Las clases (e interfaces) sólo pueden definirse en archivos de script externos. Por ejemplo, no se puede definir una clase en un script adjunto a un fotograma o a un botón en un documento de Flash (FLA). Para crear un archivo AS externo, utilice el editor de ActionScript que se incluye con Flash o el editor de texto o código que desee.

**Nota:** el código ActionScript de los archivos externos se compila en un archivo SWF cuando se publica, se exporta o se depura un archivo FLA. Por lo tanto, si realiza modificaciones en un archivo externo deberá guardarlo y volver a compilar todos los archivos FLA que lo utilizan.

En los pasos que se muestran a continuación se creará una clase denominada *Person* que contiene dos propiedades (*age* y *name*) y un único método (*showInfo()*) que muestra los valores de esas propiedades en la ventana Salida.

### Para crear el archivo de clase:

- 1 Cree un nuevo directorio en el disco duro y asígnele el nombre *PersonFiles*. Este directorio incluirá todos los archivos de este proyecto.
- 2 Realice uno de los siguientes pasos:
  - Cree un archivo nuevo en el editor de texto o de código que prefiera.
  - (Sólo para Flash Professional) Seleccione Archivo > Nuevo para abrir el cuadro de diálogo Nuevo documento, seleccione Archivo ActionScript en la lista de tipos de archivos y haga clic en Aceptar. Se abrirá la ventana Script con un archivo en blanco.
- 3 Guarde el archivo como *Person.as* en el directorio *PersonFiles*.
- 4 En la ventana Script, introduzca el siguiente código:

```
class Person {  
}
```

Esto se denomina la *declaración* de clase. En su forma más básica, una declaración de clase consta de la palabra clave `class`, seguida del nombre de la clase (en este caso, *Person*) y después llaves de apertura y cierre (`{}`). Todo lo que está entre llaves se denomina el *cuerpo* de la clase y es donde se definen las propiedades y los métodos de la clase.

**Nota:** el nombre de la clase (*Person*) coincide con el nombre del archivo AS que la contiene (*Person.as*). Esto es sumamente importante, puesto que si estos dos nombres no coinciden la clase no se compilará.

- 5 Para crear las propiedades de la clase *Person*, utilice la palabra clave `var` para definir dos variables denominadas *age* y *name*, tal como se muestra a continuación.

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
}
```

**Sugerencia:** normalmente, las propiedades de una clase se definen al principio del cuerpo de la clase, lo que facilita la comprensión del código, aunque no es obligatorio.

Obsérvese la sintaxis de los dos puntos (`var age:Number` y `var name:String`) que se ha utilizado en las declaraciones de variables. Este es un ejemplo de *strict data typing*. Cuando escribe una variable de esta manera (`var variableName:variableType`), el compilador de ActionScript 2.0 se asegura de que todos los valores asignados a dicha variable coincidan con el tipo especificado. Aunque no es obligatorio utilizar esta sintaxis, es útil y puede facilitar la depuración de los scripts. Para más información, consulte [“Strict data typing” en la página 40](#).

- 6 A continuación creará el método `showInfo()`, que devuelve una cadena con formato predeterminado que contiene los valores de las propiedades `age` y `name`. Añada la definición de la función `showInfo()` al cuerpo de la clase como se muestra a continuación.

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
    // Método para devolver valores de propiedades  
    function showInfo():String {  
        return("Hola, me llamo " + name + " y tengo " + age + " años.");  
    }  
}
```

Obsérvese el uso de la introducción de datos (opcional, aunque recomendada) en la definición de función.

```
function showInfo():String {...}
```

En este caso, lo que se escribe es el valor de devolución de la función `showInfo()` (una cadena).

- 7 El último fragmento de código que añadirá a esta sección es una función especial denominada *función constructora*. En la programación orientada a objetos, la función constructora inicializa cada instancia nueva de una clase.

La función constructora siempre tiene el mismo nombre que la clase. Para crear la función constructora de la clase, añada el siguiente código:

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
    // Método para devolver valores de propiedades  
    function showInfo():String {  
        return("Hola, me llamo " + name + " y tengo " + age + " años.");  
    }  
  
    // Función constructora  
    function Person(myName:String, myAge:Number) {  
        name = myName;  
        age = myAge;  
    }  
}
```

La función constructora `Person()` toma dos parámetros, `myName` y `myAge`, y los asigna a las propiedades `name` y `age`. Los dos parámetros de función se introducen con *strict typing* como `String` y `Number`, respectivamente. Para más información sobre las funciones constructoras, consulte [“Funciones constructoras” en la página 170](#).

**Nota:** si no crea una función constructora se crea una función vacía de forma automática durante la compilación.

- 8 Guarde el archivo como `Person.as` en el directorio `PersonFiles` que ha creado en el paso 1. Si utiliza `Flash MX 2004` (en lugar de `Flash Professional`), pase a la sección siguiente.
- 9 (Sólo para `Flash Professional`) Compruebe la sintaxis del archivo de clase seleccionando `Herramientas > Revisar sintaxis`, o presionando `Ctrl+T` (Windows) o `Comando+T` (Macintosh).  
Si aparece algún error en el panel `Salida`, compare el código de su script con el código final del paso 7, más arriba. Si no puede corregir los errores de código, copie el código completado en el paso 7 del panel `Ayuda`.

## Creación de una instancia de la clase `Person`

En el siguiente paso se crea una instancia de la clase `Person` en otro script, como un script de fotograma en un documento de `Flash` (FLA) u otro script `AS` y se asigna a una variable. Para crear una instancia de una clase personalizada, utilice el operador `new`, tal y como haría al crear una instancia de una clase incorporada de `ActionScript` (como la clase `XML` o `TextField`).

Por ejemplo, el siguiente código crea una instancia de la clase `Person` y la asigna a una variable `newPerson`.

```
var newPerson:Person = new Person("Nate", 32);
```

Este código invoca la función constructora de la clase `Person`, y se pasan como parámetros los valores `"Nate"` y `32`.

La variable `newPerson` se escribe como un objeto `Person`. Introducir los objetos de esta manera permite al compilador asegurarse de que no se intente acceder a las propiedades o los métodos que no están definidos en la clase. La excepción es si se declara que la clase sea dinámica con la palabra clave `dynamic`. Véase [“Creación de clases dinámicas” en la página 180](#).

### Para crear una instancia de la clase `Person` en un documento de `Flash`:

- 1 En `Flash`, seleccione `Archivo > Nuevo`, seleccione `Documento de Flash` en la lista de tipos de documentos y haga clic en `Aceptar`.
- 2 Guarde el archivo como `createPerson fla` en el directorio `PersonFiles` que ha creado previamente.
- 3 Seleccione `Capa 1` en la `Línea de tiempo` y abra el panel `Acciones` (`Ventana > Paneles de desarrollo > Acciones`).
- 4 En el panel `Acciones`, especifique el siguiente código:

```
var person_1:Person = new Person("Nate", 32);  
var person_2:Person = new Person("Jane", 28);  
trace(person_1.showInfo());  
trace(person_2.showInfo());
```

El código anterior crea dos instancias de la clase `Person`, `person_1` y `person_2`, y luego llama al método `showInfo()` en cada instancia.

- 5 Guarde el trabajo y seleccione `Control > Probar película`. Debería ver lo siguiente en el panel `Salida`:

```
Hola, me llamo Nate y tengo 32 años.  
Hola, me llamo Jane y tengo 28 años.
```

Al crear la instancia de una clase llamando a su función constructora, `Flash` busca un archivo `ActionScript` con el mismo nombre que el constructor en un conjunto de ubicaciones de directorios predeterminados. Este grupo de ubicaciones de directorios se conoce de forma colectiva como *ruta de clases* (véase [“Introducción a la ruta de clases” en la página 175](#)).

Ahora debería tener una idea general de cómo crear y utilizar las clases en los documentos de Flash. En el resto de este capítulo se describen más detalladamente las clases e interfaces.

## Creación y utilización de clases

Como se ha dicho con anterioridad, una clase consta de dos partes: la *declaración* y el *cuerpo*. La declaración de la clase consta como mínimo de la sentencia `class` seguida de un identificador para el nombre de clase y, a continuación, llaves de apertura y cierre. Todo lo que está dentro de las llaves es el cuerpo de la clase.

```
class className {  
    // Cuerpo de la clase  
}
```

Sólo puede definir clases en archivos ActionScript (AS). Por ejemplo, no puede definir una clase en un script de fotograma en un archivo FLA. Además, el nombre de clase especificado debe coincidir con el nombre del archivo AS que lo contiene. Por ejemplo, si crea una clase llamada Shape, el archivo AS que contiene la definición de clase debe llamarse Shape.as.

```
// En el archivo Shape.as  
class Shape {  
    // Cuerpo de la clase Shape  
}
```

Todos los archivos de clases AS que cree deben guardarse en uno de los directorios de rutas de clases designadas, directorios en los que Flash busca las definiciones de clases al compilar scripts. (Véase [“Introducción a la ruta de clases” en la página 175.](#))

Los nombres de clase deben ser identificadores; es decir, que el primer carácter debe ser una letra, un carácter de subrayado (`_`) o un símbolo de dólar (`$`), y los caracteres siguientes deben ser una letra, un número, un carácter de subrayado o un símbolo de dólar. Además, el nombre de la clase debe estar completo en el archivo en el que está declarada; es decir, debe reflejar el directorio en el que está almacenada. Por ejemplo, si crea una clase llamada RequiredClass, que esté almacenada en el directorio `myClasses/education/curriculum`, debe declarar la clase en el archivo `RequiredClass.as` de esta forma:

```
class myClasses.education.curriculum.RequiredClass {  
}
```

Por este motivo, es aconsejable planificar una estructura de directorios antes de empezar a crear clases. De otro modo, si decide mover archivos de clases después de crearlos, deberá modificar las sentencias de declaración de clases para reflejar su nueva ubicación.

## Creación de propiedades y métodos

Los miembros de una clase constan de propiedades (declaraciones de variables) y métodos (declaraciones de funciones). Debe declarar las propiedades y los métodos dentro del cuerpo de la clase (entre llaves); de lo contrario, se producirá un error durante la compilación.

Todas las variables declaradas dentro de una clase, pero fuera de una función, son propiedades de la clase. Por ejemplo, la clase `Person` mencionada anteriormente tiene dos propiedades, `age` y `name`, de tipo `Number` y `String`, respectivamente.

```
class Person {  
    var age:Number;  
    var name:String;  
}
```

Asimismo, todas las funciones declaradas dentro de una clase se consideran un método de la clase. En el ejemplo de la clase `Person`, ha creado un único método denominado `showInfo()`.

```
class Person {
    var age:Number;
    var name:String;
    function showInfo() {
        // Definición del método showInfo()
    }
}
```

## Inicialización de propiedades en línea

Puede inicializar propiedades *en línea*, es decir, cuando las declara, con valores predeterminados, como se indica a continuación:

```
class Person {
    var age:Number = 50;
    var name:String = "John Doe";
}
```

Al analizar propiedades en línea, la expresión situada en el lado derecho de una asignación debe ser una *constante en tiempo de compilación*. Es decir, la expresión no puede hacer referencia a nada que se haya establecido o definido durante la ejecución. Las constantes en tiempo de compilación incluyen literales de cadena, números, valores booleanos, `null` y `undefined`, así como funciones constructoras para las siguientes clases incorporadas: `Array`, `Boolean`, `Number`, `Object` y `String`.

Por ejemplo, la siguiente definición de clase inicializa varias propiedades en línea:

```
class CompileTimeTest {
    var foo:String = "my foo"; // Correcto
    var bar:Number = 5; // Correcto
    var bool:Boolean = true; // Correcto
    var name:String = new String("Jane"); // Correcto
    var who:String = foo; // Correcto porque 'foo' es una constante

    var whee:String = myFunc(); // Error: no es una expresión de constante en
    tiempo de compilación
    var lala:Number = whee; // Error: no es una expresión de constante en tiempo
    de compilación
    var star:Number = bar + 25; // Correcto, tanto 'bar' como '25' son constantes

    function myFunc() {
        return "Hola a todos";
    }
}
```

Esta regla sólo se aplica a las variables de instancia (variables que se copian en cada instancia de una clase), no a variables de clase (variables que pertenecen a la clase misma). Para más información sobre estos tipos de variables, consulte [“Miembros de clase e instancia” en la página 171](#).



## Creación de subclases

En la programación orientada a objetos, una subclase puede heredar las propiedades y los métodos de otra clase, denominada superclase. Para crear este tipo de relaciones entre dos clases, debe utilizar la cláusula `extends` de la sentencia `class`. Para especificar una superclase, utilice la sintaxis siguiente:

```
class SubClass extends SuperClass {}
```

La clase que se especifica en `SubClass` hereda todas las propiedades y los métodos definidos por la superclase. Por ejemplo, podría crear una clase `Mammal` que defina propiedades y métodos comunes a todos los mamíferos. Para crear una variación de la clase `Mammal`, como una clase `Marsupial`, podría ampliar la clase `Mammal`, es decir, crear una subclase de la clase `Mammal`.

```
class Marsupial extends Mammal {}
```

La subclase hereda todas las propiedades y los métodos de la superclase, incluidos las propiedades y los métodos que se han declarado como privados utilizando la palabra clave `private`. Para más información sobre variables, consulte [“Control del acceso de miembros” en la página 170](#).

Puede ampliar sus propias clases personalizadas, así como cualquier otra clase incorporada de `ActionScript`, como las clases `XML`, `Sound` o `MovieClip`. Al ampliar una clase incorporada de `ActionScript`, la clase personalizada hereda todos los métodos y las propiedades de la clase incorporada.

Por ejemplo, el código siguiente define la clase `JukeBox`, que amplía la clase `Sound` incorporada. Define una matriz llamada `songList` y un método denominado `playSong()` que reproduce una canción e invoca el método `loadSound()`, que hereda de la clase `Sound`.

```
class JukeBox extends Sound {  
    var songList:Array = new Array("beethoven.mp3", "bach.mp3", "mozart.mp3");  
    function playSong(songID:Number) {  
        this.loadSound(songList[songID]);  
    }  
}
```

Si no efectúa una llamada a `super()` en la función constructora de una subclase, el compilador generará de manera automática una llamada al constructor de su superclase inmediata sin ningún parámetro como primera sentencia de la función. Si la superclase no tiene un constructor, el compilador crea una función constructora vacía y, a continuación, genera una llamada a esa función desde la subclase. No obstante, si la superclase toma parámetros en su definición, debe crear un constructor en la subclase y llamar a la superclase con los parámetros necesarios.

No se permite la herencia múltiple (heredar de más de una clase). No obstante, las clases pueden de hecho heredar de múltiples clases si se utilizan sentencias `extends` por separado:

```
// no permitido  
class C extends A, B {}  
// permitido  
class B extends A {}  
class C extends B {}
```

También puede utilizar la palabra clave `extends` para crear subclases de una interfaz:

```
interface iA extends interface iB {}
```

## Funciones constructoras

El *constructor* de una clase es una función especial a la que se llama automáticamente cuando se crea una instancia de una clase mediante el operador `new`. La función constructora tiene el mismo nombre que la clase que la contiene. Por ejemplo, la clase `Person` que se creó con anterioridad contenía la función constructora siguiente:

```
//Función constructora de la clase Person
function Person (myName:String, myAge:Number) {
    name = myName;
    age = myAge;
}
```

Si no se declara de forma explícita ninguna función constructora (es decir, si no se crea una función cuyo nombre coincida con el nombre de la clase), el compilador crea automáticamente una función constructora vacía.

Una clase sólo puede contener una función constructora; `ActionScript 2.0` no admite funciones constructoras sobrecargadas.

## Control del acceso de miembros

De forma predeterminada, una clase puede acceder a cualquier propiedad o método de otra clase: todos los miembros de una clase son *públicos*. No obstante, en algunos casos quizá desee proteger datos o métodos de una clase para que otras clases no puedan acceder a ellos. Deberá hacer que estos miembros sean *privados*, que sólo estén disponibles para la clase que los declara o los define.

Especifique miembros públicos o privados con los atributos de miembro `public` o `private`. Por ejemplo, el siguiente código declara una variable privada (una propiedad) y un método privado (una función).

Por ejemplo, la clase siguiente (`LoginClass`) define una propiedad privada denominada `userName` y un método privado llamado `getUserName()`.

```
class LoginClass {
    private var userName:String;
    private function getUserName() {
        return userName;
    }
    // Constructor:
    function LoginClass(user:String) {
        this.userName = user;
    }
}
```

Los miembros privados (propiedades y métodos) son accesibles sólo para la clase que define dichos miembros y las subclases de esa clase original. Las instancias de la clase original o las de las subclases de dicha clase no pueden acceder de forma privada a propiedades y métodos declarados; es decir, los miembros privados son accesibles sólo dentro de las definiciones de clase, no en el nivel de la instancia.

Por ejemplo, se puede crear una subclase de `LoginClass` denominada `NewLoginClass`. Esta subclase puede acceder a la propiedad (`userName`) y al método (`getUserName()`) privados definidos por `LoginClass`.

```
class NewLoginClass extends LoginClass {
    // puede acceder a userName y a getUserName()
}
```

Sin embargo, una instancia de `LoginClass` o `NewLoginClass` no puede acceder a esos miembros privados. Por ejemplo, el código siguiente, añadido a un script de fotograma en un archivo FLA, tendría como resultado un error de compilador que indicaría que `getUserNombre()` es privado y es imposible acceder a él.

```
var loginObject:LoginClass = new LoginClass("Maxwell");
var user = loginObject.getUserNombre();
```

Fíjese también en que el control de acceso de los miembros es una función sólo de compilación; en el tiempo de ejecución, Flash Player no distingue entre los miembros privados y los públicos.

## Miembros de clase e instancia

En la programación orientada a objetos, los miembros (propiedades o métodos) de una clase pueden ser *miembros de instancias* o *miembros de clases*. Los miembros de instancias se crean para cada instancia de la clase, o se copian en ella; en cambio, los miembros de clases sólo se crean una vez por clase. Los miembros de clases también se denominan *miembros estáticos*.

Para invocar un método de instancia o acceder a una propiedad de instancia, debe hacer referencia a una instancia de la clase. Por ejemplo, el siguiente código invoca el método `showInfo()` de una instancia de la clase `MovieClip` denominada `clip_mc`:

```
clip_mc.showInfo();
```

No obstante, los miembros (estáticos) de la clase se asignan a la propia clase, no a una instancia de la clase. Para invocar un método de la clase o para acceder a una propiedad de la clase, debe hacer referencia al nombre de la clase en lugar de a una instancia concreta de la misma:

```
ClassName.classMember;
```

Por ejemplo, la clase `Math` de `ActionScript` consta sólo de propiedades y métodos estáticos. Para llamar a cualquiera de sus métodos, no debe crear una instancia de la clase `Math`. En su lugar, simplemente llame a los métodos en la propia clase `Math`. El siguiente código llama al método `sqrt()` de la clase `Math`:

```
var square_root:Number = Math.sqrt(4);
```

Los miembros de instancia pueden leer miembros estáticos, pero no escribirlos. Los miembros de instancia no son numerables en bucles `for` o `for...in`.

## Creación de miembros de clase

Para indicar que una propiedad de una clase es estática, debe utilizar el modificador `static` como se indica a continuación.

```
static var variableName;
```

También puede declarar métodos de una clase de modo que sean estáticos.

```
static function functionName() {
    // Cuerpo de función
}
```

Los métodos de clase (estáticos) sólo pueden acceder a las propiedades (estáticas) de clase, no a las propiedades de instancia. Por ejemplo, el siguiente código provocará un error de compilador, porque el método de la clase `getName()` hace referencia a la variable de instancia `name`.

```
class StaticTest {
    var name="Ted";
```

```

static function getName() {
    var local_name = name;
    // Error: no se puede acceder a variables de instancia en funciones
    estáticas.
}
}

```

Para resolver este problema, podría hacer que el método fuera un método de instancia, o hacer que la variable fuera una variable de clase.

## Utilización de miembros de clase: un ejemplo sencillo

Una forma de emplear miembros (estáticos) de clase es mantener información de estado sobre una clase y sus instancias. Por ejemplo, suponga que desea mantener un seguimiento del número de instancias que se han creado a partir de una clase concreta. Una forma fácil de hacerlo es utilizar una propiedad de clase que se incremente cada vez que se cree una nueva instancia.

En el siguiente ejemplo, creará una clase denominada `Widget` que define un único contador de instancias estáticas denominado `widgetCount`. Cada vez que se crea una nueva instancia de la clase, el valor de `widgetCount` se incrementa en 1 y el valor actual de `widgetCount` se muestra en el panel Salida.

### Para crear un contador de instancias mediante una variable de clase:

- 1 Cree un nuevo archivo `ActionScript (AS)`.
- 2 Añada el siguiente código al archivo:

```

class Widget {
    static var widgetCount:Number = 0; // Inicializa la variable de clase
    function Widget() {
        trace("Creating widget #" + widgetCount);
        widgetCount++;
    }
}

```

La variable `widgetCount` se declara como estática y, por lo tanto, sólo se inicializa a 0 una vez. Cada vez que se llama a la función constructora de la clase `Widget`, se añade 1 a `widgetCount` y luego se muestra el número de la instancia actual que se está creando.

- 3 Guarde el archivo como `Widget.as`.
- 4 Cree un nuevo documento de Flash (FLA) y guárdelo como `createWidget fla` en el mismo directorio que `Widget.as`.  
En este archivo, creará nuevas instancias de la clase `Widget`.
- 5 En `createWidget fla`, seleccione **Capa 1** en la **Línea de tiempo** y abra el panel **Acciones** (**Ventana > Paneles de desarrollo > Acciones**).
- 6 Añada el siguiente código al panel **Acciones**.

```

//Antes de crear instancias de la clase,
//widgetCount es cero (0)
trace("Widget count at start: " + Widget.widgetCount);
var widget_1 = new Widget();
var widget_2 = new Widget();
var widget_3 = new Widget();

```

7 Guarde el archivo y luego pruébelo (Control > Probar película).

Debería ver lo siguiente en el panel Salida:

```
Widget count at start: 0  
Creating widget # 0  
Creating widget # 1  
Creating widget # 2
```

## Miembros de clases y subclases

Los miembros de clases se propagan en las subclases de la superclase que definen estos miembros. En el ejemplo anterior (véase [“Utilización de miembros de clase: un ejemplo sencillo” en la página 172](#)), se ha utilizado una propiedad de clase para realizar el seguimiento del número de instancias de la clase creada. Podría crear una subclase de la clase Widget, como se muestra a continuación.

```
class SubWidget extends Widget {  
    function SubWidget() {  
        trace("Creating subwidget # "+Widget.widgetCount);  
    }  
}
```

## Creación y utilización de interfaces

En la programación orientada a objetos, una interfaz es como una clase cuyos métodos se han declarado, pero no hacen nada más. Es decir, una interfaz consta de métodos “vacíos”.

Una forma de utilizar las interfaces es aplicar un protocolo entre clases no relacionadas, como se describe a continuación. Por ejemplo, suponga que forma parte de un equipo de programadores y cada uno de ellos está trabajando en una parte distinta (es decir, en una clase distinta) de una aplicación grande. La mayoría de estas clases no están relacionadas, pero es necesario disponer de una forma en la que puedan comunicarse las distintas clases. Es decir, es necesario definir una interfaz, o protocolo de comunicación, que deben seguir todas las clases.

Una forma de llevar esto a cabo es crear una clase Communication que defina todos estos métodos y, a continuación, hacer que cada clase se amplíe a, o herede de, esta superclase. Sin embargo, dado que la aplicación consta de clases que no están relacionadas entre sí, no tiene ningún sentido organizarlas en una jerarquía de clases común. Es mejor crear una interfaz que declare los métodos que estas clases utilizarán para comunicarse y después hacer que cada clase implemente (proporcione sus propias definiciones para) estos métodos.

En general se puede programar correctamente sin utilizar interfaces. Sin embargo, cuando se utilizan adecuadamente, las interfaces pueden hacer que el diseño de las aplicaciones sea más elegante, escalable y sostenible.

### Creación de una interfaz

El proceso de creación de una interfaz es el mismo que para crear una clase. Como ocurre con las clases, sólo puede definir interfaces en archivos de ActionScript (AS) externos. Debe declarar una interfaz con la palabra clave `interface`, seguida del nombre de la interfaz y, a continuación, llaves de apertura y cierre, que definen el cuerpo de la interfaz.

```
interface interfaceName {  
    // Declaraciones del método interface  
}
```

Una interfaz sólo puede contener declaraciones de método (función), que incluyen parámetros, tipos de parámetros y tipos de devolución de función.

Por ejemplo, el siguiente código declara una interfaz denominada `MyInterface` que contiene dos métodos, `method_1()` y `method_2()`. El primer método no acepta parámetros y no tiene ningún tipo de devolución (se especifica como `Void`). La segunda declaración del método acepta un solo parámetro de tipo `String` y especifica un tipo de devolución `Boolean`.

```
interface MyInterface {
    function method_1():Void;
    function method_2(param:String):Boolean;
}
```

Las interfaces no pueden contener asignaciones ni declaraciones de variables. Las funciones declaradas en una interfaz no pueden contener llaves. Por ejemplo, la siguiente interfaz no se compilará.

```
interface BadInterface{
    // Error de compilador: no se permite especificar declaraciones de variables
    en las interfaces.
    var illegalVar;

    // Error de compilador: no se permite especificar cuerpos de función en las
    interfaces.
    function illegalMethod(){
    }
}
```

Las reglas para asignar nombres a interfaces y para almacenarlas en paquetes son las mismas que para las clases; véase [“Creación y utilización de clases” en la página 167](#) y [“Utilización de paquetes” en la página 177](#).

## Interfaces como tipos de datos

Al igual que las clases, una interfaz define un nuevo tipo de datos. Se puede entender que todas las clases que implementan una interfaz son del tipo definido por la interfaz. Esto resulta útil para determinar si un objeto dado implementa una interfaz concreta. Por ejemplo, considere la siguiente interfaz.

```
interface Movable {
    function moveUp();
    function moveDown();
}
```

Ahora fíjese en la clase `Box` que implementa la interfaz `Movable`.

```
class Box implements Movable {
    var x_pos, y_pos;

    function moveUp() {
        // Definición de método
    }
    function moveDown() {
        // Definición de método
    }
}
```

A continuación, en otro script en el que se cree una instancia de la clase `Box`, puede declarar una variable como del tipo `Movable`.

```
var newBox:Movable = new Box();
```

Durante la ejecución en Flash Player 7 y posteriores, puede convertir una expresión en un tipo de interfaz. Si la expresión es un objeto que implementa la interfaz, o tiene una superclase que implementa la interfaz, se devuelve el objeto. De lo contrario, se devuelve `null`. Esto es útil para asegurarse de que un objeto concreto implemente una interfaz determinada.

Por ejemplo, el código siguiente comprueba primero si el nombre de objeto `someObject` implementa la interfaz `Movable` antes de llamar al método `moveUp()` en el objeto.

```
if(Movable(someObject) != null) {  
    someObject.moveUp();  
}
```

## Introducción a la ruta de clases

Para utilizar una clase o una interfaz que ha definido, Flash debe ser capaz de localizar los archivos AS externos que contienen la definición de interfaz o clase. La lista de directorios en los que Flash busca las definiciones de interfaces y clases se denomina *ruta de clases*.

Al crear un archivo de clase `ActionScript`, es necesario guardar el archivo en uno de los directorios especificados en la ruta de clases o en un subdirectorio de ésta. (si lo desea, puede modificar la ruta de clases de forma que incluya la ruta de directorio que desee; véase [“Modificación de la ruta de clases” en la página 176](#)). De lo contrario, Flash no podrá *resolver* ni localizar la clase o interfaz especificada en el script. Los subdirectorios que se crean dentro de un directorio de rutas de clases se denominan *paquetes* y permiten organizar las clases. Para más información, consulte [“Utilización de paquetes” en la página 177](#).

## Rutas de clases globales y de documentos

Flash tiene dos tipos de rutas de clases: una ruta de clases global y una ruta de clases de documentos. La ruta de clases global se aplica a los archivos externos AS y FLA y se configura en el cuadro de diálogo Preferencias (Edición > Preferencias). La ruta de clases de documentos se aplica sólo a los archivos FLA y se configura en el cuadro de diálogo Configuración de publicación (Archivo > Configuración de publicación) para un archivo FLA concreto.

De forma predeterminada, la ruta de clases global contiene dos rutas de directorio: una ruta relativa que lleva al directorio que contiene el documento actual y el directorio `Classes`, que está en el directorio de configuración del usuario instalado con Flash. A continuación, se muestra la ubicación de este directorio:

- Windows 2000 o Windows XP: `C:\Documents and Settings\<usuario>\Configuración local\Datos de programa\Macromedia\Flash MX2004\<idioma>\Configuration\`
- Windows 98: `C:\Windows\Datos de programa\Macromedia\Flash MX 2004\<idioma>\Configuration\`
- Macintosh OS X: `Disco duro/Usuarios/Library/Application Support/Macromedia/Flash MX 2004/<idioma>/Configuration/`

De forma predeterminada, la ruta de clases de documentos está vacía.

## Cómo resuelve el compilador las referencias de clases

Cuando el compilador intenta resolver referencias de clases en un script FLA, primero busca en la ruta de clases de documentos especificada para dicho FLA. Si no encuentra la clase en esa ruta de clases, o ésta está vacía, Flash busca en la ruta de clases global. Si no se encuentra la clase en la ruta de clases global, se produce un error de compilador.

Cuando Flash intenta resolver las referencias de clases en un script AS, sólo busca en los directorios de la ruta de clases global, puesto que los archivos AS no tienen asociada una ruta de clases de documentos.

## Modificación de la ruta de clases

La ruta de clases global se puede modificar a través del cuadro de diálogo Preferencias. Para modificar la configuración de ruta de clases de documentos, utilice el cuadro de diálogo Configuración de publicación para el archivo FLA. Puede añadir rutas de directorios absolutas (por ejemplo, C:/my\_classes) y rutas de directorios relativas (por ejemplo, ../my\_classes o “.”).

De forma predeterminada, la ruta de clases global contiene una ruta absoluta (el directorio Classes en el directorio de configuración del usuario) y una ruta de clase relativa, marcada con un punto (.), que apunta al directorio de documento actual. Tenga en cuenta que las rutas de clases relativas pueden señalar a distintos directorios, según la ubicación del documento que se está compilando o publicando. Para más información, consulte [“Rutas de clases globales y de documentos” en la página 175](#).

### Para modificar la ruta de clases global:

- 1 Seleccione Edición > Preferencias para abrir el cuadro de diálogo Preferencias.
- 2 Haga clic en la ficha ActionScript y, luego, haga clic en el botón Configuración de ActionScript 2.0.
- 3 Siga uno de estos procedimientos:
  - Para añadir un directorio a la ruta de clases, haga clic en el botón Buscar ruta, vaya al directorio que desee añadir y haga clic en Aceptar.  
También puede hacer clic en el botón Añadir nueva ruta (+) para añadir una nueva línea a la lista de rutas de clases. Haga doble clic en la nueva línea, escriba una ruta relativa o absoluta y haga clic en Aceptar.
  - Para editar un directorio de ruta de clases existente, seleccione la ruta en la lista de rutas de clases, haga clic en el botón Buscar ruta, busque el directorio que desee añadir y haga clic en Aceptar.  
También puede hacer doble clic en la ruta de la lista de rutas de clases, escribir la ruta que desee y hacer clic en Aceptar.
  - Para eliminar un directorio de la ruta de clases, seleccione la ruta en la lista de rutas de clases y haga clic en el botón Quitar del trazado.

### Para modificar la ruta de clases de documentos:

- 1 Seleccione Archivo > Configuración de publicación para abrir el cuadro de diálogo Configuración de publicación.
- 2 Haga clic en la ficha Flash.
- 3 Haga clic en el botón Configuración situado junto al menú emergente Versión de ActionScript.
- 4 Siga uno de estos procedimientos:
  - Para añadir un directorio a la ruta de clases, haga clic en el botón Buscar ruta, vaya al directorio que desee añadir y haga clic en Aceptar.  
También puede hacer clic en el botón Añadir nueva ruta (+) para añadir una nueva línea a la lista de rutas de clases. Haga doble clic en la nueva línea, escriba una ruta relativa o absoluta y haga clic en Aceptar.



- Para editar un directorio de ruta de clases existente, seleccione la ruta en la lista de rutas de clases, haga clic en el botón Buscar ruta, busque el directorio que desee añadir y haga clic en Aceptar.  
También puede hacer doble clic en la ruta de la lista de rutas de clases, escribir la ruta que desee y hacer clic en Aceptar.
- Para eliminar un directorio de la ruta de clases, seleccione la ruta en la lista de rutas de clases y haga clic en el botón Quitar del trazado.

## Utilización de paquetes

Los archivos de clases de ActionScript pueden organizarse en *paquetes*. Un paquete es un directorio que contiene uno o más archivos de clases y que reside en un directorio de rutas de clases designado. (Véase “Introducción a la ruta de clases” en la página 175.) Un paquete puede, a su vez, contener otros paquetes, denominados *subpaquetes*, cada uno de ellos con sus propios archivos de clase.

Los nombres de paquete deben ser identificadores; es decir, que el primer carácter debe ser una letra, un carácter de subrayado (\_) o un símbolo de dólar (\$), y los caracteres siguientes deben ser una letra, un número, un carácter de subrayado o un símbolo de dólar.

Los paquetes suelen utilizarse para organizar clases relacionadas. Por ejemplo, puede tener tres clases relacionadas, Square, Circle y Triangle, que se definen en Square.as, Circle.as y Triangle.as. Sponga que ha guardado los archivos AS en un directorio especificado en la ruta de clases.

```
// En Square.as:  
class Square {}
```

```
// En Circle.as:  
class Circle {}
```

```
// En Triangle.as:  
class Triangle {}
```

Dado que estos tres archivos de clase están relacionados, puede decidir ponerlos en un paquete (directorio) denominado Shapes. En este caso, el nombre de clase calificado contendría la ruta del paquete, así como el nombre de clase simple. Para las rutas de paquetes se utiliza la sintaxis con punto, en la que cada punto indica un subdirectorio.

Por ejemplo, si ha colocado todos los archivos AS que definen una forma en el directorio Shapes, necesitará modificar el nombre de cada archivo de clase para que refleje la nueva ubicación, del modo siguiente:

```
// En Shapes/Square.as:  
class Shapes.Square {}
```

```
// En Shapes/Circle.as:  
class Shapes.Circle {}
```

```
// En Shapes/Triangle.as:  
class Shapes.Triangle {}
```

Para hacer referencia a una clase que reside en un directorio del paquete, puede especificar su nombre de clase completo o importar el paquete mediante la sentencia `import` (ver a continuación).

## Importación de clases

Para hacer referencia a una clase de otro script, la ruta de paquetes de la clase debe especificarse como prefijo para el nombre de clase. La combinación del nombre de una clase y la ruta de paquete es el *nombre de clase completo* de la clase. Si una clase reside en un directorio de rutas de clases de nivel superior y no en un subdirectorio del directorio de rutas de clases, el nombre de clase completo es el nombre de clase.

Para especificar las rutas de paquetes, utilice la notación con puntos para separar nombres de directorios de paquetes. Las rutas de paquetes tienen una estructura jerárquica, en la que cada punto representa un directorio anidado. Por ejemplo, suponga que crea una clase denominada `Data` que reside en un paquete `com/network/` en la ruta de clases. Para crear una instancia de dicha clase, puede especificar el nombre de clase completo, del siguiente modo:

```
var dataInstance = new com.network.Data();
```

También puede utilizar el nombre de clase completo para introducir las variables:

```
var dataInstance:com.network.Data = new Data();
```

Puede utilizar la sentencia `import` para importar paquetes a un script, lo cual le permitirá utilizar un nombre abreviado de clase en lugar de su nombre completo. También puede utilizar el carácter comodín (\*) para importar todas las clases de un paquete.

Por ejemplo, suponga que ha creado una clase denominada `UserClass` que se incluye en la ruta de directorio de paquete `macr/util/users`:

```
// En el archivo macr/util/users/UserClass.as
class macr.util.users.UserClass { ... }
```

Suponga que en otro script ha importado dicha clase mediante la sentencia `import` de la siguiente forma:

```
import macr.util.users.UserClass;
```

Más adelante, en el mismo script puede hacer referencia a dicha clase por su nombre abreviado:

```
var myUser:UserClass = new UserClass();
```

También puede utilizar el carácter comodín (\*) para importar todas las clases de un paquete dado. Por ejemplo, suponga que tiene un paquete llamado `macr.util` que contiene dos archivos de clases `ActionScript`, `foo.as` y `bar.as`. En otro script, podría importar ambas clases de dicho paquete mediante el carácter comodín, como se muestra a continuación:

```
import macr.util.*;
```

A continuación, en el mismo script, podrá hacer referencia directamente a las clases `foo` o `bar`.

```
var myFoo:foo = new foo();
var myBar:bar = new bar();
```

La sentencia `import` sólo se aplica al script actual (fotograma u objeto) en el que se llama. Si una clase importada no se utiliza en un script, no se incluirá en el código de bytes del archivo SWF resultante, y no estará disponible para los archivos SWF que pueda llamar el archivo FLA que contiene la sentencia `import`. Para más información, consulte [import en la página 407](#).

## Métodos get/set implícitos

En la programación orientada a objetos no se aconseja el acceso directo a las propiedades de una clase. Las clases normalmente definen métodos “get” que proporcionan acceso de lectura y métodos “set” que proporcionan acceso de escritura a una propiedad determinada. Por ejemplo, imagine una clase que contiene una propiedad llamada `userName`:

```
var userName:String;
```

En lugar de permitir que las instancias de la clase accedan directamente a esta propiedad (por ejemplo, `obj.userName = "Jody"`), la clase puede utilizar dos métodos, `getUserName` y `setUserName`, que se implementarían del siguiente modo:

```
function getUserName:String() {  
    return userName;  
}  
  
function setUserName(name:String): {  
    userName = name;  
}
```

Como puede ver, `getUserName` devuelve el valor actual de `userName` y `setUserName` establece el valor de `userName` en el parámetro `string` pasado al método. Una instancia de la clase puede entonces utilizar la siguiente sintaxis para obtener o establecer la propiedad `userName`.

```
// Se llama al método "get"  
var name = obj.getUserName();  
// Se llama al método "set"  
obj.setUserName("Jody");
```

Sin embargo, si desea utilizar una sintaxis más concisa, utilice los métodos *get/set implícitos*. Los métodos *get/set implícitos* permiten acceder directamente a las propiedades de clases a la vez que siguen las recomendaciones de OOP.

Para definir estos métodos, utilice los atributos de los métodos `get` y `set`. Debe crear métodos que obtengan o establezcan el valor de una propiedad, así como añadir la palabra clave `get` o `set` antes del nombre del método.

```
function get user():String {  
    return userName;  
}  
  
function set user(name:String):Void {  
    userName = name;  
}
```

Un método `get` no debe aceptar ningún parámetro. Un método `set` debe aceptar exactamente un parámetro necesario. Un método `set` puede tener el mismo nombre que un método `get` en el mismo ámbito. Los métodos `get/set` no pueden tener el mismo nombre que otras propiedades. Por ejemplo, en el código de ejemplo anterior que define los métodos `get` y `set` denominados `user` no podrá tener también una propiedad denominada `user` en la misma clase.

A diferencia de los métodos más corrientes, los métodos `get/set` se invocan sin ningún paréntesis ni argumentos. Por ejemplo, ahora se puede utilizar la siguiente sintaxis para acceder o modificar el valor de `userName` con los métodos `get/set` definidos anteriormente.

```
var name = obj.user;  
obj.user = "Jack";
```

**Nota:** los métodos `get/set implícitos` son la abreviatura sintáctica del método `Object.addProperty()` en ActionScript 1.

## Creación de clases dinámicas

De forma predeterminada, las propiedades y los métodos de una clase están fijos. Es decir, una instancia de una clase no puede crear ni acceder a propiedades ni métodos que la clase no haya declarado o definido originalmente. Por ejemplo, imagine una clase `Person` que defina dos propiedades, `name` y `age`:

```
class Person {  
    var name:String;  
    var age:Number;  
}
```

Si en otro script se crea una instancia de la clase `Person` y se intenta acceder a una propiedad de la clase que no existe, el compilador generará un error. Por ejemplo, el siguiente código crea una nueva instancia de la clase `Person` (`a_person`) y luego intenta asignar un valor a una propiedad denominada `hairColor`, que no existe.

```
var a_person:Person = new Person();  
a_person.hairColor = "blue"; // Error de compilador
```

Este código causa un error de compilador porque la clase `Person` no declara una propiedad denominada `hairColor`. En la mayoría de los casos, esto es exactamente lo que desea que suceda.

Sin embargo, en algunos casos, puede que durante la ejecución desee añadir y acceder a propiedades o métodos de una clase que no se han definido en la definición de clase original. El modificador de clase `dynamic` le permite hacer justamente esto. Por ejemplo, el siguiente código añade el modificador `dynamic` a la clase `Person` mencionada anteriormente:

```
dynamic class Person {  
    var name:String;  
    var age:Number;  
}
```

Ahora, las instancias de la clase `Person` pueden añadir y acceder a propiedades y métodos que no se han definido en la clase original.

```
var a_person:Person = new Person();  
a_person.hairColor = "blue"; // No se produce un error de compilador porque la  
    clase es dinámica
```

Las subclases de clases dinámicas son también dinámicas.

## Cómo se compilan y exportan las clases

De forma predeterminada, las clases utilizadas por un archivo SWF se empaquetan y exportan en el primer fotograma del archivo SWF. También puede especificar el fotograma en el que se empaquetan y exportan sus clases. Esto es útil, por ejemplo, si un archivo SWF utiliza muchas clases que necesiten mucho tiempo para descargarse. Si las clases se exportan en el primer fotograma, el usuario deberá esperar hasta que todo el código de clase se haya descargado para que aparezca el fotograma. Especificando un fotograma posterior en la línea de tiempo, se puede visualizar una breve animación de carga en los primeros fotogramas de la línea de tiempo mientras se descarga el código de clase del fotograma posterior.

**Para especificar el fotograma de exportación para clases para un documento de Flash:**

- 1 Con un archivo FLA abierto, seleccione Archivo > Configuración de publicación.
- 2 En el cuadro de diálogo Configuración de publicación, haga clic en la ficha Flash.
- 3 Haga clic en el botón Configuración situado junto al menú emergente de la versión ActionScript para abrir el cuadro de diálogo Configuración de ActionScript.
- 4 En el cuadro de texto Fotograma de exportación para clases, introduzca el número del fotograma en el que desee exportar el código de clase.  
Si el fotograma especificado no existe en la línea de tiempo, obtendrá un mensaje de error cuando publique el archivo SWF.
- 5 Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de ActionScript y, a continuación, haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de publicación.



## PARTE IV

# Trabajo con datos y medios externos

En esta sección se explica cómo incorporar datos y medios externos a las aplicaciones de Macromedia Flash.

Capítulo 10: Trabajo con datos externos .....	185
Capítulo 11: Trabajo con elementos multimedia externos .....	201





# CAPÍTULO 10

## Trabajo con datos externos

En Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004, puede utilizar ActionScript para cargar datos de fuentes externas en un archivo SWF. También puede enviar datos de un archivo SWF para que los procese un servidor de aplicaciones (por ejemplo, Macromedia ColdFusion MX o Macromedia JRun) u otro tipo de script de servidor, por ejemplo PHP o Perl. Flash Player puede enviar y cargar datos a través de HTTP o HTTPS o desde un archivo de texto local. También puede crear conexiones de socket TCP/IP persistentes para aplicaciones que requieren una latencia baja, como por ejemplo las aplicaciones de chat o los servicios de cotizaciones bursátiles.

A los datos que se cargan en un archivo SWF o se envían desde éste se les puede aplicar el formato XML (Lenguaje extensible de marcado, Extensible Markup Language) o el de pares de nombre/valor.

Flash Player también puede enviar datos a su entorno host o recibir datos de éste (por ejemplo un navegador Web) o de otra instancia de Flash Player del mismo equipo.

De forma predeterminada, un archivo SWF sólo puede acceder a los datos que residen en el mismo dominio (por ejemplo, [www.macromedia.com](http://www.macromedia.com)) donde se ha originado la película Flash. Para más información, consulte [“Funciones de seguridad de Flash Player” en la página 196](#).

### Envío y carga de variables hacia y desde una fuente remota

Un archivo SWF es una ventana que permite capturar y mostrar información, del mismo modo que en una página HTML. Sin embargo, los archivos SWF pueden permanecer cargados en el navegador y actualizarse continuamente con nueva información sin necesidad de volver a cargar toda la página. Las funciones y los métodos de ActionScript permiten enviar información a scripts de servidor, archivos de texto y archivos XML, así como recibir la información de dichas fuentes.

Además, los scripts de servidor pueden solicitar información específica de una base de datos y enviarla a un archivo SWF. Los scripts de servidor pueden estar escritos en muchos lenguajes diferentes: algunos de los más comunes son CFML, Perl, ASP (Páginas de Active Server de Microsoft, Microsoft Active Server Pages) y PHP. El hecho de almacenar información en una base de datos para después recuperarla permite crear contenido dinámico y personalizado para el archivo SWF. Por ejemplo, puede crear un tablero de mensajes, perfiles personales de los usuarios o una cesta de la compra que efectúe un seguimiento de las compras del cliente de modo que se puedan determinar las preferencias del usuario.

Existen varias funciones y métodos de `ActionScript` que permiten pasar información hacia y desde un archivo SWF. Cada función y método utiliza un protocolo para transferir información y requiere que la información tenga un formato específico.

- Las funciones y los métodos `MovieClip` que utilizan el protocolo HTTP o HTTPS para enviar información en formato URL codificado son `getURL()`, `loadVariables()`, `loadVariablesNum()`, `loadMovie()` y `loadMovieNum()`.
- Los métodos `LoadVars` que utilizan el protocolo HTTP o HTTPS para enviar y cargar información en formato URL codificado son `load()`, `send()` y `sendAndLoad()`.
- Los métodos que utilizan el protocolo HTTP o HTTPS para enviar y cargar información como XML son `XML.send()`, `XML.load()` y `XML.sendAndLoad()`.
- Los métodos que crean y utilizan una conexión de socket TCP/IP para enviar y cargar información como XML son `XMLSocket.connect()` y `XMLSocket.send()`.

## Comprobación de los datos cargados

Cada función o método que carga datos en un archivo SWF (excepto `XMLSocket.send()`) es *asíncrono*: los resultados de una acción se devuelven en un tiempo indeterminado.

Para poder utilizar los datos cargados en un archivo SWF, primero debe comprobar que se han cargado. Por ejemplo, no puede cargar variables y manipular sus valores en el mismo script. En el script siguiente, no puede utilizar la variable `lastFrameVisited` hasta que esté seguro de que la variable se ha cargado desde el archivo `myData.txt`:

```
loadVariables("myData.txt", 0);
gotoAndPlay(lastFrameVisited);
```

Cada función o método tiene una técnica específica que puede utilizarse para comprobar los datos que han cargado. Si utiliza `loadVariables()` o `loadMovie()` puede cargar información en un clip de película de destino y utilizar el evento `data` del controlador `onClipEvent()` para ejecutar un script. Si utiliza `loadVariables()` para cargar los datos, el controlador `onClipEvent(data)` se ejecutará al cargarse la última variable. Si utiliza `loadMovie()` para cargar los datos, el controlador `onClipEvent(data)` se ejecutará cada vez que se envíe un fragmento del archivo SWF a Flash Player.

Por ejemplo, la siguiente acción de botón carga las variables del archivo `myData.txt` en el clip de película `loadTargetMC`:

```
on(release) {
    loadVariables("myData.txt", _root.loadTargetMC);
}
```

Un controlador `onClipEvent()` asignado a la instancia `loadTargetMC` utiliza la variable `lastFrameVisited`, que se carga del archivo `myData.txt`. La siguiente acción se ejecutará solamente cuando se hayan cargado todas las variables, incluida `lastFrameVisited`:

```
onClipEvent(data) {
    gotoAndPlay(lastFrameVisited);
}
```

Si utiliza los métodos `XML.load()`, `XML.sendAndLoad()` y `XMLSocket.connect()`, deberá definir un controlador que procese los datos cuando lleguen. Este controlador es una propiedad del objeto `XML` o `XMLSocket` a la que se le asigna una función definida por el usuario. Se llama automáticamente a los controladores cuando se recibe la información. Para el objeto `XML`, utilice `XML.onLoad()` o `XML.onData()`. Para el objeto `XMLSocket`, utilice `XMLSocket.onConnect()`.

Para más información, consulte [“Utilización de la clase XML” en la página 189](#) y [“Utilización de la clase XMLSocket” en la página 192](#).

## Utilización del protocolo HTTP para conectar con scripts de servidor

Las funciones `loadVariables()`, `loadVariablesNum()`, `getURL()`, `loadMovie()` y `loadMovieNum()`, así como los métodos `MovieClip.loadVariables()`, `MovieClip.loadMovie()` y `MovieClip.getURL()`, pueden comunicarse con scripts de servidor a través de los protocolos HTTP y HTTPS. Estas funciones envían todas las variables desde la línea de tiempo a la que están asociadas. Cuando se utilizan como métodos del objeto `MovieClip`, `loadVariables()`, `getURL()` y `loadMovie()` envían todas las variables del clip de película especificado; cada función (o método) gestiona su respuesta del modo siguiente:

- `getURL()` devuelve la información a una ventana del navegador, no a Flash Player.
- `loadVariables()` carga variables en una línea de tiempo o un nivel especificado de Flash Player.
- `loadMovie()` carga un archivo SWF en un nivel o un clip de película especificado de Flash Player.

Cuando utilice `loadVariables()`, `getURL()` o `loadMovie()`, puede especificar varios parámetros:

- *URL* es el archivo en el que residen las variables remotas.
- *Ubicación* es el nivel o destino del archivo SWF que recibe las variables (la función `getURL` no toma este parámetro).

Para más información sobre niveles y destinos, consulte [“Varias líneas de tiempo y niveles” en Utilización de Flash de la Ayuda](#).

- *Variables* establece el método HTTP, ya sea GET o POST, por medio del que se enviarán las variables. Si se omite este parámetro, Flash Player se establece de forma predeterminada como GET, pero no se envía ninguna variable.

Por ejemplo, si desea realizar el seguimiento de las puntuaciones más altas de un juego, puede almacenarlas en un servidor y utilizar `loadVariables()` para cargarlas en el archivo SWF cada vez que alguien juegue una partida. La llamada de función puede ser similar a la siguiente:

```
loadVariables("http://www.mySite.com/scripts/high_score.php", _root.scoreClip, GET);
```

De esta forma se cargan las variables del script PHP llamado `high_score.php` en la instancia del clip de película `scoreClip` mediante el método HTTP GET.

Cualquiera de las variables cargadas con la función `loadVariables()` debe tener el formato MIME estándar *application/x-www-form-urlencoded* (formato estándar que se utiliza en los scripts CGI). El archivo que especifique en el parámetro *URL* de la función `loadVariables()` debe escribir los pares de variable y valor en este formato para que Flash pueda leerlos. Este archivo puede especificar diversas variables; los pares de variable y valor deben estar separados por el carácter ampersand (&) y las palabras contenidas en un valor deben estar separadas por el signo más (+). Por ejemplo, la siguiente frase define varias variables:

```
highScore1=54000&playerName1=rockin+good&highScore2=53455&playerName2=bonehelmet&highScore3=42885&playerName3=soda+pop
```

Para más información, consulte [loadVariables\(\) en la página 429](#), [getURL\(\) en la página 400](#), [loadMovie\(\) en la página 426](#) y la entrada [Clase LoadVars](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

## Utilización de la clase LoadVars

Puede utilizar la clase LoadVars en lugar de `loadVariables()` para transferir variables entre un archivo SWF y un servidor. La clase LoadVars permite enviar todas las variables de un objeto a una URL y cargar todas las variables de una URL en un objeto. La respuesta del servidor desencadena el método `LoadVars.onLoad()` y establece variables en el destino. Puede utilizar LoadVars para obtener información sobre errores e indicaciones de progreso y para empezar a transmitir los datos mientras se descargan.

La clase LoadVars es parecida a la clase XML; utiliza los métodos `load()`, `send()` y `sendAndLoad()` para iniciar la comunicación con el servidor. La principal diferencia entre las clases LoadVars y XML consiste en que los datos de LoadVars son una propiedad del objeto LoadVars, en lugar de un árbol XML DOM (Modelo de Objetos de Documento, Document Object Model) almacenado en el objeto XML.

Para llamar a los métodos del objeto LoadVars, debe crearlo primero. Este objeto es un contenedor que almacena los datos cargados.

El procedimiento siguiente muestra la manera de utilizar un objeto LoadVars para cargar variables desde un archivo de texto y mostrar dichas variables en un campo de texto.

### Para cargar datos con el objeto LoadVars:

- 1 En un editor de texto como el Bloc de notas o SimpleText, cree un archivo de texto y añada el texto siguiente:  
`day=11&month=July&year=2003`
- 2 Guarde el archivo como `date.txt`.
- 3 En Flash, cree un documento nuevo.
- 4 Cree un campo de texto dinámico en el escenario y asígnele el nombre de instancia `date_txt`.
- 5 Seleccione el fotograma 1 en la línea de tiempo y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones) si todavía no está abierto.
- 6 Introduzca los códigos siguientes en el panel Acciones:

```
var dateVars = new LoadVars();
dateVars.onLoad = function(ok) {
    if (ok) {
        date_txt.text = dateVars.day+"/"+dateVars.month+"/"+dateVars.year;
    }
};
dateVars.load("date.txt");
```

Este código carga las variables en `date.txt` (`day`, `month`, `year`) y, a continuación, les aplica formato y las muestra en el campo de texto `date_txt`.

- 7 Guarde el documento como `dateReader.fla` en el mismo directorio que contiene `date.txt` (el archivo de texto que ha guardado en el paso 3).
- 8 Seleccione Control > Probar película para probar el documento.

Para más información, consulte la entrada [Clase LoadVars](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

## Acerca de XML

XML (*Lenguaje extensible de marcado, Extensible Markup Language*) se está convirtiendo en el estándar para el intercambio de datos estructurados en aplicaciones de Internet. Puede integrar datos en Flash con servidores que utilizan tecnología XML para crear aplicaciones complejas, como las aplicaciones de chat o los sistemas de cotización de valores.

En XML, al igual que en HTML, se utilizan etiquetas para *marcar*, o especificar, un cuerpo de texto. En HTML se utilizan etiquetas predefinidas para indicar cómo debe aparecer el texto en un navegador Web (por ejemplo, la etiqueta `<b>` indica que el texto debe aparecer en negrita). En XML, debe definir las etiquetas que identifican el tipo de una parte de datos (por ejemplo, `<password>VerySecret</password>`). XML separa la estructura de la información del modo en el que ésta se muestra, lo que permite que el mismo documento XML se pueda utilizar y reutilizar en diferentes entornos.

Cada etiqueta XML se denomina *nodo*, o elemento. Cada nodo tiene un tipo (1, que indica un elemento XML, o 3, que indica un nodo de texto) y los elementos también pueden tener atributos. Un nodo anidado en otro nodo se denomina *nodo secundario*. La estructura jerárquica de árbol de los nodos se llama XML DOM (Modelo de Objetos de Documento, Document Object Model) y es parecida a JavaScript DOM, que es la estructura de los elementos de un navegador Web.

En el ejemplo siguiente, `<PORTFOLIO>` es el nodo principal, no tiene atributos y contiene el nodo secundario `<HOLDING>` que tiene los atributos `SYMBOL`, `QTY`, `PRICE` y `VALUE`:

```
<PORTFOLIO>
  <HOLDING SYMBOL="RICH"
    QTY="75"
    PRICE="245.50"
    VALUE="18412.50" />
</PORTFOLIO>
```

## Utilización de la clase XML

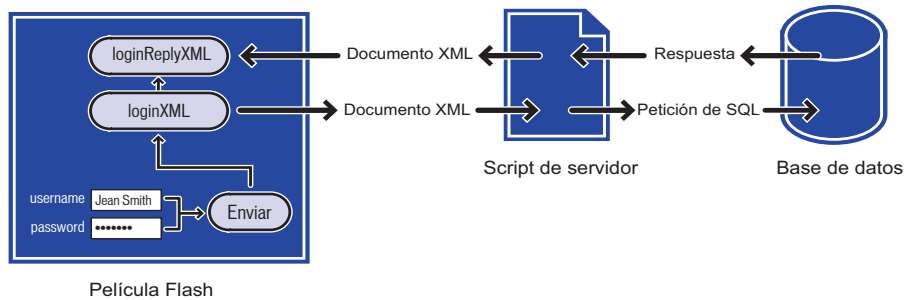
Los métodos de la clase XML de ActionScript (por ejemplo, `appendChild()`, `removeNode()` e `insertBefore()`) permiten estructurar los datos XML en Flash para enviarlos a un servidor y manipular e interpretar los datos XML descargados.

Los métodos de clase XML siguientes envían datos XML a un servidor y los cargan en dicho servidor con el método `POST` de HTTP:

- El método `load()` descarga XML de un URL y lo coloca en un objeto XML de ActionScript.
- El método `send()` pasa un objeto XML a una URL. La información devuelta se envía a otra ventana del navegador.
- El método `sendAndLoad()` envía un objeto XML a una URL. La información devuelta se coloca en un objeto XML de ActionScript.

Por ejemplo, puede crear un sistema de cotización de valores que almacene toda la información (nombres de usuario, contraseñas, ID de sesión, valores de la cartera e información de transacciones) en una base de datos.

El script de servidor que pasa la información entre Flash y la base de datos lee y graba los datos en formato XML. Puede utilizar ActionScript para convertir la información recopilada en el archivo SWF (por ejemplo, un nombre de usuario y una contraseña) en un objeto XML y después enviar los datos al script de servidor como un documento XML. También puede utilizar ActionScript para cargar el documento XML que el servidor devuelve a un objeto XML para utilizarlo en el archivo SWF.



*Flujo y conversión de datos entre una película Flash, un script de servidor y una base de datos.*

La validación de la contraseña para el sistema de cotización de valores requiere dos scripts: una función definida en el fotograma 1 y un script que crea y envía los objetos XML asociados al botón Enviar del formulario.

Cuando los usuarios introducen su información en los campos de texto del archivo SWF con las variables `username` y `password`, las variables deben convertirse a XML antes de pasarlas al servidor. La primera sección del script carga las variables en un objeto XML recién creado llamado `loginXML`. Cuando un usuario hace clic en el botón Enviar, el objeto `loginXML` se convierte en una cadena de XML y se envía al servidor.

El script siguiente está asociado al botón Enviar. Para entender este script, lea las líneas de comentarios (indicadas por los caracteres `//`):

```

on(release) {
    // A. Construir un documento XML con un elemento LOGIN
    loginXML = new XML();
    loginElement = loginXML.createElement("LOGIN");
    loginElement.attributes.username = username;
    loginElement.attributes.password = password;
    loginXML.appendChild(loginElement);

    // B. Construir un objeto XML para albergar la respuesta del servidor
    loginReplyXML = new XML();
    loginReplyXML.onLoad = onLoginReply;

    // C. Enviar el elemento LOGIN al servidor,
    // colocar la respuesta en loginReplyXML
    loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",
        loginReplyXML);
}
  
```

La primera sección del script genera el siguiente código XML cuando el usuario hace clic en el botón Enviar:

```
<LOGIN USERNAME="JeanSmith" PASSWORD="VerySecret" />
```

El servidor recibe el XML, genera una respuesta XML y la envía al archivo SWF. Si se acepta la contraseña, el servidor responde con el mensaje que se muestra a continuación:

```
<LOGINREPLY STATUS="OK" SESSION="rnr6f7vkj2oe14m7jkkycilb" />
```

Este XML incluye un atributo `SESSION` que contiene un ID de sesión exclusivo generado de forma aleatoria que se utilizará en todas las comunicaciones entre el cliente y el servidor durante el resto de la sesión. Si se rechaza la contraseña, el servidor responde con el mensaje que se muestra a continuación:

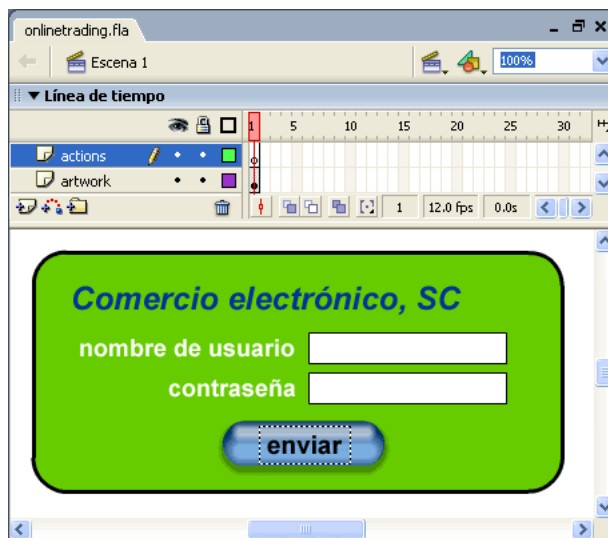
```
<LOGINREPLY STATUS="FAILED" />
```

El nodo XML `LOGINREPLY` debe cargarse en un objeto XML vacío en el archivo SWF. La sentencia siguiente crea el objeto XML `loginReplyXML` para recibir el nodo XML:

```
// B. Construir un objeto XML para albergar la respuesta del servidor
loginReplyXML = new XML();
loginReplyXML.onLoad = onLoginReply;
```

La segunda sentencia asigna la función `onLoginReply()` al controlador `loginReplyXML.onLoad`.

El elemento XML `LOGINREPLY` llega de modo asíncrono, al igual que los datos de la función `loadVariables()`, y se carga en el objeto `loginReplyXML`. Cuando llegan los datos, se llama al controlador `onLoad` del objeto `loginReplyXML`. Debe definir la función `onLoginReply()` y asignarla al controlador `loginReplyXML.onLoad` para que pueda procesar el elemento `LOGINREPLY`. También debe asignar la función `onLoginReply()` al fotograma que contiene el botón **Enviar**.



La función `onLoginReply()` se define en el primer fotograma del archivo SWF. Para entender este script, lea las líneas de comentarios.

```
function onLoginReply() {
    // Obtener el primer elemento XML
    var e = this.firstChild;
    // Si el primer elemento XML es un elemento LOGINREPLY cuyo
    // estado es OK, ir a la pantalla del portafolio. De lo contrario,
    // ir a la pantalla donde falló la conexión y dejar que el usuario lo intente
    de nuevo.
```

```

        if (e.nodeName == "LOGINREPLY" && e.attributes.status == "OK") {
// Guardar el ID de la sesión para futuras comunicaciones con el servidor
        sessionID = e.attributes.session;
// Ir a la pantalla de visualización del portafolio
        gotoAndStop("portfolioView");
        } else {
// Error de inicio de sesión Ir a la pantalla de error de conexión.
        gotoAndStop("loginFailed");
        }
    }
}

```

La primera línea de esta función, `var e = this.firstChild`, utiliza la palabra clave `this` para referirse al objeto XML `loginReplyXML` que acaba de cargarse con XML del servidor. Puede utilizar `this` porque `onLoginReply()` se ha invocado como `loginReplyXML.onLoad`, así que aunque `onLoginReply()` parezca ser una función normal, en realidad se comporta como un método de `loginReplyXML`.

Para enviar el nombre de usuario y la contraseña como XML al servidor y para volver a cargar una respuesta XML en el archivo SWF, puede utilizar el método `sendAndLoad()`, como se muestra a continuación:

```

// C. Enviar el elemento LOGIN al servidor,
// colocar la respuesta en loginReplyXML
loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi", loginReplyXML);

```

**Nota:** este diseño es solamente un ejemplo y Macromedia no garantiza el nivel de seguridad que proporciona. Si está implementando un sistema de seguridad protegido mediante contraseña, asegúrese de que comprende perfectamente la seguridad de la red.

Para más información, consulte “[Integración de XML y Flash en una aplicación Web](http://www.macromedia.com/support/flash/interactivity/xml/)” en la dirección [www.macromedia.com/support/flash/interactivity/xml/](http://www.macromedia.com/support/flash/interactivity/xml/) y la entrada [Clase XML](#) en el [Capítulo 12](#), “[Diccionario de ActionScript](#)”, en la [página 213](#).

## Utilización de la clase XMLSocket

ActionScript proporciona una clase `XMLSocket` incorporada que permite abrir una conexión continua con un servidor. Una conexión de socket permite al servidor publicar (o enviar) la información al cliente tan pronto como ésta se encuentre disponible. Sin una conexión continua, el servidor debe esperar una solicitud HTTP. Esta conexión abierta elimina los problemas de latencia y se usa con frecuencia para las aplicaciones en tiempo real como los chats. Los datos se envían por la conexión de socket como una cadena y deberán estar en formato XML. Puede utilizar la clase XML para estructurar los datos.

Para crear una conexión de socket debe crear una aplicación de servidor que espere la solicitud 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.

Puede utilizar los métodos `connect()` y `send()` de la clase `XMLSocket` para transferir XML desde y hacia un servidor a través de una conexión de socket. El método `connect()` establece una conexión de socket con un puerto de servidor Web. El método `send()` pasa un objeto XML al servidor especificado en la conexión de socket.

Cuando se invoca el método `connect()`, Flash Player abre una conexión TCP/IP con el servidor y la mantiene abierta hasta que produce una de las condiciones que se indican a continuación:

- Se llama al método `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).

En el ejemplo siguiente se crea una conexión de socket XML y se envían los datos desde el objeto XML `myXML`. Para comprender el script, lea las líneas de comentarios (indicadas por los caracteres `//`):

```
// Crear un nuevo objeto XMLSocket
sock = new XMLSocket();
// Llamar a su método connect() para establecer una conexión con el puerto 1024
// del servidor en la URL
sock.connect("http://www.myserver.com", 1024);
// Definir una función para asignarla al objeto sock que gestiona
// la respuesta del servidor. Si la conexión es correcta, enviar el
// objeto myXML. Si falla, mostrar un mensaje de error en un campo
// de texto.
function onSockConnect(success){
    if (success){
        sock.send(myXML);
    } else {
        msg="Error al conectar con "+serverName;
    }
}
// Asignar la función onSockConnect a la propiedad onConnect
sock.onConnect = onSockConnect;
```

Para más información, consulte la entrada [Clase XMLSocket](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

## Envío de mensajes hacia y desde Flash Player

Para enviar mensajes desde un archivo SWF al entorno host correspondiente (por ejemplo, un navegador Web, una película de Macromedia Director o el reproductor Flash Player independiente), puede utilizar la función `fscommand()`. Esta función permite ampliar el archivo SWF mediante las funciones del host. Por ejemplo, puede pasar una función `fscommand()` a una función de JavaScript en una página HTML que abra una nueva ventana del navegador con propiedades específicas.

Para controlar un archivo SWF en Flash Player desde los lenguajes para scripts de los navegadores Web como JavaScript, VBScript y Microsoft JScript, puede utilizar los diferentes métodos de Flash Player (funciones que envían mensajes desde un entorno host al archivo SWF). Por ejemplo, puede tener un vínculo en una página HTML que envíe el archivo SWF a un fotograma específico.

### Utilización de `fscommand()`

Utilice la función `fscommand()` para enviar un mensaje al programa que alberga Flash Player. La función `fscommand()` tiene dos parámetros: *comando* y *argumentos*. Para enviar un mensaje a la versión independiente de Flash Player, debe utilizar comandos y argumentos predefinidos. Por ejemplo, la acción siguiente establece que el reproductor independiente ajuste la escala del archivo SWF al tamaño de pantalla completa al soltar el botón:

```
on(release) {
    fscommand("fullscreen", "true");
}
```

La tabla siguiente muestra los valores que puede especificar para los parámetros *comando* y *argumentos* de la función `fscommand()` para controlar un archivo SWF que se ejecuta en el reproductor independiente (incluidos los proyectores):

Comando	Argumentos	Propósito
<code>quit</code>	Ninguno	Cierra el proyector.
<code>fullscreen</code>	<code>true</code> o <code>false</code>	Si se especifica <code>true</code> , Flash Player se establece en el modo de pantalla completa. Si se especifica <code>false</code> , el reproductor vuelve a la vista de menú normal.
<code>allowscale</code>	<code>true</code> o <code>false</code>	Si se especifica <code>false</code> , el reproductor se establece para que el archivo SWF se dibuje siempre con su tamaño original y nunca se cambie su escala. Si se especifica <code>true</code> , se obliga al archivo SWF a cambiar su escala al 100% del reproductor.
<code>showmenu</code>	<code>true</code> o <code>false</code>	Si se especifica <code>true</code> , se activa el conjunto completo de elementos de menú contextual. Si se especifica <code>false</code> , se atenúan todos los elementos de menú contextual excepto Configuración y Acerca de Flash Player.
<code>exec</code>	Ruta de acceso a la aplicación	Ejecuta una aplicación desde el proyector.

Para utilizar `fscommand()` a fin de enviar un mensaje a un lenguaje de creación de scripts como JavaScript en un navegador Web, puede pasar los dos parámetros que desee a *comando* y *argumentos*. Estos parámetros pueden ser cadenas o expresiones y se utilizan en una función de JavaScript que “captura” o gestiona la función `fscommand()`.

Una función `fscommand()` invoca la función de JavaScript `movienam_DoFSCommand` en la página HTML que incorpora el archivo SWF, donde *movienam* es el nombre de Flash Player asignado mediante el atributo NAME de la etiqueta EMBED o el atributo ID de la etiqueta OBJECT. Si Flash Player tiene asignado el nombre `myMovie`, la función de JavaScript invocada es `myMovie_DoFSCommand`.

**Para utilizar `fscommand()` a fin de abrir un cuadro de mensaje desde un archivo SWF en la página HTML a través de JavaScript siga estos pasos:**

- 1 En la página HTML que incorpora el archivo SWF, añada el siguiente código JavaScript:

```
function theMovie_DoFSCommand(comando, args) {  
    if (command == "messagebox") {  
        alert(args);  
    }  
}
```

Si publica su archivo SWF mediante Flash con la plantilla FSCCommand en el cuadro de diálogo Configuración de publicación en HTML, este código se inserta automáticamente. Los atributos NAME e ID del archivo SWF serán el nombre del archivo. Por ejemplo, para el archivo `myMovie fla`, los atributos se establecerían en `myMovie`. Para más información sobre la publicación, consulte “Publicación” en el apartado Utilización de Flash de la Ayuda.

En las aplicaciones de Microsoft Internet Explorer, otra alternativa es asociar un controlador de eventos directamente a la etiqueta `<SCRIPT>`, como se muestra en el ejemplo siguiente:

```
<Script Language = "JavaScript" event="FSCommand (comando, args)" for=  
    "theMovie">  
...  
</Script>
```

- 2 En el documento de Flash, agregue la función `fscommand()` a un botón, como se muestra en el ejemplo siguiente:

```
on(press){  
    fscommand("messagebox", "Es un cuadro de mensajes invocado desde Flash.")  
}
```

También puede utilizar expresiones para `fscommand()` y parámetros, como se muestra en el ejemplo siguiente:

```
fscommand("messagebox", "Hola, " + name + ", bienvenido/a a nuestra página Web.")
```

- 3 Elija Archivo > Vista previa de publicación > HTML para probar el documento.

La función `fscommand()` puede enviar mensajes a Macromedia Director que Lingo interpreta como cadenas, eventos o código Lingo ejecutable. Si el mensaje es una cadena o un evento, debe escribir el código Lingo para recibirlo de la función `fscommand()` y llevar a cabo una acción en Director. Para más información, consulte el Centro de soporte de Director en [www.macromedia.com/support/director](http://www.macromedia.com/support/director).

En Visual Basic, Visual C++ y otros programas que pueden albergar controles ActiveX, `fscommand()` envía un evento VB con dos cadenas que pueden gestionarse en el lenguaje de programación del entorno. Para obtener más información, utilice las palabras clave *Flash method* para realizar búsquedas en el Centro de Soporte Flash en [www.macromedia.com/go/flash\\_support\\_sp](http://www.macromedia.com/go/flash_support_sp).

## Métodos de Flash Player

Puede utilizar los métodos de Flash Player para controlar un archivo SWF en Flash Player mediante los lenguajes de creación de scripts de navegador Web como JavaScript y VBScript. Al igual que con otros métodos, puede utilizar los métodos de Flash Player para enviar llamadas a los archivos SWF desde un entorno de creación de scripts que no sea ActionScript. Cada método tiene un nombre y la mayoría de los métodos aceptan parámetros. Un parámetro especifica un valor sobre el que opera el método. El cálculo realizado por algunos de los métodos devuelve un valor que puede ser utilizado por el entorno de scripts.

Existen dos tecnologías diferentes que permiten la comunicación entre el navegador y Flash Player: LiveConnect (Netscape Navigator 3.0 o posteriores en Windows 95/98/2000/NT o Power Macintosh) y ActiveX (Internet Explorer 3.0 y posteriores en Windows 95/98/2000/NT). Aunque las técnicas de creación de scripts son similares para todos los navegadores y lenguajes, los controles ActiveX cuentan con propiedades y eventos adicionales.

Para más información, incluída una lista completa de los métodos de creación de scripts de Flash Player, utilice las palabras clave *Flash method* para realizar una búsqueda en el Centro de Soporte de Flash en [www.macromedia.com/go/flash\\_support\\_sp](http://www.macromedia.com/go/flash_support_sp).

## Utilización de los métodos JavaScript de Flash con Flash Player

Flash Player 6 versión 40 y posteriores admiten los métodos JavaScript de Flash y FSCCommand en Netscape 6.2 y posteriores. Las versiones anteriores no admiten ni los métodos JavaScript de Flash ni FSCCommand en Netscape 6.2 o posteriores.

En Netscape 6.2 y posteriores no es necesario establecer `swLiveConnect` en `true`. Sin embargo, el establecimiento de `swLiveConnect` en `true` no tiene efectos negativos.

## Funciones de seguridad de Flash Player

De forma predeterminada, Flash Player 7 o posterior impide que un archivo SWF que se encuentre disponible en un dominio pueda acceder a los datos, objetos o variables de archivos SWF que se encuentran disponibles en otro dominio. No se puede acceder a los objetos ni a las variables de un archivo SWF de otro dominio. Además, el contenido que se carga mediante protocolos que no son seguros (protocolos que no son HTTPS) no puede acceder al contenido cargado mediante un protocolo seguro (HTTPS), aunque ambos contenidos se encuentren en el mismo dominio. Por ejemplo, un archivo SWF ubicado en `http://www.macromedia.com/main.swf` no puede cargar datos de `https://www.macromedia.com/data.txt` sin permiso expreso. Un archivo SWF que se encuentre disponible en un dominio tampoco puede cargar datos (mediante `loadVariables()`, por ejemplo) de otro dominio.

Las direcciones IP numéricas idénticas son compatibles. Sin embargo, los nombres de dominio no son compatibles con las direcciones IP, aunque el nombre de dominio se resuelva en la misma dirección IP.

En las tablas siguientes se muestran ejemplos de dominios compatibles:

<code>www.macromedia.com</code>	<code>www.macromedia.com</code>
<code>data.macromedia.com</code>	<code>data.macromedia.com</code>
<code>65.57.83.12</code>	<code>65.57.83.12</code>

En la tabla siguiente se muestran ejemplos de dominios incompatibles:

<code>www.macromedia.com</code>	<code>data.macromedia.com</code>
<code>macromedia.com</code>	<code>www.macromedia.com</code>
<code>www.macromedia.com</code>	<code>macromedia.com</code>
<code>65.57.83.12</code>	<code>www.macromedia.com</code> (incluso si este dominio se resuelve en <code>65.57.83.12</code> )
<code>www.macromedia.com</code>	<code>65.57.83.12</code> (incluso si <code>www.macromedia.com</code> se resuelve en esta dirección IP)

Para más información sobre cómo permitir a un archivo SWF disponible en un dominio acceder a datos, objetos o variables de archivos SWF que se encuentren disponibles en otro dominio, consulte [“Cómo permitir el acceso a datos entre archivos SWF de varios dominios” en la página 197](#). Para más información sobre cómo permitir a un archivo SWF disponible mediante un protocolo seguro (HTTPS) acceder a datos, objetos o variables de archivos SWF que se encuentren disponibles mediante protocolos no seguros, consulte [“Cómo permitir el acceso entre archivos SWF de un sitio HTTP a un sitio HTTPS” en la página 198](#). Para obtener información sobre cómo permitir a un archivo SWF disponible en un dominio que cargue datos (utilizando `loadVariables()`, por ejemplo) de otro dominio, consulte [“Carga de datos de varios dominios” en la página 198](#).

Para obtener información sobre cómo afectan estos cambios de seguridad al contenido diseñado con Flash MX y anteriores, consulte [“Compatibilidad con modelos de seguridad de Flash Player anteriores” en la página 199](#).

## Cómo permitir el acceso a datos entre archivos SWF de varios dominios

Un archivo SWF puede cargar otro archivo SWF desde cualquier ubicación de Internet. Sin embargo, para que cada archivo pueda acceder a los datos del otro (variables y objetos), ambos tienen que haberse creado en el mismo dominio. De forma predeterminada, en Flash Player 7 y versiones posteriores, los dos dominios deben coincidir exactamente para que los dos archivos puedan compartir datos. No obstante, es posible que un archivo SWF permita acceder a varios archivos SWF que se encuentran disponibles en dominios específicos llamando a `LocalConnection.allowDomain` o `System.security.allowDomain()`.

Por ejemplo, suponga que `main.swf` está disponible en `www.macromedia.com`. El archivo SWF cargará otro SWF (`data.swf`) de `data.macromedia.com` en una instancia de clip de película (`target_mc`).

```
// En macromedia.swf
target_mc.loadMovie("http://data.macromedia.com/data.swf");
```

Además, suponga que `data.swf` define un método llamado `getData()` en su línea de tiempo principal. De forma predeterminada, `main.swf` no puede llamar al método `getData()` definido en `data.swf` una vez que se ha cargado el archivo. Esto se debe a que los dos archivos SWF no residen en el mismo dominio. Por ejemplo, una vez que el archivo `data.swf` se haya cargado, la llamada de método siguiente en el archivo `main.swf` no se podrá realizar.

```
// En macromedia.swf, una vez que el archivo data.swf se haya cargado:
target_mc.getData(); // Esta llamada de método no se podrá realizar
```

Sin embargo, es posible que el archivo `data.swf` permita acceder a archivos SWF que se encuentren disponibles en `www.macromedia.com` mediante el uso del controlador `LocalConnection.allowDomain` o del método `System.security.allowDomain()`, según el tipo de acceso necesario. El código siguiente, añadido a `data.swf`, permite que un archivo SWF que se encuentre disponible en `www.macromedia.com` pueda acceder a sus variables y métodos:

```
// En data.swf
System.security.allowDomain("www.macromedia.com");
my_lc.allowDomain = function(sendingDomain)
    return(sendingDomain=="www.macromedia.com");
}
```

Tenga en cuenta que `allowDomain` permite a cualquier archivo SWF del dominio permitido crear un script de cualquier otro archivo SWF del dominio que permita el acceso, a menos que el archivo SWF al que acceda esté alojado en un sitio que use un protocolo seguro (HTTPS). En ese caso, debe usar `allowInsecureDomain` en lugar de `allowDomain`; véase [“Cómo permitir el acceso entre archivos SWF de un sitio HTTP a un sitio HTTPS”](#) a continuación.

Para más información sobre coincidencias de nombre de dominio, consulte [“Funciones de seguridad de Flash Player”](#) en la página 196.

## Cómo permitir el acceso entre archivos SWF de un sitio HTTP a un sitio HTTPS

Tal como se ha explicado en la sección anterior, debe usar un controlador o método `allowDomain` para permitir que un archivo SWF que se encuentre disponible en un dominio pueda acceder a un archivo SWF que esté disponible en otro dominio. No obstante, si el archivo SWF al que se accede está alojado en un sitio que usa un protocolo seguro (HTTPS), el controlador o método `allowDomain` no permitirá el acceso de un archivo SWF alojado en un sitio que use un protocolo no seguro. Para permitir el acceso en esas condiciones, debe usar la sentencia `LocalConnection.allowInsecureDomain()` o `System.security.allowInsecureDomain()`.

Por ejemplo, si el archivo SWF de `https://www.someSite.com/data.swf` debe permitir el acceso de un archivo SWF de `http://www.someSite.com`, el código siguiente, añadido a `data.swf`, permitirá el acceso:

```
// En data.swf
System.security.allowInsecureDomain("www.someSite.com");
my_lc.allowInsecureDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com");
}
```

## Carga de datos de varios dominios

Un documento de Flash puede cargar datos desde una fuente externa mediante una de las siguientes llamadas para cargar datos: `XML.load()`, `XML.sendAndLoad()`, `LoadVars.load()`, `LoadVars.sendAndLoad()`, `loadVariables()`, `loadVariablesNum()`. Asimismo, un archivo SWF puede importar bibliotecas compartidas en tiempo de ejecución o elementos definidos en otro archivo SWF durante el tiempo de ejecución. De forma predeterminada, los datos o los medios SWF (en el caso de bibliotecas compartidas en tiempo de ejecución) deben residir en el mismo dominio que el archivo SWF que carga dichos medios o datos externos.

Para que los datos y los elementos de las bibliotecas compartidas en tiempo de ejecución puedan ser utilizados por los archivos SWF en diferentes dominios, utilice un *archivo de política para distintos dominios*. Un archivo de política para distintos dominios es un archivo XML que permite al servidor indicar que sus datos y documentos están disponibles para los archivos SWF que se encuentran disponibles en determinados dominios, o en todos los dominios. Cualquier archivo SWF que se encuentre disponible en un dominio especificado por el archivo de política del servidor podrá acceder a los datos y elementos desde dicho servidor.

Cuando un documento de Flash intenta acceder a datos desde otro dominio, Flash Player intenta cargar automáticamente un archivo de política desde dicho dominio. Si el dominio del documento Flash que intenta acceder a los datos se incluye en el archivo de política, se podrá acceder a los datos de forma automática.

Los archivos de política deben denominarse `crossdomain.xml` y deben residir en el directorio raíz del servidor en el que los datos se encuentran disponibles. Los archivos de política sólo funcionan en servidores que se comunican utilizando los protocolos HTTP, HTTPS o FTP. Los archivos de política son específicos del puerto y el protocolo del servidor en el que residen.

Por ejemplo, un archivo de política ubicado en `https://www.macromedia.com:8080/crossdomain.xml` sólo se aplicará a las llamadas para cargar datos realizadas a `www.macromedia.com` en el puerto 8080 utilizando el protocolo HTTPS.

Una excepción a esta regla es cuando se utiliza un objeto XMLSocket para conectar con un servidor de socket de otro dominio. En ese caso, un servidor HTTP que se ejecuta en el puerto 80 del mismo dominio que el servidor de socket deberá proporcionar el archivo de política para la llamada de método.

Un archivo de política XML contiene una sola etiqueta `<cross-domain-policy>`, la cual, a su vez, 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 mediante un solo asterisco, (\*), que incluye todos los dominios y todas las direcciones IP, o mediante un asterisco seguido por un sufijo, que incluye sólo los dominios que terminan con el sufijo especificado. Los sufijos deben empezar por un punto. Sin embargo, los dominios comodín con sufijos pueden coincidir con los dominios formados únicamente por el sufijo sin el punto inicial. Por ejemplo, se considera que `foo.com` forma parte de `*.foo.com`. Los comodines no pueden utilizarse en las especificaciones de dominio IP.

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 a los archivos que se hayan cargado mediante una sintaxis de nombre de dominio. Flash Player no lleva a cabo la resolución DNS.

A continuación, se presenta un ejemplo de archivo de política que permite acceder a documentos de Flash originados en `foo.com`, `friendOfFoo.com`, `*.foo.com` y `105.216.0.40`, a partir de un documento de Flash en `foo.com`:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="www.friendOfFoo.com" />
  <allow-access-from domain="*.foo.com" />
  <allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

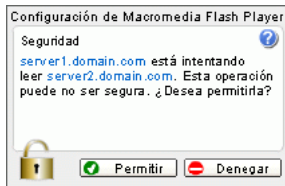
Un archivo de política que no contenga ninguna etiqueta `<allow-access-from>` tiene el mismo efecto que un servidor que no tenga ninguna política.

## Compatibilidad con modelos de seguridad de Flash Player anteriores

Como resultado de los cambios en la función de seguridad de Flash Player (véase [“Funciones de seguridad de Flash Player” en la página 196](#)), el contenido que se ejecuta correctamente en Flash Player 6 o anterior puede no ejecutarse correctamente en Flash Player 7 o posteriores.

Por ejemplo, en Flash Player 6 un archivo SWF que reside en `www.macromedia.com` podría acceder a los datos de un servidor situado en `data.macromedia.com`. Es decir, Flash Player 6 permitiría que un archivo SWF de un dominio cargase datos de un dominio “similar”.

En Flash Player 7 y posteriores, si un archivo SWF de la versión 6 (o anterior) intenta cargar datos de un servidor que reside en otro dominio, y ese servidor no proporciona un archivo de política que permita el acceso desde el dominio de dicho archivo SWF, aparecerá el cuadro de diálogo Configuración de Macromedia Flash Player. El cuadro de diálogo solicita al usuario que permita o deniegue el acceso a datos de varios dominios.



Si el usuario hace clic en Permitir, se permitirá al archivo SWF acceder a los datos solicitados; si el usuario hace clic en Denegar, no se permitirá al archivo SWF acceder a los datos solicitados.

Para impedir que aparezca este cuadro de diálogo, cree un archivo de política de seguridad en el servidor que proporciona los datos. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).



# CAPÍTULO 11

## Trabajo con elementos multimedia externos

Si importa una imagen o un sonido mientras edita un documento en Macromedia Flash MX 2004 o en Macromedia Flash MX Professional 2004, la imagen y el sonido se empaquetan y se almacenan en el archivo SWF al publicarlo. Además de importar elementos multimedia durante el proceso de edición, puede cargar elementos multimedia externos en tiempo de ejecución. Existen varias razones por las cuales es conveniente excluir los archivos multimedia de un documento de Flash.

**Reducir el tamaño de archivo** Si excluye grandes archivos multimedia del documento de Flash y los carga en tiempo de ejecución, puede reducir el tiempo de descarga inicial de las aplicaciones y presentaciones, especialmente si la conexión a Internet es lenta.

**Dividir en módulos grandes presentaciones** Puede dividir una gran presentación o aplicación en archivos SWF independientes y cargarlos como desee en tiempo de ejecución. Esto no sólo reduce el tiempo de descarga inicial, sino que también facilita el mantenimiento y la actualización del contenido de la presentación.

**Separar el contenido de la presentación** Este es un tema común en el desarrollo de aplicaciones, especialmente en las aplicaciones basadas en datos. Por ejemplo, una aplicación de cesta de la compra puede mostrar una imagen JPEG de cada producto. Si carga los archivos JPEG de cada imagen en tiempo de ejecución, puede actualizar fácilmente la imagen de un producto sin modificar el archivo FLA original.

**Beneficiarse de las funciones que son sólo de tiempo de ejecución** Algunas funciones, como por ejemplo la reproducción de flujo FLV y MP3, sólo están disponibles en tiempo de ejecución mediante ActionScript.

### Información general sobre la carga de archivos multimedia externos

Existen cuatro tipos de archivos multimedia que pueden cargarse en una aplicación Flash en tiempo de ejecución: archivos SWF, MP3, JPEG y FLV. Flash Player puede cargar archivos multimedia externos desde cualquier dirección HTTP o FTP, desde un disco local utilizando una ruta relativa o mediante el protocolo `file://`.

Para cargar archivos SWF y JPEG externos, puede utilizar las funciones `loadMovie()` o `loadMovieNum()`, o el método `MovieClip.loadMovie()`. Al cargar un archivo SWF o JPEG, debe especificarse un clip de película o nivel de película como destino de dichos archivos. Para más información sobre cómo cargar archivos SWF y JPEG, consulte [“Carga de archivos SWF y JPEG externos” en la página 202](#).

Para reproducir un archivo MP3 externo (MPEG capa 3), utilice el método `loadSound()` de la clase `Sound`. Este método permite especificar si el archivo MP3 se debe reproducir en flujo o descargar completamente antes de empezar a reproducirse. También puede leer la información ID3 incorporada en los archivos MP3, si están disponibles. Para más información, consulte [“Lectura de etiquetas ID3 en archivos MP3” en la página 204](#).

Flash Video (FLV) es el formato de vídeo nativo que utiliza Flash Player. Los archivos FLV se pueden reproducir a través de HTTP o desde el sistema de archivos local. La reproducción de archivos FLV externos ofrece varias ventajas frente a la incorporación de vídeo en un documento de Flash, como por ejemplo mejor rendimiento y administración de la memoria y velocidades de fotogramas de vídeo y Flash independientes. Para más información, consulte [“Reproducción dinámica de archivos FLV externos” en la página 205](#).

También puede precargar archivos multimedia externos o hacer un seguimiento del progreso de la descarga. Flash Player 7 presenta la clase `MovieClipLoader`, que puede utilizarse para hacer un seguimiento del progreso de la descarga de los archivos SWF o JPEG. Para precargar archivos MP3 y FLV, puede usar el método `getBytesLoaded()` de la clase `Sound` y la propiedad `bytesLoaded` de la clase `NetStream`. Para más información, consulte [“Precarga de archivos multimedia externos” en la página 206](#).

## Carga de archivos SWF y JPEG externos

Para cargar un archivo SWF o JPEG, utilice las funciones globales `loadMovie()` o `loadMovieNum()`, o el método `loadMovie()` de la clase `MovieClip`. Para cargar un archivo SWF o JPEG en un nivel de Flash Player, utilice `loadMovieNum()`. Para cargar un archivo SWF o JPEG en un clip de película de destino, utilice la función o el método `loadMovie()`. En cualquiera de esos casos, el contenido cargado reemplaza al contenido del nivel o clip de película de destino especificado.

Cuando se carga un archivo SWF o JPEG en un clip de película de destino, la esquina superior izquierda del archivo SWF o imagen JPEG se sitúa en el punto de registro del clip de película. Dado que este punto suele ser el centro del clip de película, es posible que el contenido cargado no aparezca centrado. Además, cuando se carga una imagen SWF o JPEG en una línea de tiempo raíz, la esquina superior izquierda de la imagen se sitúa en la esquina superior izquierda del escenario. El contenido cargado hereda la rotación y la escala del clip de película, pero el contenido original del clip de película se elimina.

También puede enviar variables de código de ActionScript con una llamada `loadMovie()` o `loadMovieNum()`. Esto es útil, por ejemplo, si la URL que especifica en la llamada de método es un script del servidor mediante el cual se devuelve un archivo JPEG o SWF según los datos que se envían desde la aplicación Flash.

Para los archivos de imagen, Flash sólo admite el tipo estándar de archivo de imagen JPEG, no admite archivos JPEG progresivos.

Cuando utilice la función global `loadMovie()` o `loadMovieNum()`, especifique el nivel o clip de destino como parámetro. Por ejemplo, el código siguiente carga el archivo `contents.swf` de la aplicación Flash en una instancia de clip de película denominada `target_mc`:

```
loadMovieNum("contents.swf", target_mc);
```

De forma similar, puede utilizar `MovieClip.loadMovie()` para conseguir el mismo resultado:

```
target_mc.loadMovie("contents.swf");
```

El código siguiente carga la imagen JPEG `flowers.jpg` en la instancia del clip de película

```
image_clip:
```

```
image_clip.loadMovie("flowers.jpg");
```

Para más información sobre `loadMovie()`, `loadMovieNum()` y `MovieClip.loadMovie()`, consulte las entradas correspondientes en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Archivos SWF cargados y línea de tiempo raíz

La propiedad `_root` de ActionScript especifica o devuelve una referencia a la línea de tiempo raíz de un archivo SWF. Si carga un archivo SWF en un clip de película de otro archivo SWF, las referencias a `_root` del archivo SWF cargado se resolverán en la línea de tiempo raíz del archivo SWF principal, no en la línea de tiempo del archivo SWF cargado. En ocasiones, esto puede provocar un comportamiento inesperado en tiempo de ejecución, como por ejemplo cuando el archivo SWF principal y el archivo SWF cargado utilizan la propiedad `_root` para especificar una variable.

En Flash Player 7 y versiones posteriores, puede utilizar la propiedad `MovieClip._lockroot` para conseguir que las referencias hechas a `_root` en un clip de película se resuelvan en la línea de tiempo del clip, en lugar de en la línea de tiempo del archivo SWF que contiene dicho clip de película. Para más información, consulte [“Especificación de una línea de tiempo raíz para archivos SWF cargados” en la página 126](#).

## Acceso a los datos de archivos SWF cargados

Un archivo SWF puede cargar otro archivo SWF desde cualquier ubicación de Internet. No obstante, para que un SWF pueda acceder a los datos (variables, métodos, etc.) definidos en el otro SWF, ambos archivos deben originarse en el mismo dominio. En Flash Player 7 y versiones posteriores, se prohíbe la creación de scripts en varios dominios, a menos que se especifique lo contrario en el archivo SWF cargado llamando a `System.security.allowDomain()`.

Para más información, consulte [“Funciones de seguridad de Flash Player” en la página 196](#) y `System.security.allowDomain()` en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Carga de archivos MP3 externos

Para cargar archivos MP3 en tiempo de ejecución, utilice el método `loadSound()` de la clase `Sound`. En primer lugar, cree un objeto `Sound`:

```
var song_1_sound = new Sound();
```

A continuación, utilice el nuevo objeto para llamar a `loadSound()` a fin de cargar un evento o un flujo de sonido. Los sonidos de evento se cargan completamente antes de reproducirse; los flujos de sonido se reproducen a medida que se van descargando. Puede establecer el parámetro *isStreaming* del método `loadSound()` para especificar un sonido como un sonido de evento o un flujo de sonido. Tras cargar un sonido de evento, debe llamar al método `start()` de la clase `Sound` para reproducir el sonido. Los flujos de sonido empiezan a reproducirse cuando se han cargado suficientes datos en el archivo SWF; no es necesario utilizar `start()`.

Por ejemplo, el código siguiente crea un objeto `Sound`, denominado `classical`, y después carga un archivo MP3 denominado `beethoven.mp3`:

```
var classical:Sound = new Sound();
classical.loadSound("http://server.com/mp3s/beethoven.mp3", true);
```

En la mayoría de los casos, deberá establecer en `true` el parámetro *isStreaming*, especialmente si carga grandes archivos de sonido que deben empezar a reproducirse lo antes posible, por ejemplo al crear una aplicación MP3 “jukebox”. No obstante, si descarga clips de sonido más cortos y tiene que reproducirlos en un momento concreto (por ejemplo, cuando un usuario hace clic en un botón), establezca *isStreaming* en `false`.

Para determinar si un sonido ha terminado de descargarse, utilice el controlador de eventos `Sound.onLoad`. Este controlador de eventos recibe automáticamente un valor booleano (`true` o `false`) que indica si el archivo se ha descargado correctamente.

Por ejemplo, imagine que está creando un juego en línea que utiliza diferentes sonidos en función del nivel que haya alcanzado el usuario en el juego. El código siguiente carga un archivo MP3 (`blastoff.mp3`) en un objeto `Sound`, denominado `gameSound`, y reproduce el sonido al terminar de descargarse.

```
var gameSound = new Sound();
gameSound.onLoad = function (loadedOK) {
    if(loadedOK) {
        gameSound.start();
    }
}
gameSound.loadSound("http://server.com/sounds/blastoff.mp3", false);
```

Para los archivos de sonido, Flash Player sólo admite el tipo de archivo de sonido MP3.

Para más información, consulte [Sound.loadSound\(\)](#), [Sound.start\(\)](#) y [Sound.onLoad](#) en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Lectura de etiquetas ID3 en archivos MP3

Las etiquetas ID3 son campos de datos añadidos a un archivo MP3 que contienen información sobre el mismo; por ejemplo, el título de la canción, el título del álbum y el nombre del artista.

Para leer etiquetas ID3 en un archivo MP3, utilice la propiedad `Sound.ID3`, cuyas propiedades corresponden a los nombres de las etiquetas ID3 incluidas en el archivo MP3 que se está cargando. Para determinar cuándo están disponibles las etiquetas ID3 para un archivo MP3 que se está descargando, utilice el controlador de eventos `Sound.onID3`. Flash Player7 admite las etiquetas de las versiones 1.0, 1.1, 2.3 y 2.4; las etiquetas de la versión 2.2 no son compatibles.

Por ejemplo, el código siguiente carga un archivo MP3, denominado `favoriteSong.mp3` en el objeto `Sound` denominado `song`. Cuando están disponibles las etiquetas ID3 para el archivo, aparece un campo de texto, denominado `display_txt`, que muestra el nombre del artista y el título de la canción.

```
var song = new Sound();
song.onID3 = function () {
    display_txt.text = "Artista: " + song.id3.TCOM + newline;
    display_txt.text += "Canción: " + song.id3.TIT2);
}
song.loadSound("mp3s/favoriteSong.mp3, true");
```

Dado que las etiquetas ID3 2.0 se encuentran al principio de un archivo MP3 (antes de los datos de sonido), dichas etiquetas estarán disponibles en cuanto el archivo empiece a descargarse. No obstante, las etiquetas ID3 1.0 se encuentran al final del archivo (después de los datos de sonido) y, por lo tanto, no están disponibles hasta que el archivo MP3 termina de descargarse.

Cada vez que hay disponibles nuevos datos ID3, se llama al controlador de eventos `onID3`. Esto significa que si un archivo MP3 contiene etiquetas ID3 2.0 y 1.0, se llamará dos veces al controlador `onID3`, puesto que las etiquetas se encuentran en sitios distintos del archivo.

Para obtener una lista de las etiquetas ID3 admitidas, consulte [Sound.ID3 en la página 632](#).

## Reproducción dinámica de archivos FLV externos

En lugar de importar vídeo en el entorno de edición de Flash, puede utilizar código de ActionScript para reproducir de forma dinámica archivos FLV externos en Flash Player. Los archivos FLV pueden reproducirse desde una dirección HTTP o desde el sistema de archivos local. Para reproducir archivos FLV, se utilizan las clases `NetConnection` y `NetStream` y el método `attachVideo()` de la clase `Video`. (Para más información, consulte las entradas [Clase NetConnection](#), [Clase NetStream](#) y [Video.attachVideo\(\)](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.)

Se pueden crear archivos FLV importando vídeo a la herramienta de edición de Flash y exportándolo como un archivo FLV. (Véase “Macromedia Flash Video” en el apartado Utilización de Flash de la Ayuda). Si dispone de Flash Professional, puede utilizar el plugin de exportación de FLV para exportar archivos FLV desde las aplicaciones de edición de vídeo admitidas. (Véase “Exportación de archivos FLV de aplicaciones de edición de vídeo (sólo en Flash Professional)” en el apartado Utilización de Flash de la Ayuda.)

La utilización de archivos FLV externos ofrece algunas posibilidades que no están disponibles al utilizar vídeo importado:

- Puede utilizar videoclips más largos en los documentos de Flash sin ralentizar la reproducción. Los archivos FLV externos se reproducen utilizando la *memoria caché*. Esto significa que los archivos grandes se almacenan en partes pequeñas y que se accede a ellos de forma dinámica; además, no requieren tanta memoria como los archivos de vídeo incorporados.
- Un archivo FLV externo puede tener una velocidad de fotogramas distinta a la del documento de Flash en el que se reproduce. Por ejemplo, puede establecer la velocidad de fotogramas del documento de Flash en 30 fps y la velocidad de fotogramas del vídeo en 21 fps. Esto le permite controlar mejor que la reproducción del vídeo se lleve a cabo sin problemas.
- Con los archivos FLV externos no es necesario interrumpir la reproducción de un documento de Flash mientras se carga el archivo de vídeo. A veces, los archivos de vídeo importados pueden interrumpir la reproducción de un documento para realizar ciertas funciones, como por ejemplo acceder a una unidad de CD-ROM. Los archivos FLV pueden realizar funciones independientemente del documento de Flash y, por lo tanto, no interrumpen la reproducción.
- La rotulación de contenido de vídeo es más fácil con los archivos FLV, debido a que se pueden utilizar controladores de eventos para acceder a los metadatos del vídeo.

El siguiente procedimiento muestra los pasos que se deben seguir para reproducir un archivo llamado `videoFile.flv`, almacenado en la misma ubicación que el archivo `SWF`.

**Para reproducir un archivo FLV externo en un documento de Flash:**

- 1 Con el documento abierto en la herramienta de edición de Flash, en el panel Biblioteca (Ventana > Biblioteca), seleccione Nuevo vídeo en el menú de opciones Biblioteca para crear un objeto de vídeo.
- 2 Arrastre un objeto de vídeo desde el panel Biblioteca hasta el escenario. Esto creará una instancia de objeto de vídeo.
- 3 Con el objeto de vídeo seleccionado en el escenario, introduzca `my_video` en el cuadro de texto Nombre de instancia del inspector de propiedades (Ventana > Propiedades).
- 4 Abra el panel Componentes (Ventana > Paneles de desarrollo > Componentes) y arrastre un componente `TextArea` al escenario.
- 5 Con el objeto `TextArea` seleccionado en el escenario, introduzca `status` en el cuadro de texto Nombre de instancia del inspector de propiedades.
- 6 Seleccione fotograma 1 en la línea de tiempo y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 7 Añada el código siguiente al panel Acciones:

```
// Cree un objeto NetConnection:
var netConn:NetConnection = new NetConnection();
// Cree una conexión de transmisión local:
netConn.connect(null);
// Cree un objeto NetStream y defina una función onStatus():
var netStream:NetStream = new NetStream(netConn);
netStream.onStatus = function(infoObject) {
    status.text += "Status (NetStream)" + newline;
    status.text += "Level: "+infoObject.level + newline;
    status.text += "Code: "+infoObject.code + newline;
};
// Asocie la salida de vídeo NetStream al objeto Video:
my_video.attachVideo(netStream);
// Establezca el tiempo de búfer:
netStream.setBufferTime(5);
// Reproduciendo el archivo FLV:
netStream.play("videoFile.flv");
```

## Precarga de archivos multimedia externos

ActionScript proporciona varias maneras de precargar archivos multimedia externos o hacer un seguimiento del progreso de la descarga. Para precargar archivos `SWF` y `JPEG`, utilice la clase `MovieClipLoader`, la cual proporciona un mecanismo detector de eventos para comprobar el progreso de la descarga. Esta clase es nueva en Flash Player 7. Para más información, consulte [“Precarga de archivos SWF y JPEG” en la página 207](#).

Para realizar un seguimiento del progreso de la descarga de los archivos `MP3`, use los métodos `Sound.getBytesLoaded()` y `Sound.getBytesTotal()`; para realizar un seguimiento del progreso de la descarga de los archivos `FLV`, use las propiedades `NetStream.bytesLoaded` y `NetStream.bytesTotal`. Para más información, consulte [“Precarga de archivos MP3 y FLV” en la página 208](#).

## Precarga de archivos SWF y JPEG

Para precargar archivos SWF y JPEG en instancias de clip de película, se puede utilizar la [Clase MovieClipLoader](#). Esta clase proporciona un mecanismo detector de eventos para notificar sobre el estado de las descargas de archivos en clips de película. Siga los pasos que se describen a continuación para utilizar un objeto MovieClipLoader con el fin de precargar archivos SWF y JPEG:

**Cree un nuevo objeto MovieClipLoader** Puede usar un solo objeto MovieClipLoader para realizar un seguimiento del progreso de descarga de varios archivos o para crear un objeto por separado para el progreso de cada archivo.

```
var loader:MovieClipLoader = new MovieClipLoader();
```

**Cree un objeto detector y cree controladores de eventos** El objeto detector puede ser un objeto de ActionScript cualquiera, como por ejemplo un objeto genérico Object, un clip de película o un componente personalizado.

Por ejemplo, el código siguiente crea un objeto detector genérico denominado loadListener y define para sí las funciones onLoadStart, onLoadProgress y onLoadComplete.

```
// Cree un objeto detector:
var loadListener:Object = new Object();
loadListener.onLoadStart = function (loadTarget) {
    trace("Loading into " + loadTarget + " has started.");
}
loadListener.onLoadProgress = function(loadTarget, bytesLoaded, bytesTotal) {
    var percentLoaded = bytesLoaded/bytesTotal * 100;
    trace("%" + percentLoaded + " into target " + loadTarget);
}
loadListener.onLoadComplete = function(loadTarget) {
    trace("Load completed into: " + loadTarget);
}
```

**Registre el objeto detector con el objeto MovieClipLoader** Para que el objeto detector reciba los eventos de carga, debe registrarlo con el objeto MovieClipLoader.

```
loader.addListener(loadListener);
```

**Inicie la carga del archivo (JPEG o SWF) en un clip de destino** Para iniciar la descarga del archivo JPEG o SWF, use el método MovieClipLoader.loadClip().

```
loader.loadClip("scene_2.swf");
```

**Nota:** sólo puede usar los métodos MovieClipLoader para realizar un seguimiento del progreso de descarga de los archivos cargados con el método MovieClipLoader.loadClip(). No puede usar la función loadMovie() ni el método MovieClip.loadMovie().

En el ejemplo siguiente se utiliza el método setProgress() del componente ProgressBar para mostrar el progreso de descarga de un archivo SWF. (Véase “Componente ProgressBar” en el apartado Utilización de componentes de la Ayuda.)

**Para visualizar el progreso de la descarga mediante el componente ProgressBar:**

- 1 En un documento de Flash nuevo, cree un clip de película en el escenario y denomínelo target\_mc.
- 2 Abra el panel Componentes (Ventana > Paneles de desarrollo > Componentes).
- 3 Arrastre un componente ProgressBar desde el panel Componentes al escenario.

- 4 En el inspector de propiedades, denomine pBar al componente ProgressBar y, en la ficha Parámetros, seleccione Manual en el menú emergente Modo.
- 5 Seleccione Fotograma 1 en la línea de tiempo y abra el panel Acciones (Ventana > Paneles de desarrollo > Acciones).
- 6 Añada el código siguiente al panel Acciones:

```
// Cree un objeto MovieClipLoader y un objeto detector
myLoader = new MovieClipLoader();
myListener = new Object();
// Añada las funciones callback de MovieClipLoader al objeto detector
myListener.onLoadStart = function(clip) {
    // este evento se desencadena una vez: al comenzar la carga
    pBar.label = "Now loading: " + clip;
};
myListener.onLoadProgress = function(clip, bytesLoaded, bytesTotal) {
    var percentLoaded = int (100*(bytesLoaded/bytesTotal));
    pBar.setProgress(bytesLoaded, bytesTotal);
}; myLoader.addListener(myListener);
myLoader.loadClip("veryLargeFile.swf", target_mc);
```

- 7 Para probar el documento, seleccione Control > Probar película.

Para más información, consulte la entrada [Clase MovieClipLoader](#) en el [Capítulo 12](#), “Diccionario de ActionScript”, en la página 213.

## Precarga de archivos MP3 y FLV

Para precargar archivos MP3 y FLV puede utilizar la función `setInterval()` para crear un mecanismo de sondeo que compruebe los bytes cargados para un objeto `Sound` o `NetStream` en intervalos predeterminados. Para realizar un seguimiento del progreso de la descarga de los archivos MP3, use los métodos `Sound.getBytesLoaded()` y `Sound.getBytesTotal()`; para realizar un seguimiento del progreso de la descarga de los archivos FLV, use las propiedades `NetStream.bytesLoaded` y `NetStream.bytesTotal`.

El código siguiente utiliza `setInterval()` para comprobar los bytes cargados para un objeto `Sound` o `NetStream` a intervalos predeterminados.

```
// Cree un nuevo objeto Sound para reproducir el sonido.
var songTrack = new Sound();
// Cree la función de sondeo para hacer un seguimiento del progreso de la
// descarga.
// Ésta es la función con la que se realiza el sondeo. Dicha función comprueba
// el progreso de la descarga del objeto Sound pasado como referencia.
checkProgress = function (soundObj) {
    var bytesLoaded = soundObj.getBytesLoaded();
    var bytesTotal = soundObj.getBytesTotal();
    var percentLoaded = Math.floor(bytesLoaded/bytesTotal * 100);
    trace("%" + percentLoaded + " cargado.");
}
// Cuando el archivo se termine de cargar, borre el sondeo de intervalo.
songTrack.onLoad = function () {
    clearInterval(poll);
}
// Cargue un archivo MP3 de flujo y empiece a llamar a checkProgress()
songTrack.loadSound("beethoven.mp3", true);
var poll = setInterval(checkProgress, 1000, songTrack);
```



También puede utilizar esta técnica de sondeo para precargar archivos FLV externos. Para obtener los bytes totales y el número de bytes cargados actualmente de un archivo FLV, utilice las propiedades `NetStream.bytesLoaded` y `NetStream.bytesTotal`.

Otra manera de precargar archivos FLV consiste en utilizar el método `NetStream.setBufferTime()`. Este método requiere un solo parámetro que indica el número de segundos que debe descargarse el flujo FLV antes de comenzar la reproducción.

Para más información, consulte [MovieClip.getBytesLoaded\(\)](#), [MovieClip.getBytesTotal\(\)](#), [NetStream.bytesLoaded](#), [NetStream.bytesTotal](#), [NetStream.setBufferTime\(\)](#), [setInterval\(\)](#), [Sound.getBytesLoaded\(\)](#) y [Sound.getBytesTotal\(\)](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.



# **PARTE V** Referencia

Esta sección contiene el diccionario de ActionScript, que ofrece información sobre la utilización y la sintaxis de todos los elementos del lenguaje ActionScript. También incluye apéndices que ofrecen material de referencia que se puede repasar durante la creación de scripts.

Capítulo 12: Diccionario de ActionScript . . . . .	213
Apéndice A: Mensajes de error . . . . .	783
Apéndice B: Precedencia y asociatividad de los operadores . . . . .	789
Apéndice C: Teclas del teclado y valores de códigos de tecla . . . . .	791
Apéndice D: Escritura de scripts para versiones anteriores de Flash Player . . . . .	797
Apéndice E: Programación orientada a objetos con ActionScript 1 . . . . .	801



# CAPÍTULO 12

## Diccionario de ActionScript

En este diccionario se describen la sintaxis y la utilización de los elementos de ActionScript en Macromedia Flash MX 2004 y en Macromedia Flash MX Professional 2004. Para utilizar ejemplos en un script, copie el código de ejemplo de este diccionario y péguelo en el panel Script o en un archivo de script externo. El diccionario enumera todos los elementos de ActionScript: operadores, palabras clave, declaraciones, acciones, propiedades, funciones, clases y métodos. Para ver una introducción de todas las entradas del diccionario, consulte [“Contenido del diccionario” en la página 215](#); las tablas de esta sección son un buen punto de partida para buscar acciones o métodos cuya clase no se conoce. Para más información sobre los componentes, consulte *Utilización de componentes*.

Existen dos tipos de entradas en este diccionario:

- Entradas individuales para operadores, palabras clave, funciones, variables, propiedades, métodos y sentencias.
- Entradas de clase, que proporcionan información general sobre las clases incorporadas.

Utilice esta información en las entradas de ejemplo para interpretar la estructura y las convenciones utilizadas en estos tipos de entradas.

### Entrada de muestra para la mayoría de los elementos de ActionScript

En la siguiente entrada de muestra del diccionario se explican las convenciones utilizadas para todos los elementos de ActionScript que no son clases.

#### Título de la entrada

Todas las entradas aparecen en la lista en orden alfabético. El orden alfabético no distingue entre mayúsculas y minúsculas, pasa por alto los signos de subrayado iniciales, etc.

#### Disponibilidad

A no ser que se indique lo contrario, en la sección Disponibilidad se especifica qué versiones de Flash Player admiten el elemento. No se trata de la versión de Flash utilizada para editar el contenido. Por ejemplo, si utiliza Macromedia Flash MX 2004 o Macromedia Flash MX Professional 2004 para crear contenido para Flash Player 6, sólo puede utilizar elementos de ActionScript que estén disponibles para Flash Player 6.

En algunos casos, en esta sección también se indica qué versión de la herramienta de edición admite un elemento. Para ver un ejemplo, consulte [System.setClipboard\(\)](#).

Finalmente, si un elemento sólo se admite en ActionScript 2.0, esa información se anota también en esta sección.

### Sintaxis

En esta sección se proporciona la sintaxis correcta para utilizar el elemento de ActionScript en el código. La parte necesaria de la sintaxis se muestra con *fuentes para código* y el código que debe proporcionar el usuario se muestra con *fuentes para código en cursiva*. Los corchetes ([]) indican parámetros opcionales.

### Parámetros

En esta sección se describen los parámetros listados en la sintaxis.

### Valor devuelto

En esta sección se identifican los valores que el elemento devuelve, si los hay.

### Descripción

En esta sección se identifica el tipo de elemento (por ejemplo, operador, método, función, etc.) y, a continuación, se describe cómo utilizarlo.

### Ejemplo

En esta sección aparece un ejemplo de código que muestra cómo utilizar el elemento.

### Véase también

En esta sección se muestra una lista de las entradas del diccionario de ActionScript relacionadas.

## Entrada de muestra para clases

En la siguiente entrada de muestra del diccionario se explican las convenciones utilizadas para las clases de ActionScript incorporadas. Las clases se enumeran alfabéticamente junto con los demás elementos del diccionario.

### Título de la entrada

El título de la entrada indica el nombre de la clase. El nombre de la clase va seguido de información descriptiva general.

### Tablas de resumen sobre métodos y propiedades

Cada entrada de clase contiene una tabla en la que se enumeran todos los métodos asociados. Si la clase tiene propiedades (a menudo constantes), controladores de eventos o detectores de eventos, estos elementos se resumen en tablas adicionales. Todos los elementos que aparecen enumerados en estas tablas también tienen sus propias entradas en el diccionario, que siguen a la entrada de clase.

## Constructor

Si una clase requiere el uso de un constructor para poder acceder a sus métodos y propiedades, el constructor se describe en la entrada de cada clase. Esta descripción tiene todos los elementos estándar (sintaxis, descripción, etc.) de las otras entradas del diccionario.

## Listas de métodos y propiedades

Los métodos y las propiedades de una clase se enumeran alfabéticamente después de la entrada de la clase.

## Contenido del diccionario

Todas las entradas del diccionario aparecen en la lista en orden alfabético. Sin embargo, algunos operadores son símbolos y se presentan en orden ASCII. Además, los métodos asociados con una clase se enumeran junto con el nombre de la clase; por ejemplo, el método `abs()` de la clase `Math` se enumera como `Math.abs()`.

Las dos tablas siguientes le ayudan a localizar estos elementos. En la primera tabla aparece una lista de los operadores simbólicos en el orden en el que aparecen en el diccionario. En la segunda tabla aparecen todos los demás elementos de ActionScript.

Operadores simbólicos	Véase la entrada
--	-- (decremento)
++	++ (incremento)
!	! (NOT lógico)
!=	!= (desigualdad)
!==	!== (desigualdad estricta)
%	% (módulo)
%=	%= (asignación de módulo)
&	& (operador AND en modo bit)
&&	&& (AND lógico)
&=	&= (asignación AND en modo bit)
()	() (paréntesis)
-	- (menos)
*	* (multiplicación)
*=	*= (asignación de multiplicación)
,	, (coma)
.	. (punto)
:	: (tipo)
?:	?: (condicional)
/	/ (división)
//	// (delimitador de comentario)

Operadores simbólicos	Véase la entrada
<code>/*</code>	<code>/*</code> (delimitador de comentario)
<code>/=</code>	<code>/=</code> (asignación de división)
<code>[]</code>	<code>[]</code> (acceso a matriz)
<code>^</code>	<code>^</code> (XOR en modo bit)
<code>^=</code>	<code>^=</code> (asignación XOR en modo bit)
<code>{}</code>	<code>{}</code> (inicializador de objeto)
<code> </code>	<code> </code> (OR en modo bit)
<code>  </code>	<code>  </code> (OR lógico)
<code> =</code>	<code> =</code> (asignación OR en modo bit)
<code>~</code>	<code>~</code> (NOT en modo bit)
<code>+</code>	<code>+</code> (suma)
<code>+=</code>	<code>+=</code> (asignación de suma)
<code>&lt;</code>	<code>&lt;</code> (menor que)
<code>&lt;&lt;</code>	<code>&lt;&lt;</code> (desplazamiento a la izquierda en modo bit)
<code>&lt;&lt;=</code>	<code>&lt;&lt;=</code> (desplazamiento a la izquierda en modo bit y asignación)
<code>&lt;=</code>	<code>&lt;=</code> (menor o igual que)
<code>&lt;&gt;</code>	<code>&lt;&gt;</code> (desigualdad)
<code>=</code>	<code>=</code> (asignación)
<code>-=</code>	<code>-=</code> (asignación de resta)
<code>==</code>	<code>==</code> (igualdad)
<code>===</code>	<code>===</code> (igualdad estricta)
<code>&gt;</code>	<code>&gt;</code> (mayor que)
<code>&gt;=</code>	<code>&gt;=</code> (mayor o igual que)
<code>&gt;&gt;</code>	<code>&gt;&gt;</code> (desplazamiento a la derecha en modo bit)
<code>&gt;&gt;=</code>	<code>&gt;&gt;=</code> (desplazamiento a la derecha en modo bit y asignación)
<code>&gt;&gt;&gt;</code>	<code>&gt;&gt;&gt;</code> (desplazamiento a la derecha en modo bit sin signo)
<code>&gt;&gt;&gt;=</code>	<code>&gt;&gt;&gt;=</code> (desplazamiento a la derecha en modo bit sin signo y asignación)



En la tabla siguiente se muestra una lista de todos los elementos de ActionScript que no son operadores simbólicos.

Elemento de ActionScript	Véase la entrada
#endinitclip	<a href="#">#endinitclip</a>
#include	<a href="#">#include</a>
#initclip	<a href="#">#initclip</a>
__proto__	<a href="#">Object.__proto__</a>
_accProps	<a href="#">_accProps</a>
_alpha	<a href="#">MovieClip._alpha</a> , <a href="#">Button._alpha</a> , <a href="#">TextField._alpha</a>
_currentframe	<a href="#">MovieClip._currentframe</a>
_droptarget	<a href="#">MovieClip._droptarget</a>
_focusrect	<a href="#">_focusrect</a> , <a href="#">Button._focusrect</a> , <a href="#">MovieClip._focusrect</a>
_framesloaded	<a href="#">MovieClip._framesloaded</a>
_global	<a href="#">_global</a> <a href="#">object</a>
_height	<a href="#">Button._height</a> , <a href="#">MovieClip._height</a> , <a href="#">TextField._height</a>
_highquality	<a href="#">_highquality</a> , <a href="#">Button._highquality</a> , <a href="#">MovieClip._highquality</a> , <a href="#">TextField._highquality</a>
_lockroot	<a href="#">MovieClip._lockroot</a>
_name	<a href="#">Button._name</a> , <a href="#">MovieClip._name</a> , <a href="#">TextField._name</a>
_parent	<a href="#">_parent</a> , <a href="#">Button._parent</a> , <a href="#">MovieClip._parent</a> , <a href="#">TextField._parent</a>
_quality	<a href="#">_quality</a> , <a href="#">Button._quality</a> , <a href="#">TextField._quality</a>
_root	<a href="#">_root</a>
_rotation	<a href="#">Button._rotation</a> , <a href="#">MovieClip._rotation</a> , <a href="#">TextField._rotation</a>
_soundbuftime	<a href="#">_soundbuftime</a> , <a href="#">Button._soundbuftime</a> , <a href="#">MovieClip._soundbuftime</a> , <a href="#">TextField._soundbuftime</a>
_target	<a href="#">Button._target</a> , <a href="#">MovieClip._target</a> , <a href="#">TextField._target</a>
_totalframes	<a href="#">MovieClip._totalframes</a>
_url	<a href="#">Button._url</a> , <a href="#">MovieClip._url</a> , <a href="#">TextField._url</a>
_visible	<a href="#">Button._visible</a> , <a href="#">MovieClip._visible</a> , <a href="#">TextField._visible</a>
_width	<a href="#">Button._width</a> , <a href="#">MovieClip._width</a> , <a href="#">TextField._width</a>
_x	<a href="#">Button._x</a> , <a href="#">MovieClip._x</a> , <a href="#">TextField._x</a>
_xmouse	<a href="#">Button._xmouse</a> , <a href="#">MovieClip._xmouse</a> , <a href="#">TextField._xmouse</a>
_xscale	<a href="#">Button._xscale</a> , <a href="#">MovieClip._xscale</a> , <a href="#">TextField._xscale</a>
_y	<a href="#">Button._y</a> , <a href="#">MovieClip._y</a> , <a href="#">TextField._y</a>
_ymouse	<a href="#">Button._ymouse</a> , <a href="#">MovieClip._ymouse</a> , <a href="#">TextField._ymouse</a>

Elemento de ActionScript	Véase la entrada
<code>_yscale</code>	<a href="#">Button._yscale</a> , <a href="#">MovieClip._yscale</a> , <a href="#">TextField._yscale</a>
<code>abs</code>	<a href="#">Math.abs()</a>
<code>Accessibility</code>	<a href="#">Clase Accessibility</a>
<code>acos</code>	<a href="#">Math.acos()</a>
<code>activityLevel</code>	<a href="#">Camera.activityLevel</a> , <a href="#">Microphone.activityLevel</a>
<code>add</code>	<a href="#">add</a>
<code>addListener</code>	<a href="#">Key.addListener()</a> , <a href="#">Mouse.addListener()</a> , <a href="#">MovieClipLoader.addListener()</a> , <a href="#">Selection.addListener()</a> , <a href="#">Stage.addListener()</a> , <a href="#">TextField.addListener()</a>
<code>addPage</code>	<a href="#">PrintJob.addPage()</a>
<code>addProperty</code>	<a href="#">Object.addProperty()</a>
<code>addRequestHeader</code>	<a href="#">LoadVars.addRequestHeader()</a> , <a href="#">XML.addRequestHeader()</a>
<code>align</code>	<a href="#">Stage.align</a> , <a href="#">TextFormat.align</a>
<code>allowDomain</code>	<a href="#">LocalConnection.allowDomain</a> , <a href="#">System.security.allowDomain()</a>
<code>allowInsecureDomain</code>	<a href="#">LocalConnection.allowInsecureDomain()</a> , <a href="#">System.security.allowInsecureDomain()</a>
<code>and</code>	<a href="#">and</a>
<code>appendChild</code>	<a href="#">XML.appendChild()</a>
<code>apply</code>	<a href="#">Function.apply()</a>
<code>Arguments</code>	<a href="#">Clase Arguments</a>
<code>Array</code>	<a href="#">Clase Array</a> , <a href="#">Array()</a>
<code>asfunction</code>	<a href="#">asfunction</a>
<code>asin</code>	<a href="#">Math.asin()</a>
<code>atan</code>	<a href="#">Math.atan()</a>
<code>atan2</code>	<a href="#">Math.atan2()</a>
<code>attachAudio</code>	<a href="#">MovieClip.attachAudio()</a>
<code>attachMovie</code>	<a href="#">MovieClip.attachMovie()</a>
<code>attachSound</code>	<a href="#">Sound.attachSound()</a>
<code>attachVideo</code>	<a href="#">Video.attachVideo()</a>
<code>attributes</code>	<a href="#">XML.attributes</a>
<code>autosize</code>	<a href="#">TextField.autoSize</a>
<code>avHardwareDisable</code>	<a href="#">System.capabilities.avHardwareDisable</a>
<code>background</code>	<a href="#">TextField.background</a>
<code>backgroundColor</code>	<a href="#">TextField.backgroundColor</a>
<code>BACKSPACE</code>	<a href="#">Key.BACKSPACE</a>

Elemento de ActionScript	Véase la entrada
bandwidth	<a href="#">Camera.bandwidth</a>
beginFill	<a href="#">MovieClip.beginFill()</a>
beginGradientFill	<a href="#">MovieClip.beginGradientFill()</a>
blockIndent	<a href="#">TextFormat.blockIndent</a>
bold	<a href="#">TextFormat.bold</a>
Boolean	<a href="#">Boolean()</a> , <a href="#">Clase Boolean</a>
border	<a href="#">TextField.border</a>
borderColor	<a href="#">TextField.borderColor</a>
bottomScroll	<a href="#">TextField.bottomScroll</a>
break	<a href="#">break</a>
bufferLength	<a href="#">NetStream.bufferLength</a>
bufferTime	<a href="#">NetStream.bufferTime</a>
builtInItems	<a href="#">ContextMenu.builtInItems</a>
bullet	<a href="#">TextFormat.bullet</a>
Button	<a href="#">Clase Button</a>
bytesLoaded	<a href="#">NetStream.bytesLoaded</a>
bytesTotal	<a href="#">NetStream.bytesTotal</a>
call	<a href="#">call()</a> , <a href="#">Function.call()</a>
callee	<a href="#">arguments.callee</a>
caller	<a href="#">arguments.caller</a>
Camera	<a href="#">Clase Camera</a>
capabilities	<a href="#">Objeto System.capabilities</a>
CAPSLOCK	<a href="#">Key.CAPSLOCK</a>
caption	<a href="#">ContextMenuItem.caption</a>
case	<a href="#">case</a>
catch	<a href="#">try..catch..finally</a>
ceil	<a href="#">Math.ceil()</a>
charAt	<a href="#">String.charAt()</a>
charCodeAt	<a href="#">String.charCodeAt()</a>
childNodes	<a href="#">XML.childNodes</a>
chr	<a href="#">chr</a>
class	<a href="#">class</a>
clear	<a href="#">MovieClip.clear()</a> , <a href="#">SharedObject.clear()</a> , <a href="#">Video.clear()</a>
clearInterval	<a href="#">clearInterval()</a>

Elemento de ActionScript	Véase la entrada
cloneNode	<a href="#">XML.cloneNode()</a>
close	<a href="#">LocalConnection.close()</a> , <a href="#">NetStream.close()</a> , <a href="#">XMLSocket.close()</a>
Color	<a href="#">Clase Color</a> , <a href="#">TextFormat.color</a>
concat	<a href="#">Array.concat()</a> , <a href="#">String.concat()</a>
connect	<a href="#">LocalConnection.connect()</a> , <a href="#">NetConnection.connect()</a> , <a href="#">XMLSocket.connect()</a>
condenseWhite	<a href="#">TextField.condenseWhite</a>
constructor	<a href="#">Clase Array</a> , <a href="#">Clase Boolean</a> , <a href="#">Clase Camera</a> , <a href="#">Clase Color</a> , <a href="#">Clase ContextMenu</a> , <a href="#">Clase ContextMenuItem</a> , <a href="#">Clase Date</a> , <a href="#">Clase Error</a> , <a href="#">Clase LoadVars</a> , <a href="#">Clase LocalConnection</a> , <a href="#">Clase Microphone</a> , <a href="#">Clase NetConnection</a> , <a href="#">Clase NetStream</a> , <a href="#">Clase Number</a> , <a href="#">Clase Object</a> , <a href="#">Clase PrintJob</a> , <a href="#">Clase SharedObject</a> , <a href="#">Clase Sound</a> , <a href="#">Clase String</a> , <a href="#">Clase TextField.StyleSheet</a> , <a href="#">Clase TextFormat</a> , <a href="#">Clase XML</a> , <a href="#">Clase XMLSocket</a>
contentType	<a href="#">LoadVars.contentType</a> , <a href="#">XML.contentType</a>
ContextMenu	<a href="#">Clase ContextMenu</a>
ContextMenuitem	<a href="#">Clase ContextMenuItem</a>
continue	<a href="#">continue</a>
CONTROL	<a href="#">Key.CONTROL</a>
copy	<a href="#">ContextMenu.copy()</a> , <a href="#">ContextMenuitem.copy()</a>
cos	<a href="#">Math.cos()</a>
createElement	<a href="#">XML.createElement()</a>
createEmptyMovieClip	<a href="#">MovieClip.createEmptyMovieClip()</a>
createTextField	<a href="#">MovieClip.createTextField()</a>
createTextNode	<a href="#">XML.createTextNode()</a>
currentFps	<a href="#">Camera.currentFps</a> , <a href="#">NetStream.currentFps</a>
curveTo	<a href="#">MovieClip.curveTo()</a>
CustomActions	<a href="#">Clase CustomActions</a>
customItems	<a href="#">ContextMenu.customItems</a>
data	<a href="#">SharedObject.data</a>
Date	<a href="#">Clase Date</a>
deblocking	<a href="#">Video.deblocking</a>
default	<a href="#">default</a>
delete	<a href="#">delete</a>
DELETEKEY	<a href="#">Key.DELETEKEY</a>
do while	<a href="#">do while</a>

Elemento de ActionScript	Véase la entrada
docTypeDecl	<a href="#">XML.docTypeDecl</a>
domain	<a href="#">LocalConnection.domain()</a>
DOWN	<a href="#">Key.DOWN</a>
duplicateMovieClip	<a href="#">duplicateMovieClip()</a> , <a href="#">MovieClip.duplicateMovieClip()</a>
duration	<a href="#">Sound.duration</a>
dynamic	<a href="#">dynamic</a>
E	<a href="#">Math.E</a>
else	<a href="#">else</a>
else if	<a href="#">else if</a>
embedFonts	<a href="#">TextField.embedFonts</a>
enabled	<a href="#">Button.enabled</a> , <a href="#">ContextMenuitem.enabled</a> , <a href="#">MovieClip.enabled</a>
END	<a href="#">Key.END</a>
endFill	<a href="#">MovieClip.endFill()</a>
ENTER	<a href="#">Key.ENTER</a>
eq	<a href="#">eq</a> (igual; específico para cadenas)
Error	<a href="#">Clase Error</a>
ESCAPE (constante)	<a href="#">Key.ESCAPE</a>
escape (función)	<a href="#">escape</a>
eval	<a href="#">eval()</a>
exactSettings	<a href="#">System.exactSettings</a>
exp	<a href="#">Math.exp()</a>
extends	<a href="#">extends</a>
false	<a href="#">false</a>
finally	<a href="#">try..catch..finally</a>
findText	<a href="#">TextSnapshot.findText()</a>
firstChild	<a href="#">XML.firstChild</a>
floor	<a href="#">Math.floor()</a>
flush	<a href="#">SharedObject.flush()</a>
focusEnabled	<a href="#">MovieClip.focusEnabled</a>
font	<a href="#">TextFormat.font</a>
for	<a href="#">for</a>
for..in	<a href="#">for..in</a>
fps	<a href="#">Camera.fps</a>
fromCharCode	<a href="#">String.fromCharCode()</a>
fscommand	<a href="#">fscommand()</a>

Elemento de ActionScript	Véase la entrada
function	<a href="#">function</a> , <a href="#">Clase Function</a>
gain	<a href="#">Microphone.gain</a>
ge	<a href="#">ge</a> (mayor o igual que; específico para cadenas)
get	<a href="#">Camera.get()</a> , <a href="#">CustomActions.get()</a> , <a href="#">get</a> , <a href="#">Microphone.get()</a>
getAscii	<a href="#">Key.getAscii()</a>
getBeginIndex	<a href="#">Selection.getBeginIndex()</a>
getBounds	<a href="#">MovieClip.getBounds()</a>
getBytesLoaded	<a href="#">LoadVars.getBytesLoaded()</a> , <a href="#">MovieClip.getBytesLoaded()</a> , <a href="#">Sound.getBytesLoaded()</a> , <a href="#">XML.getBytesLoaded()</a>
getBytesTotal	<a href="#">LoadVars.getBytesTotal()</a> , <a href="#">MovieClip.getBytesTotal()</a> , <a href="#">Sound.getBytesTotal()</a> , <a href="#">XML.getBytesTotal()</a>
getCaretIndex	<a href="#">Selection.getCaretIndex()</a>
getCode	<a href="#">Key.getCode()</a>
getCount	<a href="#">TextSnapshot.getCount()</a>
getDate	<a href="#">Date.getDate()</a>
getDay	<a href="#">Date.getDay()</a>
getDepth	<a href="#">Button.getDepth()</a> , <a href="#">MovieClip.getDepth()</a> , <a href="#">TextField.getDepth()</a>
getEndIndex	<a href="#">Selection.getEndIndex()</a>
getFocus	<a href="#">Selection.getFocus()</a>
getFontList	<a href="#">TextField.getFontList()</a>
getFullYear	<a href="#">Date.getFullYear()</a>
getHours	<a href="#">Date.getHours()</a>
getInstanceAtDepth	<a href="#">MovieClip.getInstanceAtDepth()</a>
getLocal	<a href="#">SharedObject.getLocal()</a>
getMilliseconds	<a href="#">Date.getMilliseconds()</a>
getMinutes	<a href="#">Date.getMinutes()</a>
getMonth	<a href="#">Date.getMonth()</a>
getNewTextFormat	<a href="#">TextField.getNewTextFormat()</a>
getNextHighestDepth	<a href="#">MovieClip.getNextHighestDepth()</a>
getPan	<a href="#">Sound.getPan()</a>
getProgress	<a href="#">MovieClipLoader.getProgress()</a>
getProperty	<a href="#">getProperty</a>
getRGB	<a href="#">Color.getRGB()</a>
getSeconds	<a href="#">Date.getSeconds()</a>
getSelected	<a href="#">TextSnapshot.getSelected()</a>

Elemento de ActionScript	Véase la entrada
getSelectedText	<a href="#">TextSnapshot.getSelectedText()</a>
getSize	<a href="#">SharedObject.getSize()</a>
getStyle	<a href="#">TextField.StyleSheet.getStyle()</a>
getStyleNames	<a href="#">TextField.StyleSheet.getStyleNames()</a>
getSWFVersion	<a href="#">MovieClip.getSWFVersion()</a>
getText	<a href="#">TextSnapshot.getText()</a>
getTextExtent	<a href="#">TextFormat.getTextExtent()</a>
getTextFormat	<a href="#">TextField.getTextFormat()</a>
getTextSnapshot	<a href="#">MovieClip.getTextSnapshot()</a>
getTime	<a href="#">Date.getTime()</a>
getTimer	<a href="#">getTimer</a>
getTimezoneOffset	<a href="#">Date.getTimezoneOffset()</a>
getTransform	<a href="#">Color.getTransform()</a> , <a href="#">Sound.getTransform()</a>
getURL	<a href="#">getURL()</a> , <a href="#">MovieClip.getURL()</a>
getUTCDate	<a href="#">Date.getUTCDate()</a>
getUTCDay	<a href="#">Date.getUTCDay()</a>
getUTCFullYear	<a href="#">Date.getUTCFullYear()</a>
getUTCHours	<a href="#">Date.getUTCHours()</a>
getUTCMilliseconds	<a href="#">Date.getUTCMilliseconds()</a>
getUTCMinutes	<a href="#">Date.getUTCMinutes()</a>
getUTCMonth	<a href="#">Date.getUTCMonth()</a>
getUTCSeconds	<a href="#">Date.getUTCSeconds()</a>
getVersion	<a href="#">getVersion</a>
getVolume	<a href="#">Sound.getVolume()</a>
getYear	<a href="#">Date.getYear()</a>
globalToLocal	<a href="#">MovieClip.globalToLocal()</a>
goto	<a href="#">gotoAndPlay()</a> , <a href="#">gotoAndStop()</a>
gotoAndPlay	<a href="#">gotoAndPlay()</a> , <a href="#">MovieClip.gotoAndPlay()</a>
gotoAndStop	<a href="#">gotoAndStop()</a> , <a href="#">MovieClip.gotoAndStop()</a>
gt	<a href="#">gt</a> (mayor que; específico para cadenas)
hasAccessibility	<a href="#">System.capabilities.hasAccessibility</a>
hasAudio	<a href="#">System.capabilities.hasAudio</a>
hasAudioEncoder	<a href="#">System.capabilities.hasAudioEncoder</a>
hasChildNodes	<a href="#">XML.hasChildNodes()</a>

Elemento de ActionScript	Véase la entrada
hasEmbeddedVideo	<a href="#">System.capabilities.hasEmbeddedVideo</a>
hasMP3	<a href="#">System.capabilities.hasMP3</a>
hasPrinting	<a href="#">System.capabilities.hasPrinting</a>
hasScreenBroadcast	<a href="#">System.capabilities.hasScreenBroadcast</a>
hasScreenPlayback	<a href="#">System.capabilities.hasScreenPlayback</a>
hasStreamingAudio	<a href="#">System.capabilities.hasStreamingAudio</a>
hasStreamingVideo	<a href="#">System.capabilities.hasStreamingVideo</a>
hasVideoEncoder	<a href="#">System.capabilities.hasVideoEncoder</a>
height	<a href="#">Camera.height</a> , <a href="#">Stage.height</a> , <a href="#">Video.height</a>
hide	<a href="#">Mouse.hide()</a>
hideBuiltInItems	<a href="#">ContextMenu.hideBuiltInItems()</a>
hitArea	<a href="#">MovieClip.hitArea</a>
hitTest	<a href="#">MovieClip.hitTest()</a>
hitTestTextNearPos	<a href="#">TextSnapshot.hitTestTextNearPos()</a>
HOME	<a href="#">Key.HOME</a>
hscroll	<a href="#">TextField.hscroll</a>
html	<a href="#">TextField.html</a>
htmlText	<a href="#">TextField.htmlText</a>
ID3	<a href="#">Sound.ID3</a>
if	<a href="#">if</a>
ifFrameLoaded	<a href="#">ifFrameLoaded</a>
ignoreWhite	<a href="#">XML.ignoreWhite</a>
implements	<a href="#">implements</a>
import	<a href="#">import</a>
indent	<a href="#">TextFormat.indent</a>
index	<a href="#">Camera.index</a> , <a href="#">Microphone.index</a>
indexOf	<a href="#">String.indexOf()</a>
Infinity	<a href="#">Infinity</a>
-Infinity	<a href="#">-Infinity</a>
INSERT	<a href="#">Key.INSERT</a>
insertBefore	<a href="#">XML.insertBefore()</a>
install	<a href="#">CustomActions.install()</a>
instanceof	<a href="#">instanceof</a>
int	<a href="#">int</a>
interface	<a href="#">interface</a>



Elemento de ActionScript	Véase la entrada
isActive	<a href="#">Accessibility.isActive()</a>
isDebugger	<a href="#">System.capabilities.isDebugger</a>
isDown	<a href="#">Key.isDown()</a>
isFinite	<a href="#">isFinite</a>
isNaN	<a href="#">isNaN()</a>
isToggled	<a href="#">Key.isToggled()</a>
italic	<a href="#">TextFormat.italic</a>
join	<a href="#">Array.join()</a>
Key	<a href="#">Clase Key</a>
language	<a href="#">System.capabilities.language</a>
lastChild	<a href="#">XML.lastChild</a>
lastIndexOf	<a href="#">String.lastIndexOf()</a>
le	<a href="#">le</a> (menor o igual que; específico para cadenas)
leading	<a href="#">TextFormat.leading</a>
LEFT	<a href="#">Key.LEFT</a>
leftMargin	<a href="#">TextFormat.leftMargin</a>
length	<a href="#">length</a> , <a href="#">arguments.length</a> , <a href="#">Array.length</a> , <a href="#">String.length</a> , <a href="#">TextField.length</a>
level	<a href="#">_level</a>
lineStyle	<a href="#">MovieClip.lineStyle()</a>
lineTo	<a href="#">MovieClip.lineTo()</a>
list	<a href="#">CustomActions.list()</a>
LN10	<a href="#">Math.LN10</a>
LN2	<a href="#">Math.LN2</a>
load	<a href="#">LoadVars.load()</a> , <a href="#">TextField.StyleSheet.load()</a> , <a href="#">XML.load()</a> ,
loadClip	<a href="#">MovieClipLoader.loadClip()</a>
loaded	<a href="#">LoadVars.loaded</a> , <a href="#">XML.loaded</a>
loadMovie	<a href="#">loadMovie()</a> , <a href="#">MovieClip.loadMovie()</a>
loadMovieNum	<a href="#">loadMovieNum()</a>
loadSound	<a href="#">Sound.loadSound()</a>
loadVariables	<a href="#">loadVariables()</a> , <a href="#">MovieClip.loadVariables()</a>
loadVariablesNum	<a href="#">loadVariablesNum()</a>
LoadVars	<a href="#">Clase LoadVars</a>
LocalConnection	<a href="#">Clase LocalConnection</a>
localFileReadDisable	<a href="#">System.capabilities.localFileReadDisable</a>

Elemento de ActionScript	Véase la entrada
localToGlobal	<a href="#">MovieClip.localToGlobal()</a>
log	<a href="#">Math.log()</a>
LOG10E	<a href="#">Math.LOG10E</a>
LOG2E	<a href="#">Math.LOG2E</a>
lt	<a href="#">lt</a> (menor que; específico para cadenas)
manufacturer	<a href="#">System.capabilities.manufacturer</a>
Math	<a href="#">Clase Math</a>
max	<a href="#">Math.max()</a>
MAX_VALUE	<a href="#">Number.MAX_VALUE</a>
maxChars	<a href="#">TextField.maxChars</a>
maxhscroll	<a href="#">TextField.maxhscroll</a>
maxscroll	<a href="#">maxscroll</a> , <a href="#">TextField.maxscroll</a>
mbchr	<a href="#">mbchr</a>
mblength	<a href="#">mblength</a>
mbord	<a href="#">mbord</a>
mbsubstring	<a href="#">mbsubstring</a>
menu	<a href="#">Button.menu</a> , <a href="#">MovieClip.menu</a> , <a href="#">TextField.menu</a>
message	<a href="#">Error.message</a>
Microphone	<a href="#">Clase Microphone</a>
min	<a href="#">Math.min()</a>
MIN_VALUE	<a href="#">Number.MIN_VALUE</a>
MMExecute	<a href="#">MMExecute()</a>
motionLevel	<a href="#">Camera.motionLevel</a>
motionTimeOut	<a href="#">Camera.motionTimeOut</a>
Mouse	<a href="#">Clase Mouse</a>
mouseWheelEnabled	<a href="#">TextField.mouseWheelEnabled</a>
moveTo	<a href="#">MovieClip.moveTo()</a>
MovieClip	<a href="#">Clase MovieClip</a>
MovieClipLoader	<a href="#">Clase MovieClipLoader</a>
multiline	<a href="#">TextField.multiline</a>
muted	<a href="#">Camera.muted</a> , <a href="#">Microphone.muted</a>
name	<a href="#">Error.name</a> , <a href="#">Microphone.name</a>
names	<a href="#">Camera.names</a> , <a href="#">Microphone.names</a>
NaN	<a href="#">NaN</a> , <a href="#">Number.NaN</a>

Elemento de ActionScript	Véase la entrada
ne	ne (no igual; específico para cadenas)
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
NetConnection	<a href="#">Clase NetConnection</a>
NetStream	<a href="#">Clase NetStream</a>
new (operador)	new
newline	newline
nextFrame	nextFrame(), MovieClip.nextFrame()
nextScene	nextScene()
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
not	not
null	null
Number	Number(), <a href="#">Clase Number</a>
Object	<a href="#">Clase Object</a> , Object()
on	on()
onActivity	Camera.onActivity, Microphone.onActivity
onChanged	TextField.onChanged
onClipEvent	onClipEvent()
onClose	XMLSocket.onClose()
onConnect	XMLSocket.onConnect()
onData	LoadVars.onData, MovieClip.onData, XML.onData, XMLSocket.onData()
onDragOut	Button.onDragOut, MovieClip.onDragOut
onDragOver	Button.onDragOver, MovieClip.onDragOver
onEnterFrame	MovieClip.onEnterFrame
onID3	Sound.onID3
onKeyDown	Button.onKeyDown, Key.onKeyDown, MovieClip.onKeyDown
onKeyUp	Button.onKeyUp, Key.onKeyUp, MovieClip.onKeyUp
onKillFocus	Button.onKillFocus, MovieClip.onKillFocus, TextField.onKillFocus
onLoad	LoadVars.onLoad, MovieClip.onLoad, Sound.onLoad, TextField.StyleSheet.onLoad, XML.onLoad()
onLoadComplete	MovieClipLoader.onLoadComplete()

Elemento de ActionScript	Véase la entrada
onLoadError	<a href="#">MovieClipLoader.onLoadError()</a>
onLoadInit	<a href="#">MovieClipLoader.onLoadInit()</a>
onLoadProgress	<a href="#">MovieClipLoader.onLoadProgress()</a>
onLoadStart	<a href="#">MovieClipLoader.onLoadStart()</a>
onMouseDown	<a href="#">Mouse.onMouseDown</a> , <a href="#">MovieClip.onMouseDown</a>
onMouseMove	<a href="#">Mouse.onMouseMove</a> , <a href="#">MovieClip.onMouseMove</a>
onMouseUp	<a href="#">Mouse.onMouseUp</a> , <a href="#">MovieClip.onMouseUp</a>
onMouseWheel	<a href="#">Mouse.onMouseWheel</a>
onPress	<a href="#">Button.onPress</a> , <a href="#">MovieClip.onPress</a>
onRelease	<a href="#">Button.onRelease</a> , <a href="#">MovieClip.onRelease</a>
onReleaseOutside	<a href="#">Button.onReleaseOutside</a> , <a href="#">MovieClip.onReleaseOutside</a>
onResize	<a href="#">Stage.onResize</a>
onRollOut	<a href="#">Button.onRollOut</a> , <a href="#">MovieClip.onRollOut</a>
onRollOver	<a href="#">Button.onRollOver</a> , <a href="#">MovieClip.onRollOver</a>
onScroller	<a href="#">TextField.onScroller</a>
onSelect	<a href="#">ContextMenu.onSelect</a> , <a href="#">ContextMenuItem.onSelect</a>
onSetFocus	<a href="#">Button.onSetFocus</a> , <a href="#">MovieClip.onSetFocus</a> , <a href="#">Selection.onSetFocus</a> , <a href="#">TextField.onSetFocus</a>
onSoundComplete	<a href="#">Sound.onSoundComplete</a>
onStatus	<a href="#">Camera.onStatus</a> , <a href="#">LocalConnection.onStatus</a> , <a href="#">Microphone.onStatus</a> , <a href="#">NetStream.onStatus</a> , <a href="#">SharedObject.onStatus</a> , <a href="#">System.onStatus</a>
onUnload	<a href="#">MovieClip.onUnload</a>
onUpdate	<a href="#">onUpdate</a>
onXML	<a href="#">XMLSocket.onXML()</a>
or (OR lógico)	<a href="#">or</a>
ord	<a href="#">ord</a>
os	<a href="#">System.capabilities.os</a>
parentNode	<a href="#">XML.parentNode</a>
parseCSS	<a href="#">TextField.StyleSheet.parseCSS()</a>
parseFloat	<a href="#">parseFloat()</a>
parseInt	<a href="#">parseInt</a>
parseXML	<a href="#">XML.parseXML()</a>
password	<a href="#">TextField.password</a>
pause	<a href="#">NetStream.pause()</a>

Elemento de ActionScript	Véase la entrada
PGDN	<a href="#">Key.PGDN</a>
PGUP	<a href="#">Key.PGUP</a>
PI	<a href="#">Math.PI</a>
pixelAspectRatio	<a href="#">System.capabilities.pixelAspectRatio</a>
play	<a href="#">play()</a> , <a href="#">MovieClip.play()</a> , <a href="#">NetStream.play()</a>
playerType	<a href="#">System.capabilities.playerType</a>
pop	<a href="#">Array.pop()</a>
position	<a href="#">Sound.position</a>
POSITIVE_INFINITY	<a href="#">Number.POSITIVE_INFINITY</a>
pow	<a href="#">Math.pow()</a>
prevFrame	<a href="#">prevFrame()</a> , <a href="#">MovieClip.prevFrame()</a>
previousSibling	<a href="#">XML.previousSibling</a>
prevScene	<a href="#">prevScene()</a>
print	<a href="#">print()</a>
printAsBitmap	<a href="#">printAsBitmap()</a>
printAsBitmapNum	<a href="#">printAsBitmapNum()</a>
PrintJob	<a href="#">Clase PrintJob</a>
printNum	<a href="#">printNum()</a>
private	<a href="#">private</a>
prototype	<a href="#">Function.prototype</a>
public	<a href="#">public</a>
push	<a href="#">Array.push()</a>
quality	<a href="#">Camera.quality</a>
random	<a href="#">random</a> , <a href="#">Math.random()</a>
rate	<a href="#">Microphone.rate</a>
registerClass	<a href="#">Object.registerClass()</a>
removeListener	<a href="#">Key.removeListener()</a> , <a href="#">Mouse.removeListener()</a> , <a href="#">MovieClipLoader.removeListener()</a> , <a href="#">Selection.removeListener()</a> , <a href="#">Stage.removeListener()</a> , <a href="#">TextField.removeListener()</a>
removeMovieClip	<a href="#">removeMovieClip()</a> , <a href="#">MovieClip.removeMovieClip()</a>
removeNode	<a href="#">XML.removeNode()</a>
removeTextField	<a href="#">TextField.removeTextField()</a>
replaceSel	<a href="#">TextField.replaceSel()</a>
replaceText	<a href="#">TextField.replaceText()</a>
resolutionX	<a href="#">System.capabilities.screenResolutionX</a>

Elemento de ActionScript	Véase la entrada
resolutionY	<a href="#">System.capabilities.screenResolutionY</a>
restrict	<a href="#">TextField.restrict</a>
return	<a href="#">return</a>
reverse	<a href="#">Array.reverse()</a>
RIGHT	<a href="#">Key.RIGHT</a>
rightMargin	<a href="#">TextFormat.rightMargin</a>
round	<a href="#">Math.round()</a>
scaleMode	<a href="#">Stage.scaleMode</a>
screenColor	<a href="#">System.capabilities.screenColor</a>
screenDPI	<a href="#">System.capabilities.screenDPI</a>
screenResolutionX	<a href="#">System.capabilities.screenResolutionX</a>
screenResolutionY	<a href="#">System.capabilities.screenResolutionY</a>
scroll	<a href="#">scroll</a> , <a href="#">TextField.scroll</a>
seek	<a href="#">NetStream.seek()</a>
selectable	<a href="#">TextField.selectable</a>
Selection	<b>Clase Selection</b>
send	<a href="#">LoadVars.send()</a> , <a href="#">LocalConnection.send()</a> , <a href="#">PrintJob.send()</a> , <a href="#">XML.send()</a> , <a href="#">XMLSocket.send()</a>
sendAndLoad	<a href="#">LoadVars.sendAndLoad()</a> , <a href="#">XML.sendAndLoad()</a>
separatorBefore	<a href="#">ContextMenuItem.separatorBefore</a>
serverString	<a href="#">System.capabilities.serverString</a>
set	<a href="#">set</a>
set variable	<a href="#">set variable</a>
setBufferTime	<a href="#">NetStream.setBufferTime()</a>
setClipboard	<a href="#">System.setClipboard()</a>
setDate	<a href="#">Date.setDate()</a>
setFocus	<a href="#">Selection.setFocus()</a>
setFullYear	<a href="#">Date.setFullYear()</a>
setGain	<a href="#">Microphone.setGain()</a>
setHours	<a href="#">Date.setHours()</a>
setInterval	<a href="#">setInterval()</a>
setMask	<a href="#">MovieClip.setMask()</a>
setMilliseconds	<a href="#">Date.setMilliseconds()</a>
setMinutes	<a href="#">Date.setMinutes()</a>
setMode	<a href="#">Camera.setMode()</a>

Elemento de ActionScript	Véase la entrada
setMonth	<a href="#">Date.setMonth()</a>
setMotionLevel	<a href="#">Camera.setMotionLevel()</a>
setNewTextFormat	<a href="#">TextField.setNewTextFormat()</a>
setPan	<a href="#">Sound.setPan()</a>
setProperty	<a href="#">setProperty()</a>
setQuality	<a href="#">Camera.setQuality()</a>
setRate	<a href="#">Microphone.setRate()</a>
setRGB	<a href="#">Color.setRGB()</a>
setSeconds	<a href="#">Date.setSeconds()</a>
setSelectColor	<a href="#">TextSnapshot.setSelectColor()</a>
setSelected	<a href="#">TextSnapshot.setSelected()</a>
setSelection	<a href="#">Selection.setSelection()</a>
setSilenceLevel	<a href="#">Microphone.setSilenceLevel()</a>
setStyle	<a href="#">TextField.StyleSheet.setStyle()</a>
setTextFormat	<a href="#">TextField.setTextFormat()</a>
setTime	<a href="#">Date.setTime()</a>
setTransform	<a href="#">Color.setTransform()</a> , <a href="#">Sound.setTransform()</a>
setUseEchoSuppression	<a href="#">Microphone.setUseEchoSuppression()</a>
setUTCDate	<a href="#">Date.setUTCDate()</a>
setUTCFullYear	<a href="#">Date.setUTCFullYear()</a>
setUTCHours	<a href="#">Date.setUTCHours()</a>
setUTCMilliseconds	<a href="#">Date.setUTCMilliseconds()</a>
setUTCMinutes	<a href="#">Date.setUTCMinutes()</a>
setUTCMonth	<a href="#">Date.setUTCMonth()</a>
setUTCSeconds	<a href="#">Date.setUTCSeconds()</a>
setVolume	<a href="#">Sound.setVolume()</a>
setYear	<a href="#">Date.setYear()</a>
SharedObject	<a href="#">Clase SharedObject</a>
SHIFT (constante)	<a href="#">Key.SHIFT</a>
shift (método)	<a href="#">Array.shift()</a>
show	<a href="#">Mouse.show()</a>
showMenu	<a href="#">Stage.showMenu</a>
showSettings	<a href="#">System.showSettings()</a>
silenceLevel	<a href="#">Microphone.silenceLevel()</a>

Elemento de ActionScript	Véase la entrada
silenceTimeout	<a href="#">Microphone.silenceTimeout()</a>
sin	<a href="#">Math.sin()</a>
size	<a href="#">TextFormat.size</a>
slice	<a href="#">Array.slice()</a> , <a href="#">String.slice()</a>
smoothing	<a href="#">Video.smoothing</a>
sort	<a href="#">Array.sort()</a>
sortOn	<a href="#">Array.sortOn()</a>
Sound	<b>Clase Sound</b>
SPACE	<a href="#">Key.SPACE</a>
splice	<a href="#">Array.splice()</a>
split	<a href="#">String.split()</a>
sqrt	<a href="#">Math.sqrt()</a>
SQRT1_2	<a href="#">Math.SQRT1_2</a>
SQRT2	<a href="#">Math.SQRT2</a>
Stage	<b>Clase Stage</b>
start	<a href="#">PrintJob.start()</a> , <a href="#">Sound.start()</a>
startDrag	<a href="#">startDrag()</a> , <a href="#">MovieClip.startDrag()</a>
static	<a href="#">static</a>
status	<a href="#">XML.status</a>
stop	<a href="#">stop()</a> , <a href="#">MovieClip.stop()</a> , <a href="#">Sound.stop()</a>
stopAllSounds	<a href="#">stopAllSounds()</a>
stopDrag	<a href="#">stopDrag()</a> , <a href="#">MovieClip.stopDrag()</a>
Cadena	<b>Clase String</b> , <a href="#">String()</a>
StyleSheet (clase)	<b>Clase TextField.StyleSheet</b>
styleSheet (propiedad)	<a href="#">TextField.styleSheet</a>
substr	<a href="#">String.substr()</a>
substring	<a href="#">substring</a> , <a href="#">String.substring()</a>
super	<a href="#">super</a>
swapDepths	<a href="#">MovieClip.swapDepths()</a>
switch	<a href="#">switch</a>
System	<b>Clase System</b>
TAB	<a href="#">Key.TAB</a>
tabChildren	<a href="#">MovieClip.tabChildren</a>
tabEnabled	<a href="#">Button.tabEnabled</a> , <a href="#">MovieClip.tabEnabled</a> , <a href="#">TextField.tabEnabled</a>



Elemento de ActionScript	Véase la entrada
tabIndex	<a href="#">Button.tabIndex</a> , <a href="#">MovieClip.tabIndex</a> , <a href="#">TextField.tabIndex</a>
tabStops	<a href="#">TextFormat.tabStops</a>
tan	<a href="#">Math.tan()</a>
target	<a href="#">TextFormat.target</a>
targetPath	<a href="#">targetPath</a>
tellTarget	<a href="#">tellTarget</a>
text	<a href="#">TextField.text</a>
textColor	<a href="#">TextField.textColor</a>
TextField	<a href="#">Clase TextField</a>
TextFormat	<a href="#">Clase TextFormat</a>
textHeight	<a href="#">TextField.textHeight</a>
TextSnapshot	<a href="#">Objeto TextSnapshot</a>
textWidth	<a href="#">TextField.textWidth</a>
this	<a href="#">this</a>
throw	<a href="#">throw</a>
time	<a href="#">NetStream.time</a>
toggleHighQuality	<a href="#">toggleHighQuality()</a>
toLowerCase	<a href="#">String.toLowerCase()</a>
toString	<a href="#">Array.toString()</a> , <a href="#">Boolean.toString()</a> , <a href="#">Date.toString()</a> , <a href="#">Error.toString()</a> , <a href="#">LoadVars.toString()</a> , <a href="#">Number.toString()</a> , <a href="#">Object.toString()</a> , <a href="#">XML.toString()</a>
toUpperCase	<a href="#">String.toUpperCase()</a>
trace	<a href="#">trace()</a>
trackAsMenu	<a href="#">Button.trackAsMenu</a> , <a href="#">MovieClip.trackAsMenu</a>
true	<a href="#">true</a>
try	<a href="#">try..catch..finally</a>
type	<a href="#">TextField.type</a>
typeof	<a href="#">typeof</a>
undefined	<a href="#">undefined</a>
underline	<a href="#">TextFormat.underline</a>
unescape	<a href="#">unescape</a>
uninstall	<a href="#">CustomActions.uninstall()</a>
unloadClip	<a href="#">MovieClipLoader.unloadClip()</a>
unloadMovie	<a href="#">unloadMovie()</a> , <a href="#">MovieClip.unloadMovie()</a>
unloadMovieNum	<a href="#">unloadMovieNum()</a>

Elemento de ActionScript	Véase la entrada
unshift	<a href="#">Array.unshift()</a>
unwatch	<a href="#">Object.unwatch()</a>
UP	<a href="#">Key.UP</a>
updateAfterEvent	<a href="#">updateAfterEvent()</a>
updateProperties	<a href="#">Accessibility.updateProperties()</a>
url	<a href="#">TextFormat.url</a>
useCodePage	<a href="#">System.useCodepage</a>
useEchoSuppression	<a href="#">Microphone.useEchoSuppression()</a>
useHandCursor	<a href="#">Button.useHandCursor</a> , <a href="#">MovieClip.useHandCursor</a>
UTC	<a href="#">Date.UTC()</a>
valueOf	<a href="#">Boolean.valueOf()</a> , <a href="#">Number.valueOf()</a> , <a href="#">Object.valueOf()</a>
var	<a href="#">var</a>
variable	<a href="#">TextField.variable</a>
version	<a href="#">System.capabilities.version</a>
Video	<a href="#">Clase Video</a>
visible	<a href="#">ContextMenuItem.visible</a>
void	<a href="#">void</a>
watch	<a href="#">Object.watch()</a>
while	<a href="#">while</a>
width	<a href="#">Camera.width</a> , <a href="#">Stage.width</a> , <a href="#">Video.width</a>
with	<a href="#">with</a>
wordwrap	<a href="#">TextField.wordWrap</a>
XML	<a href="#">Clase XML</a>
xmlDecl	<a href="#">XML.xmlDecl</a>
XMLNode	<a href="#">Clase XMLNode</a>
XMLSocket	<a href="#">Clase XMLSocket</a>

## -- (decremento)

### Disponibilidad

Flash Player 4.

### Sintaxis

*--expresión*  
*expresión--*

### Parámetros

Ninguno.

### Valor devuelto

Un número.

### Descripción

Operador (aritmético); operador unario de decremento previo y decremento posterior que resta 1 a la *expression*. La forma de decremento previo del operador (*--expression*) resta 1 a la *expression* y devuelve el resultado. La forma de decremento posterior del operador (*expression--*) resta 1 a la *expression* y devuelve el valor inicial de la *expression* (el valor antes de la resta).

### Ejemplo

La forma de decremento previo del operador decrementa *x* a 2 (*x - 1 = 2*) y devuelve el resultado como *y*:

```
x = 3;  
y = --x;  
//y es igual a 2
```

La forma de decremento posterior del operador decrementa *x* a 2 (*x - 1 = 2*) y devuelve el valor original de *x* como resultado *y*:

```
x = 3;  
y = x--  
//y es igual a 3
```

## ++ (incremento)

### Disponibilidad

Flash Player 4.

### Sintaxis

*++expression*  
*expression++*

### Parámetros

Ninguno.

### Valor devuelto

Un número.

## Descripción

Operador (aritmético); operador unario de incremento previo e incremento posterior que suma 1 a *expression*. *expression* puede ser una variable, un elemento de una matriz o una propiedad de un objeto. La forma de incremento previo del operador ( $++expression$ ) suma 1 a *expression* y devuelve el resultado. La forma de incremento posterior del operador ( $expression++$ ) suma 1 a *expression* y devuelve el valor inicial de *expression* (el valor antes de la suma).

La forma de incremento previo del operador incrementa  $x$  a 2 ( $x + 1 = 2$ ) y devuelve el resultado como  $y$ :

```
x = 1;
y = ++x
//y es igual a 2
```

La forma de incremento posterior del operador incrementa  $x$  a 2 ( $x + 1 = 2$ ) y devuelve el valor original de  $x$  como el resultado  $y$ :

```
x = 1;
y = x++;
//y es igual a 1
```

## Ejemplo

En el ejemplo siguiente se utiliza  $++$  como operador de incremento posterior para ejecutar cinco veces una reproducción `while`.

```
i = 0;
while(i++ < 5){
  trace("ésta es la ejecución " + i);
}
```

En el ejemplo siguiente se utiliza  $++$  como operador de incremento previo:

```
var a = [];
var i = 0;
while (i < 10) {
  a.push(++i);
}
trace(a.join());
```

Este script muestra el resultado siguiente en el panel Salida:

1,2,3,4,5,6,7,8,9,10

En el ejemplo siguiente se utiliza  $++$  como operador de incremento posterior:

```
var a = [];
var i = 0;
while (i < 10) {
  a.push(i++);
}
trace(a.join());
```

Este script muestra el resultado siguiente en el panel Salida:

0,1,2,3,4,5,6,7,8,9

## ! (NOT lógico)

### Disponibilidad

Flash Player 4.

### Sintaxis

*!expression*

### Parámetros

Ninguno.

### Valor devuelto

Valor booleano.

### Descripción

Operador (lógico); invierte el valor booleano de una variable o expresión. Si *expression* es una variable con el valor absoluto o convertido *true*, el valor de *!expression* es *false*. Si la expresión *x && y* da como resultado *false*, la expresión *!(x && y)* da como resultado *true*.

En las expresiones siguientes se ilustra el resultado de la utilización del operador *!*:

*!true* devuelve *false*

*!false* devuelve *true*

### Ejemplo

En el ejemplo siguiente, la variable *happy* se establece en *false*. La condición *if* comprueba la condición *!happy* y, si la condición es *true*, la acción *trace()* envía una cadena al panel Salida.

```
happy = false;
if (!happy) {
    trace("no te preocupes, sé feliz");
}
```

## != (desigualdad)

### Disponibilidad

Flash Player 5.

### Sintaxis

*expression1 != expression2*

### Parámetros

Ninguno.

### Valor devuelto

Valor booleano.

## Descripción

Operador (desigualdad); comprueba exactamente lo contrario que el operador `==`. Si *expression1* es igual a *expression2*, el resultado es `false`. Al igual que con el operador `==`, la definición de *igual* depende de los tipos de datos que se comparan.

- Los números, cadenas y valores booleanos se comparan por valor.
- Las variables, objetos, matrices y funciones se comparan por referencia.

## Ejemplo

En el ejemplo siguiente se muestra el resultado del operador `!=`:

```
5 != 8 devuelve true
```

```
5 != 5 devuelve false
```

En este ejemplo se ilustra la utilización del operador `!=` en una sentencia `if`.

```
a = "David";  
b = "tonto"  
if (a != b){  
    trace("David no es tonto");  
}
```

## Véase también

`!=` (desigualdad estricta), `==` (igualdad), `===` (igualdad estricta)

# != (desigualdad estricta)

## Disponibilidad

Flash Player 6.

## Sintaxis

*expression1* `!=` *expression2*

## Descripción

Operador; comprueba exactamente lo contrario que el operador `===`. El operador de desigualdad estricta realiza las mismas acciones que el operador de desigualdad, salvo que no hay conversión de los tipos de datos. Si *expression1* es igual a *expression2* y sus tipos de datos son iguales, el resultado es `false`. Al igual que con el operador `===`, la definición de *igual* depende de los tipos de datos que se comparen.

- Los números, cadenas y valores booleanos se comparan por valor.
- Las variables, objetos, matrices y funciones se comparan por referencia.

## Ejemplo

En el código siguiente se muestra el valor devuelto de las operaciones que utilizan operadores de igualdad, de igualdad estricta y de desigualdad estricta.

```
s1 = new String("5");  
s2 = new String("5");  
s3 = new String("Hola");  
n = new Number(5);  
b = new Boolean(true);
```

```

s1 == s2; // true
s1 == s3; // false
s1 == n; // true
s1 == b; // false

s1 === s2; // true
s1 === s3; // false
s1 === n; // false
s1 === b; // false

s1 !== s2; // false
s1 !== s3; // true
s1 !== n; // true
s1 !== b; // true

```

### Véase también

`!=` (desigualdad), `==` (igualdad), `===` (igualdad estricta)

## % (módulo)

### Disponibilidad

Flash Player 4. En los archivos de Flash 4, el operador % se amplía en el archivo SWF como `x - int(x/y) * y`, y es posible que no sea tan rápido ni tan preciso en versiones posteriores de Flash Player.

### Sintaxis

*expression1* % *expression2*

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Operador (aritmético); calcula el resto de *expression1* dividida por *expression2*. Si cualquiera de los parámetros de *expression* no es numérico, el operador módulo intenta convertirlos en números. La *expression* puede ser un número o una cadena que se convierte en un valor numérico.

### Ejemplo

A continuación, se muestra un ejemplo numérico en el que se utiliza el operador módulo (%).

```

trace (12 % 5);
// devuelve 2
trace (4.3 % 2.1);
// devuelve aproximadamente 0.1

```

## %= (asignación de módulo)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* %= *expression2*

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Operador (asignación compuesta aritmética); asigna a *expression1* el valor de *expression1* % *expression2*. Por ejemplo, las dos expresiones siguientes son iguales:

```
x %= y
x = x % y
```

### Ejemplo

En el ejemplo siguiente se asigna el valor 4 a la variable *x*.

```
x = 14;
y = 5;
trace(x %= y);
// devuelve 4
```

### Véase también

[% \(módulo\)](#)

## & (operador AND en modo bit)

### Disponibilidad

Flash Player 5. En Flash 4, el operador & se utilizaba para concatenar cadenas. En Flash 5 y versiones posteriores, el operador & es un operador AND en modo bit y deben utilizarse los operadores `add` y `+` para concatenar cadenas. Los archivos de Flash 4 que utilizan el operador & se actualizan automáticamente de forma que pasen a utilizar `add` al abrirlos en el entorno de edición de Flash 5 o posterior.

### Sintaxis

*expression1* & *expression2*

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.



## Descripción

Operador (en modo bit); convierte *expression1* y *expression2* en números enteros de 32 bits sin signo y realiza una operación booleana AND en cada bit de los parámetros enteros. El resultado es un nuevo número entero de 32 bits sin signo.

## && (AND lógico)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* && *expression2*

### Parámetros

Ninguno.

### Valor devuelto

Valor booleano.

### Descripción

Operador (lógico); realiza una operación booleana en los valores de una o ambas expresiones. Calcula el resultado de *expression1* (la expresión situada a la izquierda del operador) y devuelve *false* si la expresión da como resultado *false*. Si *expression1* da como resultado *true*, el resultado es *expression2* (la expresión situada a la derecha del operador). Si *expression2* da como resultado *true*, el resultado final es *true*; si no, es *false*.

### Ejemplo

En este ejemplo se utiliza el operador && para realizar una prueba a fin de determinar si un jugador ha ganado el juego. La variable *turns* y la variable *score* se actualizan para reflejar los turnos jugados y los puntos que consigue el jugador durante el juego. El script siguiente muestra “¡Ha ganado la partida!” en el panel Salida cuando la puntuación del jugador alcanza los 75 puntos o más en 3 turnos o menos.

```
turns=2;
score=77;
winner = (turns <= 3) && (score >= 75);
if (winner) {
    trace("¡Ha ganado la partida!");
} else {
    trace("¡Inténtelo de nuevo!");
}
```

## &= (asignación AND en modo bit)

### Disponibilidad

Flash Player 5.

### Sintaxis

*expression1* &= *expression2*

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Operador; asigna a *expression1* el valor de *expression1* & *expression2*. Por ejemplo, las dos expresiones siguientes son iguales.

```
x &= y;  
x = x & y;
```

### Ejemplo

En el ejemplo siguiente se asigna el valor 9 a x.

```
x = 15;  
y = 9;  
trace(x &= y);  
// devuelve 9
```

### Véase también

[& \(operador AND en modo bit\)](#)

## () (paréntesis)

### Disponibilidad

Flash Player 4.

### Sintaxis

```
(expression1, expression2);  
function(parameter1,..., parameterN);
```

### Parámetros

*expression1*, *expression2*    Números, cadenas, variables o texto.

*function*    Función que se va a realizar en el contenido de los paréntesis.

*parameter1*...*parameterN*    Serie de parámetros que deben ejecutarse antes de que los resultados se pasen como parámetros a la función de fuera de los paréntesis.

### Valor devuelto

Ninguno.

## Descripción

Operador; realiza una operación de agrupación en uno o más parámetros o rodea uno o más parámetros y los pasa como parámetros a una función fuera de los paréntesis.

Sintaxis 1: controla el orden de ejecución de los operadores en la expresión. Los paréntesis prevalecen sobre el orden de precedencia normal y hacen que las expresiones entre paréntesis se calculen primero. Cuando los paréntesis están anidados, el contenido de los paréntesis interiores se calcula antes que el contenido de los exteriores.

Sintaxis 2: rodea uno o más parámetros y los pasa como parámetros a la función de fuera de los paréntesis.

## Ejemplo

Sintaxis 1: en las sentencias siguientes se muestra la utilización de los paréntesis para controlar el orden de ejecución de las expresiones. El valor de cada expresión se muestra debajo de cada línea, de la manera siguiente:

```
trace((2 + 3) * (4 + 5));  
// muestra 45
```

```
trace(2 + (3 * (4 + 5)));  
// muestra 29
```

```
trace(2 + (3 * 4) + 5);  
// muestra 19
```

Sintaxis 2: en los ejemplos siguientes se muestra la utilización de paréntesis con funciones.

```
getDate();  
  
invoice(item, amount);  
  
function traceParameter(param){  
    trace(param);  
}  
traceParameter(2*2);
```

## Véase también

[with](#)

## - (menos)

### Disponibilidad

Flash Player 4.

### Sintaxis

(Negación) *-expression*

(Resta) *expression1 - expression2*

### Parámetros

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Operador (aritmético); utilizado para negación o resta.

Sintaxis 1: cuando se utiliza para negar, invierte el signo de *expression* numérica.

Sintaxis 2: cuando se utiliza para restar, realiza una resta aritmética en dos expresiones numéricas, de manera que se sustrae la *expression2* de la *expression1*. Cuando ambas expresiones son números enteros, la diferencia es un número entero. Cuando una o ambas expresiones son números de coma flotante, la diferencia es un número de coma flotante.

**Ejemplo**

Sintaxis 1: en la sentencia siguiente se invierte el signo de la expresión  $2 + 3$ .

$-(2 + 3)$

El resultado es -5.

Sintaxis 2: en la sentencia siguiente se resta el entero 2 del entero 5.

$5 - 2$

El resultado es 3, que es un entero.

Sintaxis 2: en la sentencia siguiente se resta el número de coma flotante 1,5 del número de coma flotante 3,25.

$3.25 - 1.5$

El resultado es 1.75, que es un número de coma flotante.

**\* (multiplicación)****Disponibilidad**

Flash Player 4.

**Sintaxis**

*expression1* \* *expression2*

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Operador (aritmético); multiplica dos expresiones numéricas. Cuando ambas expresiones son números enteros, el producto es un número entero. Cuando una o ambas expresiones son números de coma flotante, el producto es un número de coma flotante.

## Ejemplo

Sintaxis 1: en la sentencia siguiente se multiplican los enteros 2 y 3:

```
2 * 3
```

El resultado es 6, que es un entero.

Sintaxis 2: en esta sentencia se multiplican los números de coma flotante 2.0 y 3.1416:

```
2.0 * 3.1416
```

El resultado es 6.2832, que es un número de coma flotante.

## \*= (asignación de multiplicación)

### Disponibilidad

Flash Player 4.

### Sintaxis

```
expression1 *= expression2
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Operador (asignación compuesta aritmética); asigna a *expression1* el valor de *expression1* \* *expression2*. Por ejemplo, las dos expresiones siguientes son iguales:

```
x *= y  
x = x * y
```

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se asigna el valor 50 a la variable *x*.

```
x = 5;  
y = 10;  
trace (x *= y);  
// devuelve 50
```

Sintaxis 2: en las líneas segunda y tercera del ejemplo siguiente se calculan las expresiones situadas a la derecha de los signos igual y se asignan los resultados a *x* e *y*.

```
i = 5;  
x = 4 - 6;  
y = i + 2;  
trace(x *= y);  
// devuelve -14
```

### Véase también

[\\* \(multiplicación\)](#)

## , (coma)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1, expression2*

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Operador; calcula el resultado de *expression1*, después, el de *expression2* y devuelve el valor de *expression2*. Este operador se utiliza principalmente con la sentencia de reproducción indefinida `for`.

### Ejemplo

El siguiente código de ejemplo utiliza el operador coma:

```
var a=1, b=2, c=3;
```

Esto equivale a escribir el código siguiente:

```
var a=1;  
var b=2;  
var c=3;
```

## . (punto)

### Disponibilidad

Flash Player 4.

### Sintaxis

*object.property\_or\_method*

*instancename.variable*

*instancename.childinstance.variable*

### Parámetros

*object* Instancia de una clase. El objeto puede ser una instancia de cualquiera de las clases incorporadas de ActionScript o una clase personalizada. Este parámetro siempre se encuentra a la izquierda del operador punto (.).

*property\_or\_method* Nombre de una propiedad o método asociado con un objeto. Todos los métodos y las propiedades válidos de los objetos incorporados se listan en las tablas de resumen de métodos y propiedades de dicha clase. Este parámetro siempre se encuentra a la derecha del operador punto (.).

*instancename* Nombre de instancia de un clip de película.

*childinstance* Instancia de clip de película que depende de otro clip de película o que está anidado en éste.

*variable* Variable de la línea de tiempo del nombre de instancia del clip de película situado a la izquierda del operador punto (.).

#### Valor devuelto

Ninguno.

#### Descripción

Operador; se utiliza para desplazarse por las jerarquías de clips de película a fin de acceder a clips de película anidados, a variables o a propiedades. El operador punto también se utiliza para comprobar o establecer las propiedades de un objeto, ejecutar un método de un objeto o crear una estructura de datos.

#### Ejemplo

En la sentencia siguiente se identifica el valor actual de la variable `hairColor` en el clip de película `person_mc`.

```
person_mc.hairColor
```

Esto es equivalente a la sintaxis de Flash 4 que se muestra a continuación:

```
/person_mc:hairColor
```

## : (tipo)

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
[modifiers] [var] variableName[:type]  
function functionName()[:type] { ... }  
function functionName(parameter1[:type], ... , parameterN[:type]) { ... }
```

#### Parámetros

*variableName* Identificador para una variable.

*type* Tipo de datos nativo, nombre de clase que se ha definido o nombre de interfaz.

*functionName* Identificador para una función.

*parameter* Identificador para un parámetro de función.

#### Descripción

Operador; especifica el tipo de variable, el tipo de devolución de la función o el tipo de parámetro de la función. Cuando se utiliza en una declaración o asignación de variable, este operador especifica el tipo de variable; cuando se utiliza en una declaración o definición de función, este operador especifica el tipo de devolución de la función; cuando se utiliza con un parámetro de función en una definición de función, este operador especifica el tipo de variable esperado para el parámetro.

Los tipos son una función exclusiva de la compilación. Todos los tipos se verifican al realizar la compilación y se generan errores si existen discordancias. Para más información, consulte [Apéndice A, “Mensajes de error”, en la página 783](#). Las discordancias pueden producirse durante operaciones de asignación, llamadas de función y diferenciación de miembros de clase utilizando el operador punto (.). A fin de evitar errores de discordancia de tipo, debe utilizarse la especificación explícita de tipos (véase [“Strict data typing” en la página 40](#)).

Los tipos que pueden utilizarse son: todos los tipos de clases, interfaces y objetos nativos que defina, así como Void y Function (que sólo existen como tipos, no como objetos). Los tipos nativos reconocidos son: Array, Boolean, Button, Color, CustomActions, Date, Function, LoadVars, LocalConnection, Microphone, MovieClip, NetConnection, NetStream, Number, Object, SharedObject, Sound, String, TextField, TextFormat, Video, Void, XML, XMLNode y XMLSocket.

### Ejemplo

Sintaxis 1: en el ejemplo siguiente se declara una variable pública denominada `userName` cuyo tipo es `String` y se le asigna una cadena vacía.

```
public var userName:String = "";
```

Sintaxis 2: en este ejemplo se muestra cómo especificar el tipo de parámetro de una función. El código siguiente define una función denominada `setDate()` que toma un parámetro denominado `currentDate` del tipo `Date`.

```
function setDate(currentDate>Date) {  
    this.date = currentDate;  
}
```

Sintaxis 3: el código siguiente define una función denominada `squareRoot()` que toma un parámetro denominado `val` de tipo `Number` y devuelve la raíz cuadrada de `val`, que también es un tipo `Number`.

```
function squareRoot(val:Number):Number {  
    return Math.sqrt(val);  
}
```

## ?: (condicional)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* ? *expression2* : *expresión3*

### Parámetros

*expression1* Expresión que da como resultado un valor booleano, normalmente una expresión de comparación, como `x < 5`.

*expression2*, *expresión3* Valores de cualquier tipo.

### Valor devuelto

Ninguno.



## Descripción

Operador; ordena a Flash que calcule el resultado de *expression1* y, si el valor de *expression1* es true, devuelve el valor de *expression2*; en caso contrario, devuelve el valor de *expression3*.

## Ejemplo

En la sentencia siguiente se asigna el valor de la variable x a la variable z porque *expression1* da como resultado true:

```
x = 5;  
y = 10;  
z = (x < 6) ? x: y;  
trace (z);  
// devuelve 5
```

## / (división)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* / *expression2*

### Parámetros

*expression* Número o variable que da como resultado un número.

### Valor devuelto

Ninguno.

### Descripción

Operador (aritmético); divide la *expression1* por la *expression2*. El resultado de la operación de división es un número de coma flotante de doble precisión.

## Ejemplo

En la sentencia siguiente se divide el número de coma flotante 22.0 entre 7.0 y, a continuación, se muestra el resultado en el panel Salida.

```
trace(22.0 / 7.0);
```

El resultado es 3.1429, que es un número de coma flotante.

## // (delimitador de comentario)

### Disponibilidad

Flash 1.

### Sintaxis

// comentario

### Parámetros

*comment* Cualquier carácter.

## Valor devuelto

Ninguno.

## Descripción

Comentario; indica el comienzo de un comentario de script. Cualquier carácter que aparezca entre el delimitador de comentario `//` y el carácter de final de línea se interpreta como un comentario y el intérprete de ActionScript lo pasa por alto.

## Ejemplo

En este script se utilizan delimitadores de comentario para identificar la primera, la tercera la quinta y la séptima línea como comentarios.

```
// registra la posición X del clip de película ball
ballX = ball._x;
// registra la posición Y del clip de película ball
ballY = ball._y;
// registra la posición X del clip de película bat
batX = bat._x;
// registra la posición Y del clip de película bat
batY = bat._y;
```

## Véase también

`/*` (delimitador de comentario)

# /\* (delimitador de comentario)

## Disponibilidad

Flash Player 5.

## Sintaxis

```
/* comment */

/*
comment
comment
*/
```

## Parámetros

*comment* Cualquier carácter.

## Valor devuelto

Ninguno.

## Descripción

Comentario; indica una o más líneas de comentarios del script. Cualquier carácter que aparezca entre la etiqueta de apertura de comentario `/*` y la etiqueta de cierre de comentario `*/` se interpreta como un comentario y el intérprete de ActionScript lo pasa por alto. Utilice el primer tipo de sintaxis para identificar comentarios de una sola línea. Utilice el segundo tipo de sintaxis para identificar comentarios en varias líneas sucesivas. Si no se especifica la etiqueta de cierre `*/` cuando se utiliza esta forma de delimitador de comentario, se generará un mensaje de error.

## Ejemplo

En este script se utilizan delimitadores de comentario al principio del script.

```
/* registra las posiciones X e Y de los clips de película de  
ball y bat  
*/  
  
ballX = ball._x;  
ballY = ball._y;  
batX = bat._x;  
batY = bat._y;
```

## Véase también

`//` (delimitador de comentario)

# /= (asignación de división)

## Disponibilidad

Flash Player 4.

## Sintaxis

*expression1* /= *expression2*

## Parámetros

*expression1*, *expression2* Número o variable que da como resultado un número.

## Valor devuelto

Ninguno.

## Descripción

Operador (asignación compuesta aritmética); asigna a *expression1* el valor de *expression1* / *expression2*. Por ejemplo, las dos sentencias siguientes son iguales:

```
x /= y  
x = x / y
```

## Ejemplo

A continuación se muestra la utilización del operador /= con variables y números:

```
x = 10;  
y = 2;  
x /= y;  
// x contiene ahora el valor 5
```

## [] (acceso a matriz)

### Disponibilidad

Flash Player 4.

### Sintaxis

```
my_array = [ "a0", a1, ...aN]
myMultiDimensional_array = [[ "a0", ...aN], ...[ "a0", ...aN]]
my_array[E] = value
myMultiDimensional_array[E][E] = value
object["value"]
```

### Parámetros

*my\_array* Nombre de una matriz.

*a0, a1, ...aN* Elementos de una matriz.

*myMultiDimensional\_array* Nombre de una matriz multidimensional simulada.

*E* Número (o índice) de un elemento de una matriz.

*object* Nombre de un objeto.

*value* Cadena o expresión que da como resultado una cadena con el nombre de una propiedad del objeto.

### Valor devuelto

Ninguno.

### Descripción

Operador; inicializa una nueva matriz o una matriz multidimensional con los elementos especificados (*a0*, etc.) o accede a los elementos de una matriz. El operador de acceso a matriz permite establecer y recuperar dinámicamente nombres de instancias, variables y objetos. También permite acceder a propiedades de objetos.

Sintaxis 1: una matriz es un objeto cuyas propiedades se denominan *elementos*, que se identifican individualmente mediante un número denominado *índice*. Cuando se crea una matriz, los elementos se especifican con el operador de acceso a matriz (o *corchetes*). Una matriz puede contener elementos de varios tipos. Por ejemplo, la matriz siguiente, denominada `employee`, dispone de tres elementos; el primero es un número y los dos segundos son cadenas (entre comillas).

```
employee = [15, "Barbara", "Erick"];
```

Sintaxis 2: puede anidar corchetes para simular matrices multidimensionales. En el código siguiente se crea una matriz denominada `ticTacToe` con tres elementos; cada elemento también es una matriz que contiene tres elementos.

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];
```

```
// elija Depurar > Mostrar variables en el modo de probar película
// para ver una lista de elementos de matriz
```

Sintaxis 3: especifique el índice de cada elemento entre corchetes para acceder a él directamente; puede agregar un nuevo elemento a una matriz o cambiar o recuperar el valor de un elemento existente. El primer elemento de una matriz siempre es el 0:

```
my_array[0] = 15;
my_array[1] = "Hola";
my_array[2] = true;
```

Puede utilizar corchetes para agregar un cuarto elemento como en el ejemplo siguiente:

```
my_array[3] = "George";
```

Sintaxis 4: puede utilizar corchetes para acceder a un elemento de una matriz multidimensional. El primer grupo de corchetes identifica el elemento de la matriz original y el segundo grupo identifica el elemento en la matriz anidada. En la línea de código siguiente el número 6 se envía al panel Salida.

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];
trace(ticTacToe[1][2]);

// devuelve 6
```

Sintaxis 5: puede utilizar el operador de acceso a matriz en lugar de la función `eval` para establecer y recuperar de forma dinámica los valores de nombres de clip de película o las propiedades de un objeto:

```
name["mc" + i] = "left_corner";
```

## Ejemplo

Sintaxis 1: en los siguientes ejemplos de código se muestran dos maneras diferentes de crear un nuevo objeto Array vacío; en la primera línea se utilizan corchetes.

```
my_array = [];
my_array = new Array();
```

Sintaxis 1 y 2: en el ejemplo siguiente se crea una matriz denominada `employee_array` y se utiliza la acción `trace()` para enviar los elementos al panel Salida. En la cuarta línea, se cambia un elemento de la matriz y la quinta línea envía la matriz recién modificada al panel Salida:

```
employee_array = ["Barbara", "George", "Mary"];
trace(employee_array);
// Barbara, George, Mary
employee_array[2]="Sam";
trace(employee_array);
// Barbara, George, Sam
```

Sintaxis 3: en el ejemplo siguiente, se calcula el resultado de la expresión entre corchetes ("`piece`"+`i`) y el resultado se utiliza como nombre de la variable que debe recuperarse del clip de película `my_mc`. En este ejemplo, la variable `i` debe pertenecer a la misma línea de tiempo que el botón. Si, por ejemplo, la variable `i` es igual a 5, el valor de la variable `piece5` del clip de película `my_mc` se visualizará en el panel Salida:

```
on(release) {
    x = my_mc["piece"+i];
    trace(x);
}
```

Sintaxis 3: en el código que se muestra a continuación, se calcula el resultado de la expresión entre corchetes y éste se utiliza como nombre de la variable que se va a recuperar del clip de película

```
name_mc:
```

```
name_mc["A" + i];
```

Si está familiarizado con la sintaxis de barras de ActionScript de Flash 4, puede utilizar la función `eval` para obtener el mismo resultado:

```
eval("name.A" & i);
```

Sintaxis 3: puede utilizar el operador de acceso a matriz situado a la izquierda de una sentencia de asignación para establecer dinámicamente nombres de instancias, variables y objetos:

```
name[index] = "Gary";
```

**Véase también**

[Clase Array](#), [Clase Object](#), [eval\(\)](#)

## ^ (XOR en modo bit)

### Disponibilidad

Flash Player 5.

### Sintaxis

```
expression1 ^ expression2
```

### Parámetros

```
expression1, expression2    Número.
```

### Valor devuelto

Ninguno.

### Descripción

Operador (en modo bit); convierte *expression1* y *expression2* en números enteros de 32 bits sin signo y devuelve un 1 en cada posición de bit donde los bits correspondientes de *expression1* o de *expression2*, pero no ambos, son 1.

### Ejemplo

En el ejemplo siguiente se utiliza el operador XOR en modo bit en los decimales 15 y 9 y se asigna el resultado a la variable `x`.

```
// decimal 15 = binario 1111
// decimal 9 = binario 1001
x = 15 ^ 9
trace(x)
// 1111 ^ 1001 = 0110
// devuelve el decimal 6 (= binario 0110)
```

## **^= (asignación XOR en modo bit)**

### **Disponibilidad**

Flash Player 5.

### **Sintaxis**

*expression1* ^= *expression2*

### **Parámetros**

*expression1*, *expression2* Enteros y variables.

### **Valor devuelto**

Ninguno.

### **Descripción**

Operador (asignación compuesta en modo bit); asigna a *expression1* el valor de *expression1* ^ *expression2*. Por ejemplo, las dos sentencias siguientes son iguales:

```
x ^= y
x = x ^ y
```

### **Ejemplo**

A continuación se muestra un ejemplo de una operación ^=.

```
// decimal 15 = binario 1111
x = 15;
// decimal 9 = binario 1001
y = 9;
trace(x ^= y);
//devuelve el decimal 6 (= binario 0110)
```

### **Véase también**

[^ \(XOR en modo bit\)](#)

## **{ } (inicializador de objeto)**

### **Disponibilidad**

Flash Player 5.

### **Sintaxis**

*object* = {*name1*: *value1*, *name2*: *value2*,...*nameN*: *valueN*}

### **Parámetros**

*object* Objeto que debe crearse.

*name1,2,...N* Nombres de las propiedades.

*value1,2,...N* Valores correspondientes para cada propiedad *nombre*.

### **Valor devuelto**

Ninguno.

## Descripción

Operador; crea un nuevo objeto y lo inicializa con los pares de propiedades *name* y *value* especificados. Utilizar este operador es lo mismo que utilizar la sintaxis de `new Object` y asignar los pares de propiedades con el operador de asignación. El prototipo del objeto recién creado se denomina genéricamente objeto *Object*.

## Ejemplo

En la primera línea del código siguiente se crea un objeto vacío con el operador de inicializador y en la segunda línea se crea un nuevo objeto con una función constructora.

```
object = {};  
object = new Object();
```

En el ejemplo siguiente se crea un objeto `account` y se inicializan las propiedades `name`, `address`, `city`, `state`, `zip` y `balance` con los valores correspondientes.

```
account = { name: "Betty Skate",  
  address: "123 Main Street",  
  city: "Blossomville",  
  state: "California",  
  zip: "12345",  
  balance: "1000" };
```

El ejemplo siguiente muestra cómo los inicializadores de matriz y de objeto pueden anidarse unos dentro de otros.

```
person = { name: "Gina Vechio",  
  children: [ "Ruby", "Chickie", "Puppa" ] };
```

En el ejemplo siguiente se utiliza la información del ejemplo anterior y se genera el mismo resultado con las funciones constructoras.

```
person = new Object();  
person.name = 'Gina Vechio';  
person.children = new Array();  
person.children[0] = 'Ruby';  
person.children[1] = 'Chickie';  
person.children[2] = 'Puppa';
```

## Véase también

[\[\] \(acceso a matriz\)](#), [new](#), [Clase Object](#)

# | (OR en modo bit)

## Disponibilidad

Flash Player 5.

## Sintaxis

*expression1* | *expression2*

## Parámetros

*expression1*, *expression2*    Número.

## Valor devuelto

Ninguno.



## Descripción

Operador (en modo bit); convierte *expression1* y *expression2* en números enteros de 32 bits sin signo y devuelve un 1 en cada posición de bit donde los bits correspondientes de *expression1* o *expression2* son 1.

## Ejemplo

A continuación se muestra un ejemplo de una operación OR en modo bit.

```
// decimal 15 = binario 1111
x = 15;
// decimal 9 = binario 1001
y = 9;
trace(x | y);
// 1111 | 0011 = 1111
// devuelve el decimal 15 (= binario 1111)
```

# || (OR lógico)

## Disponibilidad

Flash Player 4.

## Sintaxis

*expression1* || *expression2*

## Parámetros

*expression1*, *expression2* Valor booleano o expresión que se convierte en un valor booleano.

## Valor devuelto

Valor booleano.

## Descripción

Operador (lógico); calcula el resultado de *expression1* y *expression2*. El resultado es *true* si una o las dos expresiones dan como resultado *true*; el resultado es *false* sólo si las dos expresiones dan como resultado *false*. Puede utilizar el operador OR lógico con cualquier número de operandos; si algún operando da como resultado *true*, el resultado es *true*.

Con expresiones no booleanas, el operador lógico OR hace que Flash calcule el resultado de la expresión situada a la izquierda; si puede convertirse en *true*, el resultado es *true*. En caso contrario, calcula el valor de la expresión de la derecha y el resultado es el valor de dicha expresión.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se utiliza el operador || en una sentencia *if*. La segunda expresión da como resultado *true*, de manera que el resultado final es *true*:

```
x = 10
y = 250
start = false
if(x > 25 || y > 200 || start){
    trace('se ha pasado la prueba de OR lógico');
}
```

Sintaxis 2: en este ejemplo se demuestra cómo una expresión no booleana puede generar un resultado inesperado. Si la expresión situada a la izquierda se convierte en `true`, ese resultado se devuelve sin convertir la expresión situada a la derecha.

```
function fx1(){
    trace ("se ha llamado a fx1");
    returns true;
}
function fx2(){
    trace ("se ha llamado a fx2");
    return true;
}
if (fx1() || fx2()){
    trace ("Se ha entrado en la sentencia IF");
}
// Lo siguiente se envía al panel Salida:
// se ha llamado a fx1
// Se ha entrado en la sentencia IF
```

## **|= (asignación OR en modo bit)**

### **Disponibilidad**

Flash Player 5.

### **Sintaxis**

*expression1* |= *expression2*

### **Parámetros**

*expression1*, *expression2* Número o variable.

### **Valor devuelto**

Ninguno.

### **Descripción**

Operador (asignación compuesta en modo bit); asigna a *expression1* el valor de *expression1* | *expression2*. Por ejemplo, las dos sentencias siguientes son iguales:

```
x |= y;
x = x | y;
```

### **Ejemplo**

En el ejemplo siguiente se utiliza el operador |=:

```
// decimal 15 = binario 1111
x = 15;
// decimal 9 = binario 1001
y = 9;
trace(x |= y);
// 1111 |= 1001
//devuelve el decimal 15 (= binario 1111)
```

### **Véase también**

[| \(OR en modo bit\)](#)

## ~ (NOT en modo bit)

### Disponibilidad

Flash Player 5.

### Sintaxis

*~ expression*

### Parámetros

*expression* Número.

### Valor devuelto

Ninguno.

### Descripción

Operador (en modo bit); convierte *expression* en un número entero de 32 bits sin signo y, después, invierte los bits. Una operación NOT en modo bit cambia el signo de un número y le resta 1.

### Ejemplo

En el ejemplo siguiente se muestra una operación NOT en modo bit realizada en una variable.

```
a = 0;
trace ("when a = 0, ~a = "+~a);
// cuando a = 0, ~a = -1
a = 1;
trace ("when a = 1, ~a = "+~a);
// cuando a = 0, ~a = -2
// por lo tanto, ~0=-1 y ~1=-2
```

## + (suma)

### Disponibilidad

Flash Player 4; Flash Player 5. En Flash 5 y versiones posteriores, + es un operador numérico o un concatenador de cadenas en función del tipo de parámetro. En Flash 4, + sólo es un operador numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos. A continuación se muestra un ejemplo de conversión de un archivo de Flash 4 que contiene una comparación de calidad numérica:

Archivo de Flash 4:

`x + y`

Archivo de Flash 5 o posterior convertido:

`Number(x) + Number(y)`

### Sintaxis

*expression1 + expression2*

### Parámetros

*expression1, expression2* Número o cadena.

## Valor devuelto

Ninguno.

## Descripción

Operador; agrega expresiones numéricas o concatena (combina) cadenas. Si una expresión es una cadena, todas las demás expresiones se convierten en cadenas y se concatenan.

Si ambas expresiones son números enteros, la suma es un número entero; si cualquiera de ellas o ambas expresiones son números de coma flotante, la suma es un número de coma flotante.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se concatenan dos cadenas y se muestra el resultado en el panel Salida.

```
name = "Carlos";  
instrument = "la batería";  
trace (name + " toca " + instrument);
```

Sintaxis 2: las variables asociadas con campos de texto dinámico y de introducción de texto tienen el tipo de datos String. En el ejemplo siguiente, la variable `deposit` es un campo de introducción de texto del escenario. Cuando un usuario introduce un depósito, el script intenta agregar `deposit` a `oldBalance`. Sin embargo, como `deposit` es un tipo de datos String, el script concatena los valores de la variable (los combina para formar una cadena) en lugar de sumarlos.

```
oldBalance = 1345.23;  
currentBalance = deposit + oldBalance;  
trace (currentBalance);
```

Por ejemplo, si un usuario introduce 475 en el campo de texto `deposit`, la acción `trace()` envía el valor 4751345.23 al panel Salida.

Para corregirlo, utilice la función `Number()` para convertir la cadena en un número, como se muestra a continuación:

```
currentBalance = Number(deposit) + oldBalance;
```

Sintaxis 3: en esta sentencia se añaden los números enteros 2 y 3 y se muestra el número entero 5 resultante en el panel Salida:

```
trace (2 + 3);
```

En esta sentencia se añaden los números de coma flotante 2.5 y 3.25 y se muestra el resultado, 5.75, también un número de coma flotante, en el panel Salida:

```
trace (2.5 +3.25);
```

## Véase también

[\\_accProps](#)

## += (asignación de suma)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* += *expression2*

## Parámetros

*expression1, expression2* Número o cadena.

## Valor devuelto

Ninguno.

## Descripción

Operador (asignación compuesta aritmética); asigna a *expression1* el valor de *expression1* + *expression2*. Por ejemplo, las dos sentencias siguientes tienen el mismo resultado:

```
x += y;  
x = x + y;
```

Este operador también realiza la concatenación de cadenas. Todas las reglas del operador de suma (+) se aplican al operador de asignación de suma (+=).

## Ejemplo

En el ejemplo siguiente se muestra una utilización numérica del operador +=.

```
x = 5;  
y = 10;  
x += y;  
trace(x);  
//x devuelve 15
```

En este ejemplo se utiliza el operador += con una expresión de cadena y se envía "Me llamo Gerardo" al panel Salida.

```
x = "Me llamo "  
x += "Gerardo"  
trace(x)  
// devuelve "Me llamo Gerardo"
```

## Véase también

[+ \(suma\)](#)

## < (menor que)

### Disponibilidad

Flash Player 4; Flash Player 5. En Flash 5 y versiones posteriores, el operador < (menor que) es un operador de comparación que puede manejar varios tipos de datos. En Flash 4, < es un operador numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos. A continuación se muestra un ejemplo de conversión de un archivo de Flash 4 que contiene una comparación numérica.

Archivo de Flash 4:

```
x < y
```

Archivo de Flash 5 o posterior convertido:

```
Number(x) < Number(y)
```

## Sintaxis

*expression1* < *expression2*

## Parámetros

*expression1, expression2* Número o cadena.

## Descripción

Operador (comparación); compara dos expresiones y determina si *expression1* es menor que *expression2*; si es así, el operador devuelve *true*. Si *expression1* es mayor o igual que *expression2*, el operador devuelve *false*. Las expresiones de cadena se comparan en orden alfabético; las letras mayúsculas van antes que las minúsculas.

## Ejemplo

En los ejemplos siguientes se muestra cómo se devuelven *true* y *false* tanto para comparaciones numéricas como de cadenas.

```
3 < 10;  
// true  
  
10 < 3;  
// false  
  
"Allen" < "Jack";  
// true  
  
"Jack" < "Allen";  
// false  
  
"11" < "3";  
// true  
  
"11" < 3;  
// comparación numérica  
// false  
  
"C" < "abc";  
// false  
  
"A" < "a";  
// true
```

# « (desplazamiento a la izquierda en modo bit)

## Disponibilidad

Flash Player 5.

## Sintaxis

*expression1* << *expression2*

## Parámetros

*expression1* Número o expresión que se va a desplazar a la izquierda.

*expression2* Número o expresión que se convierte en un entero entre 0 y 31.

## Valor devuelto

Ninguno.

## Descripción

Operador (en modo bit); convierte *expression1* y *expression2* en números enteros de 32 bits y desplaza todos los bits de *expression1* hacia la izquierda el número de espacios especificado por el número entero resultante de la conversión de *expression2*. Las posiciones de bits que se han vaciado como resultado de esta operación se rellenan con 0. Desplazar un valor una posición hacia la izquierda es equivalente a multiplicarlo por 2.

## Ejemplo

En el ejemplo siguiente, el entero 1 se desplaza 10 bits a la izquierda.

```
x = 1 << 10
```

El resultado de esta operación es  $x = 1024$ . Esto es debido a que 1 decimal es igual a 1 binario, 1 binario desplazado 10 posiciones a la izquierda es 10000000000 binario y 10000000000 binario es 1024 decimal.

En el ejemplo siguiente, el entero 7 se desplaza 8 bits a la izquierda.

```
x = 7 << 8
```

El resultado de esta operación es  $x = 1792$ . Esto es debido a que 7 decimal es igual a 111 binario, 111 binario desplazado 8 posiciones a la izquierda es 11100000000 binario y 11100000000 binario es 1792 decimal.

## Véase también

>= (desplazamiento a la derecha en modo bit y asignación), >> (desplazamiento a la derecha en modo bit), <<= (desplazamiento a la izquierda en modo bit y asignación)

# <<= (desplazamiento a la izquierda en modo bit y asignación)

## Disponibilidad

Flash Player 5.

## Sintaxis

```
expression1 <<= expression2
```

## Parámetros

*expression1* Número o expresión que se va a desplazar a la izquierda.

*expression2* Número o expresión que se convierte en un entero entre 0 y 31.

## Valor devuelto

Ninguno.

## Descripción

Operador (asignación compuesta en modo bit); este operador realiza una operación de desplazamiento a la izquierda en modo bit y almacena el contenido como resultado en *expression1*. Las dos expresiones siguientes son equivalentes.

```
A <<= B  
A = (A << B)
```

## Véase también

<< (desplazamiento a la izquierda en modo bit), >>= (desplazamiento a la derecha en modo bit y asignación), >> (desplazamiento a la derecha en modo bit)

## <= (menor o igual que)

### Disponibilidad

Flash Player 4.

Archivo de Flash 4:

```
x <= y
```

Archivo de Flash 5 o posterior convertido:

```
Number(x) <= Number(y)
```

### Sintaxis

```
expression1 <= expression2
```

### Parámetros

*expression1*, *expression2* Número o cadena.

### Valor devuelto

Valor booleano.

### Descripción

Operador (comparación); compara dos expresiones y determina si *expression1* es menor o igual que *expression2*; si lo es, el operador devuelve *true*. Si *expression1* es mayor que *expression2*, el operador devuelve *false*. Las expresiones de cadena se comparan en orden alfabético; las letras mayúsculas van antes que las minúsculas.

En Flash 5, el operador menor o igual que (<=) es un operador de comparación que puede manejar varios tipos de datos. En Flash 4, <= sólo es un operador numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos. A continuación se muestra un ejemplo de conversión de un archivo de Flash 4 que contiene una comparación numérica.

### Ejemplo

En los ejemplos siguientes se muestran casos de *true* y *false* tanto para comparaciones numéricas como de cadenas:

```
5 <= 10;  
// true
```

```
2 <= 2;  
// true
```

```
10 <= 3;  
// false
```

```
"Allen" <= "Jack";  
// true
```

```
"Jack" <= "Allen";
```



```
// false

"11" <= "3";
// true

"11" <= 3;
// comparación numérica
// false

"C" <= "abc";
// false

"A" <= "a";
// true
```

## ⌵ (desigualdad)

### Disponibilidad

Flash 2.

### Sintaxis

*expression1* ⌵ *expression2*

### Parámetros

*expression1*, *expression2* Número, cadena, valor booleano, variable, objeto, matriz o función.

### Valor devuelto

Valor booleano.

### Descripción

Operador (desigualdad); comprueba exactamente lo contrario que el operador ==. Si *expression1* es igual a *expression2*, el resultado es *false*. Al igual que con el operador ==, la definición de *igual* depende de los tipos de datos que se comparan.

- Los números, cadenas y valores booleanos se comparan por valor.
- Las variables, objetos, matrices y funciones se comparan por referencia.

Este operador se eliminó en Flash 5 y Macromedia recomienda utilizar el operador !=.

### Véase también

[!= \(desigualdad\)](#)

## = (asignación)

### Disponibilidad

Flash Player 4.

Archivo de Flash 4:

```
x = y
```

Archivo de Flash 5 o posterior convertido:

```
Number(x) == Number(y)
```

### Sintaxis

*expression1* = *expression2*

### Parámetros

*expression1* Variable, elemento de una matriz o propiedad de un objeto.

*expression2* Valor de cualquier tipo.

### Valor devuelto

Ninguno.

### Descripción

Operador; asigna el tipo de *expression2* (el parámetro de la derecha) a la variable, elemento de matriz o propiedad de *expression1*.

En Flash 5, = es un operador de asignación y el operador == se utiliza para comprobar la igualdad. En Flash 4, = es un operador de igualdad numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos.

### Ejemplo

En el ejemplo siguiente se utiliza el operador de asignación para asignar el tipo de datos Number a la variable x.

```
x = 5
```

En el ejemplo siguiente se utiliza el operador de asignación para asignar el tipo de datos String a la variable x.

```
x = "hola"
```

### Véase también

[== \(igualdad\)](#)

## -= (asignación de resta)

### Disponibilidad

Flash Player 4.

### Sintaxis

*expression1* -= *expression2*

### Parámetros

*expression1*, *expression2* Número o expresión que da como resultado un número.

### Valor devuelto

Ninguno.

## Descripción

Operador (asignación compuesta aritmética); asigna a *expression1* el valor de *expression1* - *expression2*. Por ejemplo, las dos sentencias siguientes son iguales:

```
x -= y;  
x = x - y;
```

Las expresiones de cadena deben convertirse en números o se devolverá NaN.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se utiliza el operador -= para restar 10 de 5 y asignar el resultado a la variable x.

```
x = 5;  
y = 10;  
x -= y  
trace(x);  
// devuelve -5
```

Sintaxis 2: en el ejemplo siguiente se muestra cómo las cadenas se convierten en números.

```
x = "5";  
y = "10";  
x -= y;  
trace(x);  
// devuelve -5
```

## == (igualdad)

### Disponibilidad

Flash Player 5.

### Sintaxis

```
expression1 == expression2
```

### Parámetros

*expression1*, *expression2* Número, cadena, valor booleano, variable, objeto, matriz o función.

### Valor devuelto

Valor booleano.

### Descripción

Operador (de igualdad); comprueba si dos expresiones son iguales. El resultado es `true` si las expresiones son iguales.

La definición de *igual* depende del tipo de datos del parámetro:

- Los números y los valores booleanos se comparan por valor y se consideran iguales si tienen el mismo valor.
- Las expresiones de cadena son iguales si tienen el mismo número de caracteres y los caracteres son idénticos.
- Las variables, objetos, matrices y funciones se comparan por referencia. Dos variables son iguales si se refieren al mismo objeto, matriz o función. Dos matrices independientes nunca se consideran iguales, incluso aunque tengan el mismo número de elementos.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se utiliza el operador `==` con una sentencia `if`:

```
a = "David" , b = "David";
if (a == b){
    trace("David es David");
}
```

Sintaxis 2: en estos ejemplos se muestran los resultados de operaciones que comparan tipos mixtos.

```
x = "5"; y = "5";
trace(x == y);
// true
```

```
x = "5"; y = "66";
trace(x == y);
// false
```

```
x = "chris"; y = "steve";
trace(x == y);
// false
```

## Véase también

`!=` (desigualdad), `===` (igualdad estricta), `!==` (desigualdad estricta)

## === (igualdad estricta)

### Disponibilidad

Flash Player 6.

### Sintaxis

```
expression1 === expression2
```

### Valor devuelto

Valor booleano.

### Descripción

Operador; comprueba la igualdad de dos expresiones; el operador de igualdad estricta funciona como el operador de igualdad, salvo que los tipos de datos no se convierten. El resultado es `true` si las dos expresiones, incluidos los tipos de datos, son iguales.

La definición de *igual* depende del tipo de datos del parámetro:

- Los números y los valores booleanos se comparan por valor y se consideran iguales si tienen el mismo valor.
- Las expresiones de cadena son iguales si tienen el mismo número de caracteres y los caracteres son idénticos.
- Las variables, objetos, matrices y funciones se comparan por referencia. Dos variables son iguales si se refieren al mismo objeto, matriz o función. Dos matrices independientes nunca se consideran iguales, incluso aunque tengan el mismo número de elementos.

## Ejemplo

En el código siguiente se muestra el valor devuelto de las operaciones que utilizan operadores de igualdad, de igualdad estricta y de desigualdad estricta.

```
s1 = new String("5");
s2 = new String("5");
s3 = new String("Hola");
n = new Number(5);
b = new Boolean(true);
```

```
s1 == s2; // true
s1 == s3; // false
s1 == n; // true
s1 == b; // false
```

```
s1 === s2; // true
s1 === s3; // false
s1 === n; // false
s1 === b; // false
```

```
s1 !== s2; // false
s1 !== s3; // true
s1 !== n; // true
s1 !== b; // true
```

## Véase también

`==` (igualdad), `!=` (desigualdad), `===` (igualdad estricta)

## > (mayor que)

### Disponibilidad

Flash Player 4.

Archivo de Flash 4:

`x > y`

Archivo de Flash 5 o posterior convertido:

`Number(x) > Number(y)`

### Sintaxis

*expression1* > *expression2*

### Parámetros

*expression1*, *expression2* Número o cadena.

### Valor devuelto

Valor booleano.

### Descripción

Operador (comparación); compara dos expresiones y determina si *expression1* es mayor que *expression2*; si lo es, el operador devuelve *true*. Si *expression1* es menor o igual que *expression2*, el operador devuelve *false*. Las expresiones de cadena se comparan en orden alfabético; las letras mayúsculas van antes que las minúsculas.

En Flash 5, el operador menor o igual que ( $\leq$ ) es un operador de comparación que puede manejar varios tipos de datos. En Flash 4,  $\leq$  sólo es un operador numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos.

## **>= (mayor o igual que)**

### **Disponibilidad**

Flash Player 4.

Archivo de Flash 4:

$x > y$

Archivo de Flash 5 o posterior convertido:

`Number(x) > Number(y)`

### **Sintaxis**

*expression1* >= *expression2*

### **Parámetros**

*expression1*, *expression2* Cadena, entero o número de coma flotante.

### **Valor devuelto**

Valor booleano.

### **Descripción**

Operador (comparación); compara dos expresiones y determina si *expression1* es mayor o igual que *expression2* (true) o si *expression1* es menor que *expression2* (false).

En Flash 5 o versiones posteriores, el operador mayor o igual que ( $\geq$ ) es un operador de comparación que puede gestionar distintos tipos de datos. En Flash 4,  $\geq$  es un operador numérico. Los archivos de Flash 4 que se abren en un entorno de edición de Flash 5 sufren un proceso de conversión para mantener la integridad de los tipos de datos.

## **>> (desplazamiento a la derecha en modo bit)**

### **Disponibilidad**

Flash Player 5.

### **Sintaxis**

*expression1* >> *expression2*

### **Parámetros**

*expression1* Número, cadena o expresión que se va a desplazar a la derecha.

*expression2* Número o expresión que se convierte en un entero entre 0 y 31.

### **Valor devuelto**

Ninguno.

## Descripción

Operador (en modo bit); convierte *expression1* y *expression2* en números enteros de 32 bits y desplaza todos los bits de *expression1* hacia la derecha el número de espacios especificado por el número entero resultante de la conversión de *expression2*. Los bits que se desplazan a la derecha se descartan. Para conservar el signo de la *expression* original, los bits situados a la izquierda se rellenan con 0 si el bit más significativo (el bit del extremo izquierdo) de *expression1* es 0, y se rellena con 1 si el bit más significativo es 1. Desplazar un valor a la derecha una posición es equivalente a dividirlo entre 2 y descartar el resto.

## Ejemplo

En el ejemplo siguiente, 65535 se convierte en un número entero de 32 bits y se desplaza ocho bits a la derecha.

```
x = 65535 >> 8
```

El resultado de la operación anterior es el siguiente:

```
x = 255
```

Esto es debido a que el decimal 65535 es igual al binario 1111111111111111 (dieciséis unos), el binario 1111111111111111 desplazado a la derecha ocho bits es el binario 11111111 y el binario 11111111 es el decimal 255 . El bit más significativo es 0 debido a que los números enteros son de 32 bits, así que el bit de relleno es 0.

En el ejemplo siguiente -1 se convierte en un número entero de 32 bits y se desplaza un bit a la derecha.

```
x = -1 >> 1
```

El resultado de la operación anterior es el siguiente:

```
x = -1
```

Esto se debe a que el decimal -1 es igual al binario 11111111111111111111111111111111 (treinta y dos unos); si se desplaza a la derecha un bit, el bit menos significativo (el bit del extremo derecho) se descartará y el bit más significativo se rellenará con 1. El resultado es el binario 11111111111111111111111111111111 (treinta y dos unos), lo que representa el número entero -1 de 32 bits.

## Véase también

[>= \(desplazamiento a la derecha en modo bit y asignación\)](#)

## >>= (desplazamiento a la derecha en modo bit y asignación)

### Disponibilidad

Flash Player 5.

### Sintaxis

```
expression1 ==>>expression2
```

### Parámetros

*expression1*    Número o expresión que se va a desplazar a la izquierda.

*expression2*    Número o expresión que se convierte en un entero entre 0 y 31.

### Valor devuelto

Ninguno.

### Descripción

Operador (de asignación compuesta en modo bit); este operador realiza una operación de desplazamiento a la derecha en modo bit y almacena el contenido como resultado en *expression1*.

### Ejemplo

Las dos expresiones siguientes son equivalentes.

```
A >>= B  
A = (A >> B)
```

En el código comentado siguiente se utiliza el operador (>>=) en modo bit. También es un ejemplo de la utilización de todos los operadores en modo bit.

```
function convertToBinary(number){  
    var result = "";  
    for (var i=0; i<32; i++) {  
        // Extraer el bit menos significativo con AND en modo bit  
        var lsb = number & 1;  
        // Agregar este bit a nuestra cadena de resultado  
        result = (lsb ? "1" : "0") + result;  
        // Desplazar el número un bit a la derecha para ver el bit siguiente  
        number >>= 1;}  
    return result;  
}  
trace(convertToBinary(479));  
// Devuelve la cadena 000000000000000000000000011101111  
// La cadena anterior es la representación binaria del número decimal  
// 479
```

### Véase también

<< (desplazamiento a la izquierda en modo bit)

## >>> (desplazamiento a la derecha en modo bit sin signo)

### Disponibilidad

Flash Player 5.

### Sintaxis

```
expression1 >>> expression2
```

### Parámetros

*expression1* Número, cadena o expresión que se va a desplazar a la derecha.

*expression2* Número, cadena o expresión que se convierte en un entero entre 0 y 31.

### Valor devuelto

Ninguno.



## Descripción

Operador (en modo bit); es lo mismo que el operador de desplazamiento a la derecha en modo bit ( $\gg$ ) excepto en que no conserva el signo de *expression* original debido a que los bits de la izquierda siempre se rellenan con 0.

## Ejemplo

En el ejemplo siguiente -1 se convierte en un número entero de 32 bits y se desplaza un bit a la derecha.

```
x = -1 >>> 1
```

El resultado de la operación anterior es el siguiente:

```
x = 2147483647
```

Esto es debido a que -1 decimal es el binario 11111111111111111111111111111111 (treinta y dos unos) y cuando se desplaza a la derecha (sin signo) un bit, el bit menos significativo (el situado más a la derecha) se descarta y el bit más significativo (el situado más a la izquierda) se rellena con un 0. El resultado es el binario 01111111111111111111111111111111, que representa el entero de 32 bits 2147483647.

## Véase también

[>>= \(desplazamiento a la derecha en modo bit y asignación\)](#)

# >>>= (desplazamiento a la derecha en modo bit sin signo y asignación)

## Disponibilidad

Flash Player 5.

## Sintaxis

```
expression1 >>>= expression2
```

## Parámetros

*expression1* Número o expresión que se va a desplazar a la izquierda.

*expression2* Número o expresión que se convierte en un entero entre 0 y 31.

## Valor devuelto

Ninguno.

## Descripción

Operador (de asignación compuesta en modo bit); este operador realiza una operación en modo bit de desplazamiento a la derecha sin signo y almacena el contenido como resultado en *expression1*. Las dos expresiones siguientes son equivalentes:

```
A >>>= B  
A = (A >>> B)
```

## Véase también

[>>> \(desplazamiento a la derecha en modo bit sin signo\)](#), [>>= \(desplazamiento a la derecha en modo bit y asignación\)](#)

# Clase Accessibility

## Disponibilidad

Flash Player 6 versión 65.

## Descripción

La clase Accessibility gestiona la comunicación con los lectores de pantalla. Los métodos de la clase Accessibility son estáticos, es decir, no es necesario crear una instancia de la clase para utilizarlos.

Para obtener y establecer las propiedades accesibles para un objeto específico, como por ejemplo un botón, un clip de película o un campo de texto, utilice la propiedad `_accProps`. Para determinar si el reproductor se está ejecutando en un entorno que admite ayudas de accesibilidad, use `System.capabilities.hasAccessibility`.

## Resumen de métodos para la clase Accessibility

Método	Descripción
<code>Accessibility.isActive()</code>	Indica si hay activo un programa de lectura en pantalla.
<code>Accessibility.updateProperties()</code>	Actualiza la descripción de los objetos de la pantalla para los lectores de pantalla.

## Accessibility.isActive()

### Disponibilidad

Flash Player 6 versión 65.

### Sintaxis

```
Accessibility.isActive()
```

### Parámetros

Ninguno.

### Valor devuelto

Un valor booleano de `true` si hay clientes de Microsoft Active Accessibility (MSAA) activos y el reproductor se está ejecutando en un entorno que admite comunicación entre Flash Player y las ayudas de accesibilidad; `false` en caso contrario.

### Descripción

Método: indica si hay un programa de lector de pantallas MSAA activo y el reproductor se está ejecutando en un entorno que admite comunicación entre Flash Player y las ayudas de accesibilidad. Utilice este método cuando desee que la aplicación tenga un comportamiento diferente en presencia de un lector de pantalla.

Para determinar si el reproductor se está ejecutando en un entorno que admite ayudas de accesibilidad, use `System.capabilities.hasAccessibility`.

**Nota:** si llama a este método transcurridos uno o dos segundos después de que aparezca por primera vez la ventana Flash en la que se está reproduciendo el archivo, es posible que obtenga un valor `false` aun cuando exista un cliente MSAA activo. Esto es debido a que el mecanismo de comunicación entre Flash y los clientes MSAA es asíncrono. Para evitar esta limitación, asegúrese de que transcurren uno o dos segundos entre la carga del documento y la llamada a este método.

### Véase también

[Accessibility.updateProperties\(\)](#), [\\_accProps](#),  
[System.capabilities.hasAccessibility](#)

## Accessibility.updateProperties()

### Disponibilidad

Flash Player 6 versión 65.

### Sintaxis

`Accessibility.updateProperties()`

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; hace que Flash Player vuelva a examinar todas las propiedades de accesibilidad, actualice la descripción de los objetos para los lectores de pantalla y, si es necesario, envíe eventos a los lectores de pantalla para indicar que se han producido cambios. Para más información sobre configuración de propiedades de accesibilidad, consulte [\\_accProps](#).

Para determinar si el reproductor se está ejecutando en un entorno que admite ayudas de accesibilidad, use [System.capabilities.hasAccessibility](#).

Si modifica las propiedades de accesibilidad para varios objetos, sólo es necesario que llame a la función `Accessibility.updateProperties()` una vez; varias llamadas a esta función podrían afectar negativamente al rendimiento y generar resultados incomprensibles en el lector de pantalla.

### Ejemplo

El código de ActionScript siguiente se beneficia de las propiedades de accesibilidad dinámicas. Este ejemplo corresponde a un botón sin texto que puede cambiar el icono que se muestra.

```
function setIcon( newIconNum, newTextEquivalent )
{
    this.iconImage = this.iconImages[ newIconNum ];
    if ( newTextEquivalent != undefined )
    {
        if ( this._accProps == undefined )
            this._accProps = new Object();
        this._accProps.name = newTextEquivalent;
        Accessibility.updateProperties();
    }
}
```

**Véase también**

`Accessibility.isActive(), _accProps, System.capabilities.hasAccessibility`

# **\_accProps**

**Disponibilidad**

Flash Player 6 versión 65.

**Sintaxis**

`_accProps.propertyName`  
`instanceName._accProps.propertyName`

**Parámetros**

*propertyName* Nombre de propiedad de accesibilidad (en la descripción siguiente se indican los nombres válidos).  
*instanceName* Nombre de instancia asignado a una instancia de un clip de película, un botón, un campo de texto dinámico o un campo de introducción de texto.

**Descripción**

Propiedad; permite controlar las opciones de accesibilidad del lector de pantalla para archivos SWF, clips de película, botones, campos de texto dinámico y campos de introducción de texto en tiempo de ejecución. Estas propiedades suplantando las correspondientes opciones disponibles en el panel Accesibilidad durante la edición. Para que tengan efecto los cambios sobre estas propiedades, debe llamar a `Accessibility.updateProperties()`. Para más información sobre el panel Accesibilidad, consulte “Introducción al panel Accesibilidad de Flash” en el apartado Utilización de Flash de la Ayuda.

Para determinar si el reproductor se está ejecutando en un entorno que admite ayudas de accesibilidad, use `System.capabilities.hasAccessibility`.

En la tabla siguiente se muestra una lista con los nombres y tipos de datos de cada propiedad `_accProps`, su configuración equivalente en el panel Accesibilidad, y los tipos de objetos a los que se puede aplicar la propiedad. El término *lógica inversa* significa que la configuración de la propiedad es la inversa de la configuración correspondiente en el panel Accesibilidad. Por ejemplo, si establece la propiedad `silent` en `true` equivaldrá a deseleccionar las opciones Permitir acceso a la película o Hacer que el objeto sea accesible.

Propiedad	Tipo de datos	Equivalente en el panel Accesibilidad	Se aplica a
<code>silent</code>	Booleano	Permitir acceso a la película/ Hacer que el objeto sea accesible ( <i>lógica inversa</i> )	Películas enteras Clips de película Botones Texto dinámico Introducción de texto
<code>forceSimple</code>	Booleano	Hacer que los objetos secundarios sean accesibles ( <i>lógica inversa</i> )	Películas enteras Clips de película

Propiedad	Tipo de datos	Equivalente en el panel Accesibilidad	Se aplica a
name	Cadena	Nombre	Películas enteras Clips de película Botones Introducción de texto
description	Cadena	Descripción	Películas enteras Clips de película Botones Texto dinámico Introducción de texto
shortcut	Cadena	Métodos abreviados*	Clips de película Botones Introducción de texto

\* Para más información sobre cómo asignar un método abreviado de teclado a un objeto accesible, consulte [Key.addListener\(\)](#).

Para especificar una configuración que corresponda al valor de índice de tabulación del panel Accesibilidad, utilice las propiedades [Button.tabIndex](#), [MovieClip.tabIndex](#) o [TextField.tabIndex](#).

Es imposible especificar un valor de Etiquetado automático en tiempo de ejecución.

Cuando se utiliza sin el parámetro *instanceName*, los cambios realizados en las propiedades `_accProps` se aplican a toda la película. Por ejemplo, la línea de código siguiente establece la propiedad `name` para toda la película en la cadena "Tienda de mascotas" y luego llama a `Accessibility.updateProperties()` para que el cambio surta efecto.

```
_accProps.name = "Tienda de mascotas";
Accessibility.updateProperties();
```

Por el contrario, el código siguiente establece la propiedad `name` para un clip de película con el nombre de instancia `price_mc` en la cadena "Precio":

```
price_mc._accProps.name = "Precio";
Accessibility.updateProperties();
```

Si especifica múltiples propiedades de accesibilidad, efectúe todos los cambios que pueda antes de llamar a `Accessibility.updateProperties()`, en lugar de llamarla tras cada sentencia de propiedad:

```
_accProps.name = "Tienda de mascotas";
animal_mc._accProps.name = "Animal";
animal_mc._accProps.description = "Gato, perro, pez, etc.";
price_mc._accProps.name = "Precio";
price_mc._accProps.description = "Coste de un único artículo";
Accessibility.updateProperties();
```

Si no especifica un valor para la propiedad de accesibilidad de una película u objeto, se implementarán los valores establecidos en el panel Accesibilidad.

Tras especificar una propiedad de accesibilidad, no podrá devolver su valor al que se haya establecido en el panel Accesibilidad. No obstante, puede devolver a la propiedad su valor predeterminado (`false` para valores booleanos, cadenas vacías para valores de cadena) si elimina el objeto `_accProps`:

```
my_mc._accProps.silent = true; // establecer una propiedad
// aquí otro código
delete my_mc._accProps.silent; // volver al valor predeterminado
```

Para devolver todos los valores de accesibilidad de un objeto a sus valores predeterminados, puede eliminar el objeto `instanceName._accProps`:

```
delete my_btn._accProps;
```

Para devolver todos los valores de accesibilidad de todos los objetos a sus valores predeterminados, puede eliminar el objeto global `_accProps`:

```
delete _accProps;
```

Si especifica una propiedad para un tipo de objeto que no la admite, se omite la asignación de propiedad y no se muestra ningún error. Por ejemplo, los botones no admiten la propiedad `forceSimple`, de manera que se omitirá una línea como la siguiente:

```
my_btn._accProps.forceSimple = false; //omitido
```

## Ejemplo

A continuación se muestra código de ActionScript de ejemplo que se beneficia de las propiedades dinámicas de accesibilidad. Podría asignar este código a un componente de botón de icono no textual que puede cambiar el icono que se muestra.

```
function setIcon( newIconNum, newTextEquivalent )
{
    this.iconImage = this.iconImages[ newIconNum ];
    if ( newTextEquivalent != undefined )
    {
        if ( this._accProps == undefined )
            this._accProps = new Object();
        this._accProps.name = newTextEquivalent;
        Accessibility.updateProperties();
    }
}
```

## Véase también

[Accessibility.isActive\(\)](#), [Accessibility.updateProperties\(\)](#),  
[System.capabilities.hasAccessibility](#)

# add

## Disponibilidad

Flash Player 4.

## Sintaxis

```
string1 add string2
```

## Parámetros

*string1*, *string2* Una cadena.

### Valor devuelto

Ninguno.

### Descripción

Operador; concatena (combina) dos o más cadenas. El operador `add` sustituye al operador `add (&)` de Flash 4; cuando los archivos de Flash Player 4 que utilizan el operador `&` se utilizan en un entorno de edición Flash 5 o posterior, se convierten automáticamente para utilizar el operador `add` para la concatenación de cadenas. No obstante, el operador `add` se ha eliminado de Flash Player 5, y Macromedia recomienda utilizar el operador `+` cuando se crea contenido para Flash Player 5 o una versión posterior. Utilice el operador `add` para concatenar cadenas si está creando contenido para Flash Player 4 o para versiones anteriores del reproductor.

### Véase también

[+ \(suma\)](#)

## and

### Disponibilidad

Flash Player 4.

### Sintaxis

*condition1 and condition2*

### Parámetros

*condition1, condition2* Condiciones o expresiones que dan como resultado `true` o `false`.

### Valor devuelto

Ninguno.

### Descripción

Operador; realiza una operación AND lógica en Flash Player 4. Si ambas expresiones dan como resultado `true`, toda la expresión es `true`. Este operador se eliminó en la versión Flash 5 y Macromedia recomienda utilizar el operador `&&`.

### Véase también

[&& \(AND lógico\)](#)

## Clase Arguments

### Disponibilidad

Flash Player 5; propiedad añadida en Flash Player 6.

### Descripción

La clase `Arguments` es una matriz que contiene los valores que se han pasado como parámetros a cualquier función. Cada vez que se llama a una función en `ActionScript`, se crea de forma automática un objeto `Arguments` para esa función. Se crea también una variable local, `arguments`, que permite hacer referencia al objeto `Arguments`.

## Resumen de propiedades para la clase Arguments

Propiedad	Descripción
<code>arguments.callee</code>	Hace referencia a la función a la que se está llamando.
<code>arguments.caller</code>	Hace referencia a la función que llama.
<code>arguments.length</code>	El número de parámetros pasados a una función.

### arguments.callee

#### Disponibilidad

Flash Player 5.

#### Sintaxis

`arguments.callee`

#### Descripción

Propiedad; hace referencia a la función a la que se está llamando en este momento.

#### Ejemplo

Puede utilizar la propiedad `arguments.callee` para realizar una función anónima que es recursiva, como se muestra a continuación:

```
factorial = function (x) {  
    if (x <= 1) {  
        return 1;  
    } else {  
        return x * arguments.callee(x-1);  
    }  
};
```

A continuación se muestra una función recursiva con nombre:

```
function factorial (x) {  
    if (x <= 1) {  
        return 1;  
    } else {  
        return x * factorial(x-1);  
    }  
}
```

### arguments.caller

#### Disponibilidad

Flash Player 6.

#### Sintaxis

`arguments.caller`

#### Descripción

Propiedad; hace referencia a la función que llama.



## arguments.length

### Disponibilidad

Flash Player 5.

### Sintaxis

```
arguments.length
```

### Descripción

Propiedad; el número de parámetros que se han pasado realmente a una función.

## Clase Array

### Disponibilidad

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).

### Descripción

La clase Array permite acceder a matrices y manipularlas. Una matriz es un objeto cuyas propiedades se identifican con un número que representa su posición en la matriz. Este número se denomina *índice*. Todas las matrices tienen base cero, lo que quiere decir que el primer elemento de la matriz es [0], el segundo elemento es [1] y así sucesivamente. En el ejemplo siguiente, my\_array contiene los meses del año.

```
my_array[0] = "Enero"
my_array[1] = "Febrero"
my_array[2] = "Marzo"
my_array[3] = "Abril"
```

Para crear un objeto Array, utilice el constructor `new Array()` o el operador de acceso a matriz (`[]`). Para acceder a los elementos de una matriz, utilice el operador de acceso a matriz (`[]`).

## Resumen de métodos para la clase Array

Método	Descripción
<code>Array.concat()</code>	Concatena los parámetros y los devuelve como una matriz nueva.
<code>Array.join()</code>	Une todos los elementos de una matriz en una cadena.
<code>Array.pop()</code>	Elimina el último elemento de una matriz y devuelve su valor.
<code>Array.push()</code>	Agrega uno o más elementos al final de una matriz y devuelve la nueva longitud de la matriz.
<code>Array.reverse()</code>	Invierte la dirección de una matriz.
<code>Array.shift()</code>	Elimina el primer elemento de una matriz y devuelve su valor.
<code>Array.slice()</code>	Extrae una sección de una matriz y la devuelve como una nueva matriz.
<code>Array.sort()</code>	Ordena una matriz.
<code>Array.sortOn()</code>	Ordena una matriz según un campo de la matriz.
<code>Array.splice()</code>	Agrega y elimina elementos de una matriz.

Método	Descripción
<code>Array.toString()</code>	Devuelve un valor de cadena que representa los elementos del objeto Array.
<code>Array.unshift()</code>	Agrega uno o más elementos al principio de una matriz y devuelve la nueva longitud de la matriz.

## Resumen de propiedades para la clase Array

Propiedad	Descripción
<code>Array.length</code>	Número entero que no es de base cero y que especifica el número de elementos de la matriz.

## Constructor para la clase Array

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Array()
new Array(length)
new Array(element0, element1, element2,...elementN)
```

### Parámetros

*length* Número entero que especifica el número de elementos de la matriz. En el caso de elementos no contiguos, el parámetro *length* especifica el número de índice del último elemento de la matriz más 1.

*element0...elementN* Lista de dos o más valores arbitrarios. Los valores pueden ser números, cadenas, objetos u otras matrices. El primer elemento de una matriz siempre tiene el índice o posición 0.

### Valor devuelto

Ninguno.

### Descripción

Constructor; permite crear una matriz. Puede utilizar el constructor para crear distintos tipos de matrices: una matriz vacía, una matriz con una longitud específica pero cuyos elementos no tienen valores, o una matriz cuyos elementos tienen valores específicos.

Sintaxis 1: si no especifica parámetros, se crea una matriz con una longitud 0.

Sintaxis 2: si sólo especifica una longitud, se crea una matriz con *length* como número de elementos, sin valores.

Sintaxis 3: si utiliza los parámetros *element* para especificar valores, se crea una matriz con valores específicos.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se crea un nuevo objeto Array con una longitud inicial de 0.

```
my_array = new Array();  
trace(my_array.length); // devuelve 0
```

Sintaxis 2: en el ejemplo siguiente se crea un nuevo objeto Array con una longitud inicial de 4.

```
my_array = new Array(4);  
trace(my_array.length); // devuelve 4
```

Sintaxis 3: en el ejemplo siguiente se crea el nuevo objeto Array `go_gos_array`, con una longitud inicial de 5.

```
go_gos_array = new Array("Belinda", "Gina", "Kathy", "Charlotte", "Jane");  
trace(my_array.length); // devuelve 5  
trace(go_gos_array.join(", ")); // muestra elementos
```

Los elementos iniciales de la matriz `go_gos` se identifican como se muestra a continuación:

```
go_gos_array[0] = "Belinda";  
go_gos_array[1] = "Gina";  
go_gos_array[2] = "Kathy";  
go_gos_array[3] = "Charlotte";  
go_gos_array[4] = "Jane";
```

El código siguiente añade un sexto elemento a la matriz `go_gos_array` y cambia el segundo elemento:

```
go_gos_array[5] = "Donna";  
go_gos_array[1] = "Nina";  
trace(go_gos_array.join(" + "));
```

## Véase también

[Array.length, \[\] \(acceso a matriz\)](#)

## Array.concat()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.concat( [ value0,value1,...valueN ])
```

### Parámetros

*value0,...valueN* Números, elementos o cadenas que se van a concatenar en una nueva matriz. Si no se pasa ningún valor, se creará un duplicado de *my\_array*.

### Valor devuelto

Ninguno.

### Descripción

Método; concatena los elementos especificados en los parámetros con los elementos de *my\_array* y crea una matriz nueva. Si los parámetros *value* especifican una matriz, se concatenan los elementos de esa matriz en lugar de la propia matriz. La matriz *my\_array* se deja intacta.

## Ejemplo

En el código siguiente se concatenan dos matrices:

```
alpha_array = new Array("a","b","c");
numeric_array = new Array(1,2,3);
alphaNumeric_array=alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// crea la matriz ["a","b","c",1,2,3]
```

En el código siguiente se concatenan tres matrices:

```
num1_array = [1,3,5];
num2_array = [2,4,6];
num3_array = [7,8,9];
nums_array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// crea la matriz [1,3,5,2,4,6,7,8,9]
```

Las matrices anidadas no se despliegan del mismo modo que las matrices normales. Los elementos de una matriz anidada no se desglosan en elementos independientes de una matriz `x_array`, como se muestra en el ejemplo siguiente.

```
a_array = new Array ("a","b","c");

// 2 y 3 son elementos de una matriz anidada
n_array = new Array(1, [2, 3], 4);

x_array = a_array.concat(n_array);
trace(x_array[0]); // "a"
trace(x_array[1]); // "b"
trace(x_array[2]); // "c"
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

## Array.join()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.join([separator])
```

### Parámetros

*separator* Carácter o cadena que separa los elementos de la matriz en la cadena devuelta. Si omite este parámetro, se utiliza la coma como separador predeterminado.

### Valor devuelto

Una cadena.

### Descripción

Método; convierte los elementos de una matriz en cadenas, inserta el separador especificado entre los elementos, los concatena y devuelve la cadena resultante. Las matrices anidadas siempre se separan con una coma y no con el separador pasado al método `join()`.

## Ejemplo

El ejemplo siguiente crea una matriz, con tres elementos: Tierra, Luna y Sol. A continuación, une la matriz tres veces: primero utilizando el separador predeterminado (una coma y un espacio), después utilizando un guión y, por último, utilizando un signo más (+), y las muestra en el panel Salida:

```
a_array = new Array("Tierra","Luna","Sol")
trace(a_array.join());
// devuelve tierra, luna, sol
trace(a_array.join(" - "));
// devuelve tierra - luna - sol
trace(a_array.join(" + "));
// devuelve tierra + luna + sol
```

## Array.length

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.length
```

### Descripción

Propiedad; número entero que no es de base cero y que especifica el número de elementos de la matriz. Esta propiedad se actualiza automáticamente cuando se agregan nuevos elementos a la matriz. Cuando se asigna un valor a un elemento de matriz (por ejemplo, `my_array[index] = valor`), si `index` es un número e `index+1` es mayor que la propiedad `length`, la propiedad `length` se actualiza a `index+1`.

## Ejemplo

En el código siguiente se explica cómo se actualiza la propiedad `length`.

```
my_array = new Array();
trace(my_array.length); // la longitud inicial es 0
my_array[0] = 'a';
trace(my_array.length); // my_array.length se actualiza a 1
my_array[1] = 'b';
trace(my_array.length); // my_array.length se actualiza a 2
my_array[9] = 'c';
trace(my_array.length); // my_array.length se actualiza a 10
```

## Array.pop()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.pop()
```

### Parámetros

Ninguno.

### Valor devuelto

El valor del último elemento de la matriz especificada.

### Descripción

Método; elimina el último elemento de una matriz y devuelve el valor de ese elemento.

### Ejemplo

En el código siguiente se crea la matriz `myPets` que contiene cuatro elementos y después se elimina su último elemento.

```
myPets = ["gato", "perro", "pájaro", "pez"];
popped = myPets.pop();
trace(popped);
// devuelve pez
```

## Array.push()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.push(value,...)
```

### Parámetros

*value* Uno o más valores que se anexan a la matriz.

### Valor devuelto

La longitud de la nueva matriz.

### Descripción

Método; añade uno o más elementos al final de una matriz y devuelve la nueva longitud de la matriz.

### Ejemplo

En el ejemplo siguiente se crea la matriz `myPets` con dos elementos, gato y perro. En la segunda línea se agregan dos elementos a la matriz. Después de llamar al método `push()`, la variable `pushed` contiene cuatro elementos. Puesto que el método `push()` devuelve la nueva longitud de la matriz, la acción `trace()` de la última línea envía la nueva longitud de `myPets` (4) al panel Salida:

```
myPets = ["gato", "perro"];
pushed = myPets.push("pájaro", "pez");
trace(pushes);
```

## Array.reverse()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.reverse()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Método; invierte la matriz.

**Ejemplo**

A continuación se muestra un ejemplo del uso de este método.

```
var numbers_array = [1, 2, 3, 4, 5, 6];  
trace(numbers_array.join()); //1,2,3,4,5,6  
numbers_array.reverse();  
trace(numbers_array.join()); // 6,5,4,3,2,1
```

## Array.shift()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_array.shift()
```

**Parámetros**

Ninguno.

**Valor devuelto**

El primer elemento de una matriz.

**Descripción**

Método; elimina el primer elemento de una matriz y devuelve ese elemento.

**Ejemplo**

En el código siguiente se crea la matriz `myPets` y, a continuación, se elimina el primer elemento de la matriz y se asigna a la variable `shifted`.

```
var myPets_array = ["gato", "perro", "pájaro", "pez"];  
shifted = myPets_array.shift();  
trace(shifted); // devuelve "gato"
```

**Véase también**

[Array.pop\(\)](#)

## Array.slice()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.slice( [ start [ , end ] ] )
```

### Parámetros

*start* Número que especifica el índice del punto de inicio de la sección. Si *start* es un número negativo, el punto de inicio comienza al final de la matriz, donde -1 es el último elemento.

*end* Número que especifica el índice del punto final de la sección. Si omite este parámetro, la sección incluirá todos los elementos desde el inicio hasta el final de la matriz. Si *end* es un número negativo, el punto final se especifica desde el final de la matriz, donde -1 es el último elemento.

### Valor devuelto

Una matriz.

### Descripción

Método; extrae una sección o una subcadena de la matriz y la devuelve como una nueva matriz sin modificar la matriz original. La matriz devuelta incluye el elemento *start* y todos los elementos hasta el elemento *end*, pero sin incluir este último.

Si no se pasa ningún parámetro, se creará un duplicado de *my\_array*.

## Array.sort()

### Disponibilidad

Flash Player 5; se han añadido funciones adicionales en Flash Player 7.

### Sintaxis

```
my_array.sort()  
my_array.sort(compareFunction)  
my_array.sort(option | option |... )  
my_array.sort(compareFunction, option | option |... )
```

### Parámetros

*compareFunction* Función de comparación opcional que se utiliza para determinar el orden de los elementos de una matriz. Dados los elementos A y B, el resultado de *compareFunction* puede tener uno de los tres valores siguientes:

- -1 si A aparece antes que B en la secuencia de ordenamiento
- 0 si A = B
- 1 si A aparece antes que B en la secuencia de ordenamiento



*option* Uno o más números o cadenas, separados por el operador | (OR en modo bit), que cambian el comportamiento o la clasificación respecto al predeterminado. Los siguientes valores de *option* son aceptables:

- 1 o `Array.CASEINSENSITIVE`
- 2 o `Array.DESENDING`
- 4 o `Array.UNIQUE`
- 8 o `Array.RETURNINDEXEDARRAY`
- 16 o `Array.NUMERIC`

Para más información sobre este parámetro, consulte [Array.sortOn\(\)](#).

### Valor devuelto

El valor devuelto depende de si se pasa algún parámetro:

- Si especifica un valor de 4 o `Array.UNIQUE` como *option* y dos o más de los elementos que se están ordenando tienen campos de clasificación idénticos, Flash devuelve un valor de 0 y no modifica la matriz.
- Si especifica un valor de `Array.RETURNINDEXEDARRAY` como *option*, Flash devuelve una matriz que refleja el resultado de la clasificación y no modifica la matriz.
- De lo contrario, Flash no devuelve nada y modifica la matriz de manera que refleje el orden de clasificación.

### Descripción

Método; ordena los elementos de una matriz. Flash ordena siguiendo valores ASCII (Unicode). Si alguno de los elementos que se están ordenando no contiene el campo especificado en el parámetro *fieldName*, se interpreta el campo como *undefined*, y los elementos se sitúan de manera consecutiva en la matriz sin ningún orden en particular.

De forma predeterminada, `Array.sort()` funciona de esta manera:

- La clasificación distingue entre mayúsculas y minúsculas (*Z* va antes que *a*).
- La clasificación es ascendente (*a* va antes que *b*).
- La matriz se modifica de manera que refleje el orden de clasificación; múltiples elementos con campos de clasificación idénticos se sitúan de manera consecutiva en la matriz sin ningún orden en particular.
- Los campos numéricos se ordenan como si fueran cadenas, de manera que 100 va antes que 99, ya que “1” es un valor de cadena menor que “9”.
- No devuelve nada.

Si desea clasificar según otro orden, cree una función que realice la clasificación y pase su nombre como parámetro *compareFunction*. Puede hacer esto, por ejemplo, si desea ordenar alfabéticamente por apellido, en orden ascendente, y luego por código postal, en orden descendente.

Si desea especificar uno o más campos por los que ordenar, utilizando la clasificación predeterminada o el parámetro *options*, utilice [Array.sortOn\(\)](#).

## Ejemplo

Sintaxis 1: el ejemplo siguiente muestra el uso de `Array.sort()` con y sin pasar un valor para *option*:

```
var fruits_array = ["naranjas", "manzanas", "fresas", "piñas", "cerezas"];
trace(fruits_array.join());
fruits_array.sort();
trace(fruits_array.join());
fruits_array.sort(Array.DESENDING);
trace(fruits_array.join());
```

El panel Salida muestra el resultado siguiente:

```
naranjas,manzanas,fresas,piñas,cerezas // matriz original
manzanas,cerezas,naranjas,piñas,fresas// clasificación predeterminada
fresas,piñas,naranjas,cerezas,manzanas// orden descendente
```

Sintaxis 2: en el ejemplo siguiente se utiliza `Array.sort()` con una función de comparación.

```
var contraseñas =
    ["mom:glam","ana:ring","jay:mag","anne:home","regina:silly"];
function order (a,b){
    //Las entradas que deben ordenarse tienen el formato nombre:contraseña
    //Se ordenan utilizando solamente la parte del nombre de la entrada como
    clave
    var name1 =a.split(":")[0 ];
    var name2 =b.split(":")[0 ];
    if (name1 <name2){
        return -1;
    }
    else if (name1 >name2){
        return 1;
    }
    else {
        return 0;
    }
}
trace ("Sin ordenar:");
trace (passwords.join());

passwords.sort(order);
trace ("Ordenado:");
trace (passwords.join());
```

El panel Salida muestra el resultado siguiente:

```
Sin ordenar:
mom:glam,ana:ring,jay:mag,anne:home,regina:silly
Ordenado:
ana:ring,anne:home,jay:mag,mom:glam,regina:silly
```

## Véase también

| (OR en modo bit), `Array.sortOn()`

# Array.sortOn()

## Disponibilidad

Flash Player 6; se han añadido funciones adicionales en Flash Player 7.

## Sintaxis

```
my_array.sortOn("fieldName" )
my_array.sortOn("fieldName", option | option |... )
my_array.sortOn( [ "fieldName" , "fieldName" , ... ] )
my_array.sortOn( [ "fieldName" , "fieldName" , ... ] , option | option |... )
```

**Nota:** si se muestran corchetes ([ ]), debe incluirlos en el código; es decir, los corchetes no representan parámetros opcionales.

## Parámetros

*fieldName* Cadena que identifica un campo (de un elemento de la matriz) que debe utilizarse como valor de clasificación.

*option* Uno o más números o cadenas, separados por el operador | (OR en modo bit), que cambian el comportamiento o la clasificación respecto al predeterminado. Los siguientes valores de *option* son aceptables:

- 1 o Array.CASEINSENSITIVE
- 2 o Array.DECENDING
- 4 o Array.UNIQUE
- 8 o Array.RETURNINDEXEDARRAY
- 16 o Array.NUMERIC

Cada una de estas opciones se explica con mayor detalle en "Descripción", a continuación.

## Valor devuelto

El valor devuelto depende de si se pasa algún parámetro:

- Si especifica un valor de 4 o Array.UNIQUE como *option* y dos o más de los elementos que se están ordenando tienen campos de clasificación idénticos, Flash devuelve un valor de 0 y no modifica la matriz.
- Si especifica un valor de Array.RETURNINDEXEDARRAY como *option*, Flash devuelve una matriz que refleja el resultado de la clasificación y no modifica la matriz.
- De lo contrario, Flash no devuelve nada y modifica la matriz de manera que refleje el orden de clasificación.

## Descripción

Método; ordena los elementos de una matriz según uno o más campos de la matriz. Si pasa varios parámetros *fieldName*, el primer campo representa el campo de clasificación principal, el segundo representa el siguiente campo de clasificación, y así sucesivamente. Flash ordena siguiendo valores ASCII (Unicode). Si alguno de los elementos que se están ordenando no contiene el campo especificado en el parámetro *fieldName*, se interpreta el campo como undefined, y los elementos se sitúan de manera consecutiva en la matriz sin ningún orden en particular.

De forma predeterminada, `Array.sortOn()` funciona de esta manera:

- La clasificación distingue entre mayúsculas y minúsculas (*Z* va antes que *a*).
- La clasificación es ascendente (*a* va antes que *b*).
- La matriz se modifica de manera que refleje el orden de clasificación; múltiples elementos con campos de clasificación idénticos se sitúan de manera consecutiva en la matriz sin ningún orden en particular.
- Los campos numéricos se ordenan como si fueran cadenas, de manera que 100 va antes que 99, ya que “1” es un valor de cadena menor que “9”.
- No devuelve nada.

Puede utilizar las etiquetas *option* para omitir estos valores predeterminados. Los siguientes ejemplos utilizan diferentes etiquetas *option* a efectos ilustrativos. Si quiere ordenar una matriz sencilla (por ejemplo, una matriz con un sólo campo), o si desea especificar un orden de clasificación no admitido por el parámetro *options*, utilice `Array.sort()`.

Para pasar varias etiquetas en formato numérico, sepárelas con el operador `|` (OR en modo bit) o sume los valores de los indicadores. El código siguiente muestra tres maneras diferentes de especificar una clasificación numérica descendente:

```
my_Array.sortOn(nombreCampo, 2 | 16);
my_Array.sortOn(nombreCampo, 18);
my_Array.sortOn(nombreCampo, Array.DESENDING | Array.NUMERIC);
```

Las sugerencias para el código (véase [“Utilización de las sugerencias para el código” en la página 66](#)) están habilitadas si utiliza la forma de cadena de la etiqueta (por ejemplo, `DESCENDING`) en lugar de la forma numérica (2).

Considere la siguiente matriz:

```
var my_array:Array = new Array();
my_array.push({contraseña: "Bob", edad:29});
my_array.push({contraseña: "abcd", edad:3});
my_array.push({contraseña: "barb", edad:35});
my_array.push({contraseña: "catchy", edad:4});
```

Una clasificación predeterminada del campo de contraseña daría como resultado:

```
my_array.sortOn("contraseña")
// Bob
// abcd
// barb
// catchy
```

Una clasificación que no distinga entre mayúsculas y minúsculas del campo de contraseña daría como resultado:

```
my_array.sortOn("contraseña", Array.CASEINSENSITIVE)
// abcd
// barb
// Bob
// catchy
```

Una clasificación que no distinga entre mayúsculas y minúsculas, en orden descendente, del campo de contraseña daría como resultado:

```
my_array.sortOn("contraseña", 1|2)
// catchy
// Bob
// barb
// abcd
```

Una clasificación predeterminada del campo de edad daría como resultado:

```
my_array.sortOn("edad")
// 29
// 3
// 35
// 4
```

Una clasificación numérica del campo de edad daría como resultado:

```
my_array.sortOn("edad", 16)
// 3
// 4
// 29
// 35
```

Una clasificación numérica, en orden descendente, del campo de edad daría como resultado:

```
my_array.sortOn("edad", 18)
// 35
// 29
// 4
// 3
```

Llevar a cabo una clasificación cambia los elementos de la matriz de la siguiente manera:

```
// Antes de clasificar
// my_array[0].edad = 29;
// my_array[1].edad = 3;
// my_array[2].edad = 35;
// my_array[3].edad = 4;

// Después de una clasificación que no pase 8 como valor de option
my_array.sortOn("edad", Array.NUMERIC);
// my_array[0].edad = 3;
// my_array[1].edad = 4;
// my_array[2].edad = 29;
// my_array[3].edad = 35;
```

Llevar a cabo una clasificación que devuelva una matriz de orden no cambia los elementos de la matriz:

```
// Antes de clasificar
// my_array[0].edad = 29;
// my_array[1].edad = 3;
// my_array[2].edad = 35;
// my_array[3].edad = 4;

// Después de una clasificación que devuelve una matriz que contiene valores de índice
// Observe que no se ha modificado la matriz original.
// Puede utilizar la matriz devuelta para mostrar información ordenada
// sin modificar la matriz original.
var indexArray:Array = my_array.sortOn("edad", Array.RETURNINDEXEDARRAY);
// my_array[0].edad = 29;
```

```
// my_array[1].edad = 3;
// my_array[2].edad = 35;
// my_array[3].edad = 4;
```

## Ejemplo

Este ejemplo crea una nueva matriz y la ordena según los campos `name` y `city`: la primera clasificación utiliza `name` como primer valor de clasificación y `city` como el segundo. La primera clasificación utiliza `city` como primer valor de clasificación y `name` como el segundo.

```
var rec_array = new Array();
rec_array.push( { name: "john", city: "omaha", zip: 68144 } );
rec_array.push( { name: "john", city: "kansas city", zip: 72345 } );
rec_array.push( { name: "bob", city: "omaha", zip: 94010 } );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// da lugar a
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn( [ "name", "city" ] );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// da lugar a
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn( ["city", "name" ] );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// da lugar a
// john, kansas city
// bob, omaha
// john, omaha
```

## Véase también

| [\(OR en modo bit\)](#), [Array.sort\(\)](#)

## Array.splice()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.splice(start, deleteCount [, value0, value1...valueN])
```

### Parámetros

*start*    Índice del elemento de la matriz donde comienza la inserción y/o el borrado.

*deleteCount* Número de elementos que se van a eliminar. Este número incluye el elemento especificado en el parámetro *start*. Si no se especifica un valor para *deleteCount*, el método elimina todos los valores desde el elemento *start* hasta el último elemento de la matriz. Si el valor es 0, no se elimina ningún elemento.

*value* Parámetro opcional que especifica los valores que deben insertarse en la matriz en el punto de inserción especificado en el parámetro *start*.

**Valor devuelto**

Ninguno.

**Descripción**

Método; agrega y elimina elementos de una matriz. Este método modifica la matriz sin hacer una copia.

## Array.toString()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_array.toString()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Una cadena.

**Descripción**

Método; devuelve un valor de cadena que representa los elementos del objeto Array especificado. Todos los elementos de la matriz, empezando por el índice 0 y finalizando por el índice *my\_Array.length-1*, se convierten en una cadena concatenada separada por comas.

**Ejemplo**

En el ejemplo siguiente se crea la matriz *my\_Array*, ésta se convierte en una cadena y se muestra 1,2,3,4,5 en el panel Salida.

```
my_array = new Array();  
my_array[0] = 1;  
my_array[1] = 2;  
my_array[2] = 3;  
my_array[3] = 4;  
my_array[4] = 5;  
trace(my_array.toString());
```

## Array.unshift()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_array.unshift(value1,value2,...valueN)
```

### Parámetros

*value1*,...*valueN* Uno o más números, elementos o variables que se deben insertar al principio de la matriz.

### Valor devuelto

La nueva longitud de la matriz.

### Descripción

Método; agrega uno o más elementos al principio de una matriz y devuelve la nueva longitud de la matriz.

## Array()

### Disponibilidad

Flash Player 6 .

### Sintaxis

```
Array  
Array( [element0 [, element1 , element2,...elementN ] ])
```

### Parámetros

*element* Uno o más elementos que se colocarán en la matriz.

### Valor devuelto

Una matriz.

### Descripción

Función de conversión; crea una matriz nueva vacía o convierte los elementos especificados en una matriz. El resultado de esta función es similar a la creación de una matriz mediante el constructor de la clase Array (véase [“Constructor para la clase Array” en la página 282](#)).

## asfunction

### Disponibilidad

Flash Player 5.

### Sintaxis

```
asfunction: function, "parameter"
```



## Parámetros

*function* Identificador de una función.

*parameter* Cadena que se pasa a la función a la que hace referencia el parámetro *function*.

## Valor devuelto

Ninguno.

## Descripción

Protocolo; un protocolo especial para URL en campos de texto HTML. En campos de texto HTML, el texto puede convertirse en un hipervínculo con una etiqueta A HTML. El atributo HREF de la etiqueta A contiene una URL que puede ser de un protocolo estándar como HTTP, HTTPS o FTP. El protocolo *asfunction* es un protocolo adicional específico de Flash que hace que el vínculo invoque una función de ActionScript.

## Ejemplo

En este ejemplo, la función `MyFunc()` se define en las tres primeras líneas de código. El objeto de `TextField myTextField` se asocia con un campo de texto HTML. El texto "Haga clic aquí" es un hipervínculo dentro del campo de texto. La función `MyFunc()` se llama cuando el usuario hace clic en el hipervínculo:

```
function MyFunc(arg){
    trace ("Ha hecho clic aquí El argumento era "+arg);
}
myTextField.htmlText = "<A HREF=\"asfunction:MyFunc,foo\">Haga clic aquí</A>";
```

Cuando se hace clic en el hipervínculo, se muestra el resultado siguiente en el panel Salida:

Ha hecho clic aquí. El argumento es Foo.

# Clase Boolean

## Disponibilidad

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).

## Descripción

La clase Boolean es un objeto envolvente con las mismas funciones que el objeto Boolean estándar de JavaScript. Utilice la clase Boolean para recuperar el tipo de datos primitivo o la representación de cadena de un objeto Boolean.

Debe utilizar el constructor `new Boolean()` para crear un objeto Boolean antes de llamar a estos métodos.

## Resumen de métodos para la clase Boolean

Método	Descripción
<code>Boolean.toString()</code>	Devuelve la representación de la cadena ("true") o ("false") del objeto Boolean.
<code>Boolean.valueOf()</code>	Devuelve el tipo de valor primitivo del objeto Boolean especificado.

## Constructor para la clase Boolean

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Boolean([x])
```

### Parámetros

*x*   Cualquier expresión. Este parámetro es opcional.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto Boolean. Si omite el parámetro *x*, el objeto Boolean se inicializa con el valor `false`. Si especifica un valor para el parámetro *x*, el método lo obtiene y devuelve el resultado como un valor Boolean según las reglas de la función [Boolean\(\)](#).

### Ejemplo

En el código siguiente se crea un nuevo objeto Boolean vacío denominado `myBoolean`.

```
myBoolean = new Boolean();
```

## Boolean.toString()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myBoolean.toString()
```

### Parámetros

Ninguno.

### Valor devuelto

Valor booleano.

### Descripción

Método; devuelve la representación de la cadena (`"true"` o `"false"`) del objeto Boolean.

## Boolean.valueOf()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myBoolean.valueOf()
```

## Parámetros

Ninguno.

## Valor devuelto

Valor booleano.

## Descripción

Método; devuelve `true` si el tipo de valor original del objeto `Boolean` es `true`, `false` si es `false`.

## Ejemplo

```
var x:Boolean = new Boolean();
trace(x.valueOf()); // false
x = (6==3+3);
trace(x.valueOf()); // true
```

# Boolean()

## Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

## Sintaxis

`Boolean(expression)`

## Parámetros

*expression* Expresión que debe convertirse en un valor booleano.

## Valor devuelto

Un valor booleano o el valor *expression*, como se describe a continuación.

## Descripción

Función; convierte el parámetro *expression* en un valor booleano y devuelve un valor de acuerdo con lo siguiente:

Si *expression* es un valor booleano, el valor devuelto es *expression*.

Si *expression* es un número, el valor devuelto es `true` si el número no es cero, o `false` si es cero.

Si *expression* es una cadena, el valor devuelto es el siguiente:

- En archivos publicados para Flash Player 6 o versiones anteriores, la cadena se convierte primero a un número; el valor es `true` si el número es diferente de cero, y `false` si no lo es.
- En archivos publicados para Flash Player 7 o versiones posteriores, el resultado es `true` si la longitud de la cadena es superior a cero; el valor es `false` para una cadena vacía.

Si *expression* no está definida, el valor devuelto es `false`.

Si *expression* es un clip de película o un objeto, el valor devuelto es `true`.

## Véase también

[Clase Boolean](#)

# break

## Disponibilidad

Flash Player 4.

## Sintaxis

`break`

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; aparece dentro de una reproducción indefinida (`for`, `for..in`, `do while` o `while`) o dentro de un bloque de sentencias asociadas con un bloque "case" concreto dentro de una acción `switch`. La acción `break` da instrucciones a Flash para que se salte el resto del cuerpo de la reproducción, detenga la acción del bucle y ejecute la sentencia que sigue a la sentencia de la reproducción indefinida. Cuando se utiliza la acción `break`, el intérprete de Flash omite el resto de las sentencias de dicho bloque `case` y salta a la primera sentencia que sigue a la acción `switch` que la contiene. Utilice la acción `break` para romper una serie de reproducciones anidadas.

## Ejemplo

En el ejemplo siguiente se utiliza la acción `break` para salir de una reproducción indefinida que, si no, es infinita.

```
i = 0;
while (true) {
    if (i >= 100) {
        break;
    }
    i++;
}
```

## Véase también

[break](#), [for](#), [for..in](#), [do while](#), [while](#), [switch](#), [case](#)

# Clase Button

## Disponibilidad

Flash Player 6.

## Descripción

Todos los símbolos de botón de un archivo SWF son instancias del objeto `Button`. Puede asignar un nombre de instancia a un botón en el inspector de propiedades y utilizar los métodos y las propiedades del objeto `Button` para manipular botones con ActionScript. Los nombres de instancias de botón se muestran en el Explorador de películas y en el cuadro de diálogo Insertar ruta de destino del panel Acciones.

La clase `Button` hereda de la [Clase Object](#).

## Resumen de métodos para la clase Button

Método	Descripción
<code>Button.getDepth()</code>	Devuelve la profundidad de una instancia de botón.

## Resumen de propiedades de la clase Button

Propiedad	Descripción
<code>Button._alpha</code>	Valor de transparencia de una instancia de botón.
<code>Button.enabled</code>	Indica si un botón está activo.
<code>Button._focusrect</code>	Indica si un botón seleccionado está rodeado por un rectángulo amarillo.
<code>Button._height</code>	Altura de una instancia de botón, en píxeles.
<code>Button._highquality</code>	Nivel de suavizado aplicado al archivo SWF actual.
<code>Button.menu</code>	Asocia un objeto ContextMenu con el objeto de botón
<code>Button._name</code>	Nombre de instancia de una instancia de botón.
<code>Button._parent</code>	Referencia al clip de película u objeto que contiene el objeto o clip de película actual.
<code>Button._quality</code>	Indica la calidad de representación del archivo SWF.
<code>Button._rotation</code>	Grado de rotación de una instancia de botón.
<code>Button._soundbuftime</code>	Número de segundos que tarda un sonido en precargarse.
<code>Button.tabEnabled</code>	Indica si un botón se incluye en un orden de tabulación automático.
<code>Button.tabIndex</code>	Indica el orden de tabulación de un objeto.
<code>Button._target</code>	Ruta de destino de una instancia de botón.
<code>Button.trackAsMenu</code>	Indica si otros botones pueden recibir eventos al soltar el ratón.
<code>Button._url</code>	URL del archivo SWF que ha creado la instancia de botón.
<code>Button.useHandCursor</code>	Indica si el cursor con forma de mano aparece cuando el ratón pasa sobre un botón.
<code>Button._visible</code>	Valor booleano que determina si una instancia de botón está oculta o visible.
<code>Button._width</code>	Anchura de una instancia de botón, en píxeles.
<code>Button._x</code>	Coordenada x de una instancia de botón.
<code>Button._xmouse</code>	Coordenada x del puntero del ratón en relación con una instancia de botón.
<code>Button._xscale</code>	Valor que especifica el porcentaje de escala horizontal que se aplica a una instancia de botón.
<code>Button._y</code>	Coordenada y de una instancia de botón.

Propiedad	Descripción
<code>Button._ymouse</code>	Coordenada y del puntero del ratón en relación con una instancia de botón.
<code>Button._yscale</code>	Valor que especifica el porcentaje de escala vertical que se aplica a una instancia de botón.

## Resumen del controlador de eventos para la clase **Button**

Controlador de eventos	Descripción
<code>Button.onDragOut</code>	Se invoca cuando se presiona el botón del ratón sobre el botón y, a continuación, el puntero se desplaza fuera del botón.
<code>Button.onDragOver</code>	Se invoca cuando el usuario presiona y arrastra el botón del ratón fuera del botón y, a continuación, sobre éste.
<code>Button.onKeyUp</code>	Se invoca cuando se suelta una tecla.
<code>Button.onKillFocus</code>	Se invoca cuando se deja de seleccionar un botón.
<code>Button.onPress</code>	Se invoca cuando se presiona el ratón mientras el puntero está sobre un botón.
<code>Button.onRelease</code>	Se invoca cuando se suelta el ratón mientras el puntero está sobre un botón.
<code>Button.onReleaseOutside</code>	Se invoca cuando se suelta el ratón mientras el puntero está fuera del botón después de presionar el botón mientras el puntero está dentro del botón.
<code>Button.onRollOut</code>	Se invoca cuando el puntero se desplaza fuera del área de un botón.
<code>Button.onRollOver</code>	Se invoca cuando el puntero del ratón se desplaza sobre un botón.
<code>Button.onSetFocus</code>	Se invoca cuando un botón se selecciona en el momento de la entrada y se suelta una tecla.

## Button.\_alpha

### Disponibilidad

Flash Player 6.

### Sintaxis

`my_btn._alpha`

### Descripción

Propiedad; valor de transparencia alfa del botón especificado por `my_btn`. Los valores válidos van de 0 (completamente transparente) a 100 (completamente opaco). El valor predeterminado es 100. Los objetos de un botón con el valor `_alpha` establecido en 0 están activos, aun cuando sean invisibles.

## Ejemplo

El código siguiente establece la propiedad `_alpha` de un botón denominado `star_btn` en 30% cuando se hace clic en el botón.

```
on(release) {  
    star_btn._alpha = 30;  
}
```

## Véase también

[MovieClip.\\_alpha](#), [TextField.\\_alpha](#)

# Button.enabled

## Disponibilidad

Flash Player 6.

## Sintaxis

`my_btn.enabled`

## Descripción

Propiedad; un valor booleano que especifica si un botón está activado. El valor predeterminado es `true`.

# Button.\_focusrect

## Disponibilidad

Flash Player 6.

## Sintaxis

`my_btn._focusrect`

## Descripción

Propiedad; un valor booleano que especifica si aparece un rectángulo amarillo alrededor del botón que está seleccionado mediante el teclado. Esta propiedad puede suplantar la propiedad `_focusrect` global.

# Button.getDepth()

## Disponibilidad

Flash Player 6.

## Sintaxis

`my_btn.getDepth()`

## Valor devuelto

Un número entero.

## Descripción

Método; devuelve la profundidad de una instancia de botón.

## Button.\_height

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_height*

### Descripción

Propiedad; indica la altura del botón expresada en píxeles.

### Ejemplo

En el código de ejemplo siguiente se establece la altura y la anchura de un botón cuando el usuario hace clic con el ratón.

```
my_btn._width = 200;  
my_btn._height = 200;
```

## Button.\_highquality

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_highquality*

### Descripción

Propiedad (global); especifica el nivel de suavizado aplicado al archivo SWF actual. Especifique 2 (calidad óptima) para aplicar alta calidad con el suavizado de mapa de bits siempre activado. Especifique 1 (alta calidad) para aplicar suavizado; esto suavizará los mapas de bits si el archivo SWF no contiene animación. Especifique 0 (baja calidad) para evitar el suavizado.

### Véase también

[\\_quality](#)

## Button.menu

### Disponibilidad

Flash Player 7.

### Utilización

```
my_button.menu = contextMenu
```

### Parámetros

*contextMenu*    Objeto ContextMenu.



## Descripción

Propiedad; asocia el objeto `ContextMenu` *contextMenu* con el objeto `Button` denominado *my\_btn*. La clase `ContextMenu` permite modificar el menú contextual que aparece al hacer clic con el botón derecho del ratón (Windows) o al mantener presionada la tecla Control y hacer clic (Macintosh) en Flash Player.

## Ejemplo

En el ejemplo siguiente se asigna un objeto `ContextMenu` a un objeto `Button` denominado *save\_btn*. El objeto `ContextMenu` contiene un único elemento de menú (denominado “Guardar...”) con una función de controlador callback asociada denominada *doSave* (no se muestra).

```
var menu_cm = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Guardar...", doSave));
function doSave(menu, obj) {
    // "Guardar" el código aquí
}
save_btn.menu = menu_cm;
```

## Véase también

[Clase ContextMenu](#), [Clase ContextMenuItem](#), [MovieClip.menu](#), [TextField.menu](#)

# Button.\_name

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_btn*.*\_name*

## Descripción

Propiedad; nombre de instancia del botón especificado por *my\_btn*.

# Button.onDragOut

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_btn.onDragOut = function() {
    // las sentencias se escriben aquí
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se presiona el botón del ratón sobre el botón y, a continuación, el puntero se desplaza fuera del botón.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

## Button.onDragOver

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onDragOver = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el usuario presiona y arrastra el botón del ratón fuera del botón y, a continuación, sobre éste.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el controlador `onKeyDown` que envía una acción `trace()` al panel Salida:

```
my_btn.onDragOver = function () {  
    trace ("se ha llamado al método onDragOver");  
};
```

### Véase también

[Button.onKeyUp](#)

## Button.onKeyDown

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onKeyDown = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un botón se selecciona mediante el teclado y se presiona una tecla. El controlador de eventos `onKeyDown` se invoca sin parámetros. Puede utilizar `Key.getAscii()` y `Key.getCode()` para determinar qué tecla se ha presionado.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onKeyDown` que envía una acción `trace()` al panel Salida.

```
my_btn.onKeyDown = function () {  
    trace ("se ha llamado al método onKeyDown");  
};
```

### Véase también

[Button.onKeyUp](#)

## Button.onKeyUp

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onKeyUp = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un botón se selecciona en el momento de la entrada y se suelta una tecla. El controlador de eventos `onKeyUp` se invoca sin parámetros. Puede utilizar `Key.getAscii()` y `Key.getCode()` para determinar qué tecla se ha presionado.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onKeyPress` que envía una acción `trace()` al panel Salida.

```
my_btn.onKeyUp = function () {  
    trace ("se ha llamado al método onKeyUp");  
};
```

## Button.onKillFocus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onKillFocus = function (newFocus) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*newFocus*    Objeto que está seleccionado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un botón deja de seleccionarse mediante el teclado. El método `onKillFocus` recibe un parámetro, *newFocus*, que es un objeto que representa el nuevo objeto seleccionado. Si no hay ningún objeto seleccionado, *newFocus* contiene el valor `null`.

## Button.onPress

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onPress = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se presiona un botón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onPress` que envía una acción `trace()` al panel Salida.

```
my_btn.onPress = function () {  
    trace ("se ha llamado al método onPress");  
};
```

## Button.onRelease

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onRelease = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se suelta un botón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onRelease` que envía una acción `trace()` al panel Salida.

```
my_btn.onRelease = function () {  
    trace ("se ha llamado al método onRelease");  
};
```

## Button.onReleaseOutside

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onReleaseOutside = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se suelta el ratón mientras el puntero está fuera del botón después de que se haya presionado el botón mientras el puntero estaba dentro del botón.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onReleaseOutside` que envía una acción `trace()` al panel Salida

```
my_btn.onReleaseOutside = function () {  
    trace ("se ha llamado al método onReleaseOutside");  
};
```

## Button.onRollOut

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onRollOut = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el puntero se desplaza fuera del área de un botón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onRollOut` que envía una acción `trace()` al panel Salida.

```
my_btn.onRollOut = function () {  
    trace ("se ha llamado al método onRollOut");  
};
```

## Button.onRollOver

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onRollOver = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el puntero se desplaza sobre el área de un botón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente, se define una función para el controlador `onRollOver` que envía una acción `trace()` al panel Salida.

```
my_btn.onRollOver = function () {  
    trace ("se ha llamado al método onRollOver");  
};
```

## Button.onSetFocus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn.onSetFocus = function(oldFocus){  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*oldFocus*    Objeto que va a dejar de estar seleccionado mediante el teclado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un botón se selecciona mediante el teclado. El parámetro *oldFocus* es el objeto que deja de estar seleccionado. Por ejemplo, si el usuario presiona la tecla Tabulador para cambiar la selección de un campo de texto a un botón en el momento de la entrada, *oldFocus* contiene la instancia de campo de texto.

Si no hay ningún objeto seleccionado anteriormente, *oldFocus* contiene un valor `null`.

## Button.\_parent

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn._parent.property  
_parent.property
```

### Descripción

Propiedad; referencia al clip de película u objeto que contiene el objeto o clip de película actual. El objeto actual es el que contiene el código de ActionScript que hace referencia a `_parent`.

Utilice `_parent` para especificar una ruta relativa a los clips de película u objetos que están por encima del clip de película u objeto actual. Puede utilizar `_parent` para subir varios niveles en la lista de visualización, como se muestra a continuación:

```
_parent._parent._alpha = 20;
```

#### Véase también

[MovieClip.\\_parent](#), [\\_root](#), [targetPath](#)

## Button.\_quality

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn._quality
```

### Descripción

Propiedad (global); establece o recupera la calidad de representación utilizada para un archivo SWF. Las fuentes de dispositivo siempre son dentadas, de modo que no se ven afectadas por la propiedad `_quality`.

**Nota:** aunque puede especificar esta propiedad para un objeto Button, en realidad se trata de una propiedad global y puede especificar su valor simplemente como `_quality`. Para más información, consulte [\\_quality](#).

## Button.\_rotation

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_btn._rotation
```

### Descripción

Propiedad; la rotación del botón, en grados, respecto a su orientación original. Los valores entre 0 y 180 corresponden a la rotación en el sentido de las agujas del reloj, mientras que los valores entre 0 y -180 corresponden a la rotación en el sentido contrario a las agujas del reloj. Los valores que quedan fuera de este rango se suman o se restan de 360 para obtener un valor dentro del rango. Por ejemplo, la sentencia `my_btn._rotation = 450` es la misma que `my_btn._rotation = 90`.

#### Véase también

[MovieClip.\\_rotation](#), [TextField.\\_rotation](#)



## Button.\_soundbuftime

### Disponibilidad

Flash Player 6.

### Sintaxis

*myButton.\_soundbuftime*

### Descripción

Propiedad (global); entero que especifica el número de segundos que un sonido se almacena previamente en una memoria intermedia antes de que empiece a reproducirse.

**Nota:** aunque puede especificar esta propiedad para un objeto Button, en realidad se trata de una propiedad global y puede especificar su valor simplemente como `_soundbuftime`. Para más información, consulte [\\_soundbuftime](#).

## Button.tabEnabled

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.tabEnabled*

### Descripción

Propiedad; especifica si *my\_btn* se incluye en el orden de tabulación automático. El valor predeterminado es `undefined`.

Si la propiedad `tabEnabled` es `undefined` o `true`, el objeto se incluye en el orden de tabulación automático. Si la propiedad `tabIndex` también se establece en un valor, el objeto se incluye en el orden de tabulación personalizado. Si `tabEnabled` es `false`, el objeto no se incluye en el orden de tabulación personalizado o automático, aunque se establezca la propiedad `tabIndex`.

### Véase también

[Button.tabIndex](#), [MovieClip.tabEnabled](#), [TextField.tabEnabled](#)

## Button.tabIndex

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.tabIndex*

### Descripción

Propiedad; permite personalizar el orden de tabulación de los objetos de un archivo SWF. Puede establecer la propiedad `tabIndex` de un botón, un clip de película o una instancia de campo de texto; de manera predeterminada el valor es `undefined`.

Si alguno de los objetos que se muestran actualmente en el archivo SWF contiene una propiedad `tabIndex`, el orden de tabulación automático está desactivado, y el orden de tabulación se calcula a partir de las propiedades `tabIndex` de los objetos del archivo SWF. El orden de tabulación personalizado sólo incluye objetos que tienen propiedades `tabIndex`.

La propiedad `tabIndex` puede ser un entero no negativo. Los objetos se ordenan de acuerdo con sus propiedades `tabIndex`, en orden ascendente. Un objeto cuyo valor `tabIndex` es 1 precede a un objeto cuyo valor `tabIndex` es 2. Si dos objetos tienen la misma propiedad `tabIndex`, el que precede en el orden de tabulación es `undefined`.

El orden de tabulación personalizado definido por la propiedad `tabIndex` es *flat*. Esto significa que no se tienen en cuenta las relaciones jerárquicas de los objetos del archivo SWF. Todos los objetos del archivo SWF con propiedades `tabIndex` se colocan en el orden de tabulación, que viene determinado por el orden de los valores `tabIndex`. Si dos objetos tienen el mismo valor de `tabIndex`, el precedente será `undefined`. No debe utilizarse el mismo valor de `tabIndex` para varios objetos.

#### Véase también

[Button.tabEnabled](#), [MovieClip.tabChildren](#), [MovieClip.tabEnabled](#),  
[MovieClip.tabIndex](#), [TextField.tabIndex](#)

## Button.\_target

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
myButton._target
```

#### Descripción

Propiedad (sólo lectura); devuelve la ruta de destino de la instancia de botón especificada por `my_btn`.

#### Véase también

[targetPath](#)

## Button.trackAsMenu

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_btn.trackAsMenu
```

#### Descripción

Propiedad; valor booleano que indica si otros botones o clips de película pueden recibir eventos al soltar el botón. Permite crear menús. Puede establecer la propiedad `trackAsMenu` en cualquiera de los objetos de clip de película o botón. Si no se ha definido la propiedad `trackAsMenu`, el comportamiento predeterminado es `false`.

Puede cambiar la propiedad `trackAsMenu` en cualquier momento; el botón modificado asume el nuevo comportamiento inmediatamente.

#### Véase también

[MovieClip.trackAsMenu](#)

## Button.\_url

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_btn._url
```

#### Descripción

Propiedad (sólo lectura); recupera la URL del archivo SWF que ha creado el botón.

## Button.useHandCursor

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_btn.useHandCursor
```

#### Descripción

Propiedad; valor booleano que, cuando se establece en `true` (valor predeterminado), indica si se visualiza un cursor con forma de mano cuando el ratón pasa por encima de un botón. Si esta propiedad está establecida en `false`, se utiliza el cursor de flecha en su lugar.

Puede cambiar la propiedad `useHandCursor` en cualquier momento; el botón modificado asume de inmediato el comportamiento del nuevo cursor. La propiedad `useHandCursor` puede leerse fuera de un objeto prototipo.

## Button.\_visible

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_btn._visible
```

#### Descripción

Propiedad; valor booleano que indica si el botón especificado por `my_btn` es visible. Los botones que no están visibles (propiedad `_visible` establecida en `false`) se desactivan.

#### Véase también

[MovieClip.\\_visible](#), [TextField.\\_visible](#)

## Button.\_width

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_width*

### Descripción

Propiedad; ancho del botón expresado en píxeles.

### Ejemplo

En el ejemplo siguiente se establecen las propiedades de altura y anchura de un botón.

```
my_btn._width=200;  
my_btn._height=200;
```

### Véase también

[MovieClip.\\_width](#)

## Button.\_x

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_x*

### Descripción

Propiedad; entero que establece la coordenada *x* de un botón con relación a las coordenadas locales del clip de película principal. Si un botón está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el botón se encuentra dentro de un clip de película que tiene transformaciones, el botón pertenece al sistema de coordenadas locales del clip de película que lo contiene. Por lo tanto, en el caso de un clip de película girado 90° en sentido contrario a las agujas del reloj, el botón incluido hereda un sistema de coordenadas girado 90° en sentido contrario a las agujas del reloj. Las coordenadas del botón hacen referencia a la posición del punto de registro.

### Véase también

[Button.\\_xscale](#), [Button.\\_y](#), [Button.\\_yscale](#)

## Button.\_xmouse

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_xmouse*

### Descripción

Propiedad (sólo lectura); devuelve la coordenada *x* de la posición del ratón con relación al botón.

### Véase también

[Button.\\_ymouse](#)

## Button.\_xscale

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_xscale*

### Descripción

Propiedad; escala horizontal del botón que se aplica desde el punto de registro del botón, expresada como porcentaje. El punto de registro predeterminado es (0,0).

Cambiar la escala del sistema de coordenadas local afecta a los valores de las propiedades *\_x* e *\_y*, que se definen en píxeles. Por ejemplo, si se cambia la escala del clip de película principal al 50%, al establecer la propiedad *\_x*, se moverá un objeto del botón la mitad del número de píxeles que se movería si el archivo SWF estuviera al 100%.

### Véase también

[Button.\\_x](#), [Button.\\_y](#), [Button.\\_yscale](#)

## Button.\_y

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_y*

### Descripción

Propiedad; coordenada *y* del botón con relación a las coordenadas locales del clip de película principal. Si un botón está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el botón se encuentra dentro de otro clip de película que tiene transformaciones, el botón está en el sistema de coordenadas locales del clip de película que lo contiene. Por lo tanto, en el caso de un clip de película girado 90° en sentido contrario a las agujas del reloj, el botón incluido hereda un sistema de coordenadas girado 90° en sentido contrario a las agujas del reloj. Las coordenadas del botón hacen referencia a la posición del punto de registro.

### Véase también

[Button.\\_x](#), [Button.\\_xscale](#), [Button.\\_yscale](#)

## Button.\_ymouse

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_ymouse*

### Descripción

Propiedad (sólo lectura); indica la coordenada *y* de la posición del ratón con relación al botón.

### Véase también

[Button.\\_xmouse](#)

## Button.\_yscale

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_btn.\_yscale*

### Descripción

Propiedad; escala vertical del botón que se aplica desde el punto de registro del botón, expresada como porcentaje. El punto de registro predeterminado es (0,0).

### Véase también

[Button.\\_y](#), [Button.\\_x](#), [Button.\\_xscale](#)

## call()

### Disponibilidad

Flash Player 4. Esta acción se eliminó en Flash 5 y Macromedia recomienda que se utilice la acción `function` en su lugar.

### Sintaxis

*call(frame)*

### Parámetros

*frame* Etiqueta o número de un fotograma en la línea de tiempo.

### Valor devuelto

Ninguno.

### Descripción

Acción desestimada; ejecuta el script en el fotograma al que se ha llamado sin mover la cabeza lectora a ese fotograma. Las variables locales no existen después de que el script se ejecute.

### Véase también

[function](#), [Function.call\(\)](#)

# Clase Camera

## Disponibilidad

Flash Player 6.

## Descripción

La clase Camera se utiliza principalmente con Macromedia Flash Communication Server, pero puede utilizarse de forma limitada sin el servidor.

La clase Camera permite capturar vídeo desde una cámara de vídeo conectada al equipo en el que se ejecuta Macromedia Flash Player. Esto le permite, por ejemplo, controlar la entrada de vídeo de una webcam conectada al sistema local. Flash proporciona funciones de audio parecidas; para más información, consulte la entrada [Clase Microphone](#).

Para crear un objeto Camera o hacer referencia al mismo, utilice `Camera.get()`.

## Resumen de métodos para la clase Camera

Método	Descripción
<code>Camera.get()</code>	Devuelve un objeto Camera predeterminado o especificado, o <code>null</code> si la cámara no está disponible.
<code>Camera.setMode()</code>	Establece aspectos del modo de captura de la cámara, incluidos la altura, la anchura y los fotogramas por segundo.
<code>Camera.setMotionLevel()</code>	Especifica el movimiento necesario para invocar a <code>Camera.onActivity(true)</code> y el tiempo que debe transcurrir sin movimiento antes de invocar a <code>Camera.onActivity(false)</code> .
<code>Camera.setQuality()</code>	Número entero que especifica la cantidad máxima de ancho de banda que puede utilizar la salida de vídeo actual, expresada en bytes por segundo.

## Resumen de propiedades para la clase Camera

Propiedad (sólo lectura)	Descripción
<code>Camera.activityLevel</code>	Cantidad de movimiento que detecta la cámara.
<code>Camera.bandwidth</code>	Cantidad máxima de ancho de banda que puede utilizar la salida de vídeo actual, expresada en bytes.
<code>Camera.currentFps</code>	Velocidad a la que la cámara captura datos, expresada en fotogramas por segundo.
<code>Camera.fps</code>	Velocidad a la que desea que la cámara capture datos, expresada en fotogramas por segundo.
<code>Camera.height</code>	Altura actual de captura, expresada en píxeles.
<code>Camera.index</code>	Índice de la cámara, según se refleja en la matriz devuelta por <code>Camera.names</code> .
<code>Camera.motionLevel</code>	Cantidad de movimiento necesario para invocar <code>Camera.onActivity(true)</code> .

Propiedad (sólo lectura)	Descripción
<code>Camera.motionTimeOut</code>	Número de milisegundos que deben transcurrir entre el momento en que la cámara deja de detectar movimiento y el momento en que se invoca <code>Camera.onActivity(false)</code> .
<code>Camera.muted</code>	Valor booleano que especifica si el usuario ha permitido o denegado el acceso a la cámara.
<code>Camera.name</code>	Nombre de la cámara según lo especifica el hardware de la cámara.
<code>Camera.names</code>	Propiedad Class; matriz de cadenas que refleja los nombres de todos los dispositivos de captura de vídeo disponibles, incluidas las tarjetas de vídeo y las cámaras.
<code>Camera.quality</code>	Número entero que especifica el nivel necesario de calidad de imagen, según lo que determina la cantidad de compresión que se aplica a cada fotograma de vídeo.
<code>Camera.width</code>	Anchura de captura actual, expresada en píxeles.

## Resumen de controladores de eventos para la clase Camera

Controlador de eventos	Descripción
<code>Camera.onActivity</code>	Se invoca cuando la cámara empieza a detectar movimiento o deja de detectarlo.
<code>Camera.onStatus</code>	Se invoca cuando el usuario permite o deniega el acceso a la cámara.

## Constructor para la clase Camera

Véase `Camera.get()`.

## Camera.activityLevel

### Disponibilidad

Flash Player 6.

### Sintaxis

`active_cam.activityLevel`

### Descripción

Propiedad de sólo lectura; valor numérico que especifica la cantidad de movimiento que detecta la cámara. Los valores oscilan entre 0 (no se detecta movimiento) y 100 (se detecta una gran cantidad de movimiento). El valor de esta propiedad puede ayudarle a determinar si debe pasar un valor a `Camera.setMotionLevel()`.

Si la cámara está disponible pero no se está utilizando porque no se ha llamado a `Video.attachVideo()`, esta propiedad se establece en -1.

Si sólo existe flujo de vídeo local descomprimido, esta propiedad sólo se establece si se ha asignado una función al controlador de eventos `Camera.onActivity`. De lo contrario, no se define.

### Véase también

`Camera.motionLevel`, `Camera.setMotionLevel()`



## Camera.bandwidth

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.bandwidth*

### Descripción

Propiedad de sólo lectura; número entero que especifica la cantidad máxima de ancho de banda que puede utilizar la salida de vídeo actual, expresada en bytes. El valor 0 indica que el vídeo Flash puede utilizar todo el ancho de banda que sea necesario para mantener la calidad de fotograma que se desee.

Para establecer esta propiedad, utilice [Camera.setQuality\(\)](#).

### Ejemplo

En el ejemplo siguiente se carga otro archivo SWF si el ancho de banda de la cámara es de 32 kilobytes o superior.

```
if(myCam.bandwidth >= 32768){  
    loadMovie("splat.swf",_root.hiddenvar);  
}
```

### Véase también

[Camera.setQuality\(\)](#)

## Camera.currentFps

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.currentFps*

### Descripción

Propiedad de sólo lectura; velocidad a la que la cámara captura datos, expresada en fotogramas por segundo. Esta propiedad no puede establecerse; no obstante, puede utilizar el método [Camera.setMode\(\)](#) para establecer una propiedad relacionada ([Camera.fps](#)), que especifica la velocidad máxima de fotogramas a la que desea que la cámara capture datos.

### Véase también

[Camera.fps](#), [Camera.setMode\(\)](#)

# Camera.fps

## Disponibilidad

Flash Player 6.

## Sintaxis

*active\_cam.fps*

## Descripción

Propiedad de sólo lectura; velocidad máxima a la que desea que la cámara capture datos, expresada en fotogramas por segundo. La velocidad máxima posible depende de las características de la cámara; es decir, si la cámara no admite el valor especificado, dicha velocidad de fotogramas no se alcanzará.

- Para establecer el valor que desee utilizar para esta propiedad, utilice `Camera.setMode()`.
- Para determinar la velocidad a la que la cámara está capturando datos actualmente, utilice la propiedad `Camera.currentFps`.

## Ejemplo

En el ejemplo siguiente se establece la velocidad de fotogramas por segundo (fps) de la cámara activa, `myCam.fps`, en el valor especificado en el cuadro de texto de usuario,

```
this.config.txt_fps.
```

```
if (this.config.txt_fps != undefined) {  
    myCam.setMode(myCam.width, myCam.height, this.config.txt_fps, false);  
}
```

**Nota:** la función `setMode` no garantiza el valor de fps solicitado; establece la velocidad de fotogramas por segundo solicitada o la más rápida disponible.

## Véase también

`Camera.currentFps`, `Camera.setMode()`

# Camera.get()

## Disponibilidad

Flash Player 6.

## Sintaxis

`Camera.get([index])`

**Nota:** la sintaxis correcta es `Camera.get()`. Para asignar el objeto `Camera` a una variable, utilice una sintaxis del tipo `active_cam = Camera.get()`.

## Parámetros

*index* Número entero opcional con base cero que especifica qué cámara debe obtenerse, según lo que determina la matriz devuelta por la propiedad `Camera.names`. Para obtener la cámara predeterminada (generalmente la recomendada para la mayoría de las aplicaciones), omita este parámetro.

## Valor devuelto

- Si no se especifica *index*, este método devuelve una referencia a la cámara predeterminada o, si la está utilizando otra aplicación, a la primera cámara disponible. Si se ha instalado más de una cámara, el usuario puede especificar la cámara predeterminada en el panel Configuración de la cámara de Flash Player. Si no hay ninguna cámara disponible o instalada, el método devuelve `null`.
- Si se especifica *index*, este método devuelve una referencia a la cámara solicitada, o `null` si no está disponible.

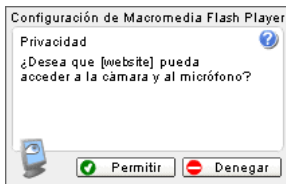
## Descripción

Método; devuelve una referencia al objeto `Camera` para capturar vídeo. Para empezar realmente a capturar vídeo, debe asociar el objeto `Camera` a un objeto `Video` (véase [Video.attachVideo\(\)](#)).

A diferencia de los objetos que ha creado mediante el constructor `new`, varias llamadas a [Camera.get\(\)](#) hacen referencia a la misma cámara. Por lo tanto, si el script contiene las líneas `first_cam = Camera.get()` y `second_cam = Camera.get()`, tanto `first_cam` como `second_cam` hacen referencia a la misma cámara (predeterminada).

Por lo general, no debe pasar un valor para *index*; simplemente debe utilizar `Camera.get()` para devolver una referencia a la cámara predeterminada. Mediante el panel de configuración de la cámara, que se mencionará más adelante en esta sección, el usuario puede especificar la cámara predeterminada que Flash debe utilizar. Si pasa un valor para *index*, es posible que esté intentando hacer referencia a una cámara distinta de la que prefiere el usuario. Es posible que utilice *index* en ocasiones contadas, como por ejemplo, si la aplicación captura vídeo de dos cámaras al mismo tiempo.

Si un archivo SWF intenta acceder a la cámara que ha devuelto [Camera.get\(\)](#), Flash Player mostrará el cuadro de diálogo de confidencialidad en el que el usuario podrá elegir si debe permitirse o denegarse el acceso a la cámara. Asegúrese de que establece un tamaño mínimo de 215 x 138 píxeles; éste es el tamaño mínimo que Flash necesita para visualizar el cuadro de diálogo.



Cuando el usuario responde a este cuadro de diálogo, el controlador de eventos [Camera.onStatus](#) devuelve un objeto de información que indica la respuesta del usuario. Para determinar si el usuario ha denegado o permitido el acceso a la cámara sin procesar este controlador de eventos, utilice la propiedad [Camera.muted](#).

El usuario también puede especificar valores de confidencialidad permanentes para un dominio determinado; para ello, debe hacer clic con el botón derecho del ratón (Windows) o presionar Control y hacer clic (Macintosh) durante la reproducción de un archivo SWF, seleccionar Configuración, abrir el panel de confidencialidad y seleccionar Recordar.



No puede utilizar ActionScript para establecer el valor Permitir o Denegar para un usuario, pero puede mostrar el panel de confidencialidad del usuario mediante `System.showSettings(0)`. Si el usuario selecciona Recordar, Flash Player deja de mostrar el cuadro de diálogo de confidencialidad en las películas de ese dominio.

Si `Camera.get` devuelve `null`, esto indica que otra aplicación está utilizando la cámara o que no se ha instalado ninguna cámara en el sistema. Para determinar si hay cámaras instaladas, utilice `Camera.names.length`. Para visualizar el panel de configuración de la cámara de Flash Player, desde el que el usuario puede seleccionar la cámara a la que `Camera.get()` debe hacer referencia, utilice `System.showSettings(3)`.



El proceso de exploración del hardware para detectar cámaras tarda un tiempo. Una vez que Flash ha detectado como mínimo una cámara, el hardware no se explora de nuevo a lo largo de la duración de la instancia del reproductor. No obstante, si Flash no encuentra ninguna cámara, la exploración se lleva a cabo cada vez que se llama a `Camera.get`. Esto resulta útil si un usuario ha olvidado conectar la cámara; si el archivo SWF proporciona un botón Intentar de nuevo que llama a `Camera.get`, Flash puede buscar la cámara sin que el usuario tenga que reiniciar la reproducción del archivo SWF.

## Ejemplo

En el ejemplo siguiente se captura y muestra vídeo localmente con un objeto Video denominado `my_video` en el escenario.

```
var my_cam = Camera.get();
my_video.attachVideo(myCam);
```

## Véase también

`Camera.index`, `Camera.muted`, `Camera.names`, `Camera.onStatus`, `Camera.setMode()`, `System.showSettings()`, `Video.attachVideo()`

## Camera.height

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.height*

### Descripción

Propiedad de sólo lectura; altura actual de la captura, expresada en píxeles. Para establecer un valor para esta propiedad, utilice [Camera.setMode\(\)](#).

### Ejemplo

En la línea de código siguiente se actualiza un cuadro de texto de la interfaz de usuario con el valor de altura actual.

```
my_txt._height = myCam.height;
```

Consulte también el ejemplo para [Camera.setMode\(\)](#).

### Véase también

[Camera.setMode\(\)](#), [Camera.width](#)

## Camera.index

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.index*

### Descripción

Propiedad de sólo lectura; número entero con base cero que especifica el índice de la cámara, según lo indicado por la matriz devuelta por [Camera.names](#).

### Ejemplo

En el ejemplo siguiente se obtiene la cámara que tiene el valor de `index`.

```
my_cam = Camera.get(index);
```

### Véase también

[Camera.get\(\)](#), [Camera.names](#)

## Camera.motionLevel

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.motionLevel*

### Descripción

Propiedad de sólo lectura; valor numérico que especifica la cantidad de movimiento necesaria para invocar a `Camera.onActivity(true)`. El rango de valores aceptables oscila entre 0 y 100, siendo 50 el valor predeterminado.

El vídeo puede visualizarse independientemente del valor de la propiedad `motionLevel`. Para más información, consulte `Camera.setMotionLevel()`.

### Véase también

`Camera.activityLevel`, `Camera.onActivity`, `Camera.onStatus`, `Camera.setMotionLevel()`

## Camera.motionTimeOut

### Disponibilidad

Flash Player 6.

### Sintaxis

`active_cam.motionTimeOut`

### Descripción

Propiedad de sólo lectura; número de milisegundos que deben transcurrir entre el momento en que la cámara deja de detectar movimiento y el momento en que se invoca a `Camera.onActivity(false)`. El valor predeterminado es 2000 (2 segundos).

Para establecer este valor, utilice `Camera.setMotionLevel()`.

### Ejemplo

En el ejemplo siguiente se establece el número de milisegundos que deben transcurrir entre el momento en que la cámara deja de detectar movimiento y el momento en que se invoca a `Camera.onActivity(false)` en 1000 milisegundos, o 1 segundo.

```
if(my_cam.motionTimeOut >= 1000){  
    my_cam.setMotionLevel(myCam.motionLevel, 1000);  
}
```

### Véase también

`Camera.onActivity`, `Camera.setMotionLevel()`

## Camera.muted

### Disponibilidad

Flash Player 6.

### Sintaxis

`active_cam.muted`

### Descripción

Propiedad de sólo lectura; número booleano que especifica si el usuario ha denegado el acceso a la cámara (`true`) o ha permitido el acceso (`false`) en el panel de configuración de la confidencialidad de Flash Player. Si se modifica este valor, se invoca a `Camera.onStatus`. Para más información, consulte `Camera.get()`.

### Véase también

`Camera.get()`, `Camera.onStatus`

## Camera.name

### Disponibilidad

Flash Player 6.

### Sintaxis

`active_cam.name`

### Descripción

Propiedad de sólo lectura; cadena que especifica el nombre de la cámara actual, según la devuelve el hardware de la cámara.

### Ejemplo

En el ejemplo siguiente se muestra el nombre de la cámara predeterminada en el panel Salida. En Windows, este nombre coincide con el nombre de dispositivo indicado en la hoja de propiedades de escáneres y cámaras.

```
my_cam = Camera.get();
trace("El nombre de la cámara es: " + my_cam.name);
```

### Véase también

`Camera.get()`, `Camera.names`

## Camera.names

### Disponibilidad

Flash Player 6.

### Sintaxis

`Camera.names`

**Nota:** la sintaxis correcta es `Camera.names`. Para asignar el valor de retorno a una variable, utilice una sintaxis del tipo `cam_array = Camera.names`. Para determinar el nombre de la cámara actual, utilice `active_cam.name`.

## Descripción

Propiedad de clase de sólo lectura; recupera una matriz de cadenas que reflejan los nombres de todas las cámaras disponibles sin mostrar el panel de configuración de confidencialidad de Flash Player. Esta matriz se comporta del mismo modo que cualquier otra matriz de ActionScript, y proporciona de forma implícita el índice con base cero de cada cámara y el número de cámaras del sistema (mediante `Camera.names.length`). Para más información, consulte la entrada [Clase Array](#).

Cuando se llama a la propiedad `Camera.names`, debe realizarse un análisis exhaustivo del hardware, y tal vez se precisen varios segundos para crear la matriz. En la mayoría de los casos, puede utilizar simplemente la cámara predeterminada.

## Ejemplo

En el ejemplo siguiente se utiliza la cámara predeterminada a menos que exista más de una cámara disponible, en cuyo caso el usuario puede seleccionar qué cámara debe establecerse como predeterminada.

```
cam_array = Camera.names;
if (cam_array.length == 1){
    my_cam = Camera.get();
}
else {
    System.showSettings(3);
    my_cam = Camera.get();
}
```

## Véase también

[Camera.get\(\)](#), [Camera.index](#), [Camera.name](#)

# Camera.onActivity

## Disponibilidad

Flash Player 6.

## Sintaxis

```
active_cam.onActivity = function(activity) {
    // las sentencias se escriben aquí
}
```

## Parámetros

*activity* Valor booleano que se establece en `true` cuando la cámara detecta movimiento, y en `false`, cuando se detiene.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando la cámara inicia o detiene la detección de movimiento. Si desea responder a este controlador de eventos, debe crear una función para procesar su valor de *activity*.



Para especificar la cantidad de movimiento necesaria para invocar a `Camera.onActivity(true)` y el tiempo que debe transcurrir sin actividad antes de invocar a `Camera.onActivity(false)`, utilice `Camera.setMotionLevel()`.

### Ejemplo

En el ejemplo siguiente se muestra `true` o `false` en el panel Salida cuando la cámara inicia o detiene la detección de movimiento.

```
// Se presupone que hay un objeto Video denominado "myVideoObject" en el
// escenario
my_cam = Camera.get();
myVideoObject.attachVideo(my_cam);
my_cam.setMotionLevel(10, 500);
my_cam.onActivity = function(mode)
{
    trace(mode);
}
```

### Véase también

[Camera.onActivity](#), [Camera.setMotionLevel\(\)](#)

## Camera.onStatus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
active_cam.onStatus = function(infoObject) {
    // las sentencias se escriben aquí
}
```

### Parámetros

*infoObject*    Parámetro definido de acuerdo con el mensaje de estado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el usuario permite o deniega el acceso a la cámara. Si desea responder a este controlador de eventos, debe crear una función para procesar el objeto de información generado por la cámara.

Si un archivo SWF intenta acceder a la cámara, Flash Player muestra un cuadro de diálogo de confidencialidad en el que el usuario puede seleccionar si desea permitir o denegar el acceso.

- Si el usuario permite el acceso, la propiedad `Camera.muted` se establece en `false`, y este controlador se invoca con un objeto de información cuya propiedad `code` es "Camera.Unmuted" y cuya propiedad `level` es "Status".
- Si el usuario deniega el acceso, la propiedad `Camera.muted` se establece en `true`, y este controlador se invoca con un objeto de información cuya propiedad `code` es "Camera.Muted" y cuya propiedad `level` es "Status".

Para determinar si el usuario ha denegado o permitido el acceso a la cámara sin procesar este controlador de eventos, utilice la propiedad `Camera.muted`.

**Nota:** Si el usuario decide permitir o denegar el acceso de forma permanente para todos los archivos SWF de un dominio determinado, este controlador no se invoca para los archivos SWF de dicho dominio a menos que el usuario cambie más adelante la configuración de la confidencialidad. Para más información, consulte `Camera.get()`.

### Ejemplo

El controlador de eventos siguiente muestra un mensaje siempre que el usuario permita o deniegue el acceso a la cámara.

```
myCam = Camera.get();
myVideoObject.attachVideo(myCam);
myCam.onStatus = function(infoMsg) {

    if(infoMsg.code == "Camera.Muted"){
        trace("El usuario deniega el acceso a la cámara");
    }
    else
        trace("El usuario permite el acceso a la cámara");
}
// Cambiar el valor de permitir o denegar para invocar la función
System.showSettings(0);
```

### Véase también

`Camera.get()`, `Camera.muted`

## Camera.quality

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.quality*

### Descripción

Propiedad de sólo lectura; número entero que especifica el nivel necesario de calidad de la imagen, según lo determina la cantidad de compresión que se aplica a cada fotograma de vídeo. Los valores de calidad aceptables oscilan entre 1 (calidad inferior, compresión máxima) y 100 (calidad superior, sin compresión). El valor predeterminado es 0, que indica que la calidad de la imagen puede variar según sea necesario para evitar que se sobrepase el ancho de banda disponible.

### Véase también

`Camera.setQuality()`

## Camera.setMode()

### Disponibilidad

Flash Player 6.

### Sintaxis

*active\_cam.setMode(width, height, fps [,favorSize])*

## Parámetros

*width* Anchura de captura solicitada, expresada en píxeles. El valor predeterminado es 160.

*height* Altura de captura solicitada, expresada en píxeles. El valor predeterminado es 120.

*fps* Velocidad solicitada a la que la cámara debería capturar datos, expresada en fotogramas por segundo. El valor predeterminado es 15.

*favorSize* Opcional: valor booleano que especifica cómo manipular la anchura, la altura y la velocidad de fotogramas, si la cámara no tiene un modo nativo que cumpla los requisitos especificados. El valor predeterminado es `true`, lo que significa que se da prioridad a mantener el tamaño de captura; si se utiliza este parámetro, se selecciona el modo que más se acerca a los valores de *width* y *height*, aun cuando hacerlo afecte negativamente al rendimiento mediante la reducción de la velocidad de fotogramas. Para optimizar la velocidad de fotogramas a expensas de la altura y la anchura, pase `false` para el parámetro *favorSize*.

## Valor devuelto

Ninguno.

## Descripción

Método; establece el modo de captura de la cámara en el modo nativo que mejor cumple los requisitos especificados. Si la cámara no tiene ningún modo nativo que coincida con todos los parámetros pasados, Flash selecciona el modo de captura que mejor sintetice el modo solicitado. Esta manipulación puede implicar el recorte de la imagen y la eliminación de fotogramas.

De forma predeterminada, Flash elimina fotogramas según sea necesario para mantener el tamaño de la imagen. Para minimizar el número de fotogramas que se eliminan, aun cuando ello signifique reducir el tamaño de la imagen, pase `false` para el parámetro *favorSize*.

Si selecciona un modo nativo, Flash intenta mantener la proporción solicitada siempre que sea posible. Por ejemplo, si emite el comando `active_cam.setMode(400,400,30)`, y los valores de anchura y altura máxima disponibles en la cámara son 320 y 288, Flash establece la altura y la anchura en 288. Al establecer ambas propiedades en el mismo valor, Flash mantiene la proporción 1:1 solicitada.

Para determinar los valores que se asignan a estas propiedades una vez que Flash haya seleccionado el modo que más se acerque a los valores solicitados, utilice `Camera.width`, `Camera.height` y `Camera.fps`.

## Ejemplo

En el ejemplo siguiente se establece la anchura, la altura y la velocidad en fps en función de los datos especificados por el usuario si el usuario hace clic en el botón. El parámetro opcional *favorSize* no se incluye, porque el valor predeterminado, `true`, proporcionará los valores más cercanos a las preferencias del usuario sin sacrificar por ello la calidad de la imagen, aunque en ese caso sí es posible que se sacrifique la velocidad de fps. A continuación, se actualiza la interfaz de usuario con la nueva configuración.

```

on (press)
{
    // Establecer la anchura, la altura y la velocidad de fps en los valores
    // especificados por el usuario.
    _root.myCam.setMode(txt_width, my_txt._height, txt_fps);

    // Actualizar los campos de texto del usuario con la nueva configuración.
    _root.txt_width = myCam.width;
    _root.txt_height = myCam.height;
    _root.txt_fps = myCam.fps;
}

```

#### Véase también

[Camera.currentFps](#), [Camera.fps](#), [Camera.height](#), [Camera.width](#)

## Camera.setMotionLevel()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
active_cam.setMotionLevel(sensitivity [, timeout])
```

### Parámetros

*sensitivity* Valor numérico que especifica la cantidad de movimiento necesaria para invocar a [Camera.onActivity](#)(true). El rango de valores aceptables oscila entre 0 y 100, siendo 50 el valor predeterminado.

*timeout* Parámetro numérico opcional que especifica cuántos milisegundos deben transcurrir sin actividad antes de que Flash considere que se ha detenido la actividad e invoque al controlador de eventos [Camera.onActivity](#)(false). El valor predeterminado es 2000 (2 segundos).

### Valor devuelto

Ninguno.

### Descripción

Método; especifica el movimiento necesario para invocar a [Camera.onActivity](#)(true). Opcionalmente, establece el número de milisegundos que deben transcurrir sin actividad antes de que Flash considere que el movimiento se ha detenido e invoque [Camera.onActivity](#)(false).

**Nota:** el vídeo puede visualizarse independientemente del valor del parámetro *sensitivity*. Este parámetro sólo determina en qué momento y circunstancias se invoca [Camera.onActivity](#), pero no si el vídeo se está capturando o visualizando.

- A fin de evitar que la cámara detecte movimiento, pase el valor 100 para *sensitivity*; [Camera.onActivity](#) no se invoca nunca. Este valor probablemente sólo lo utilice para realizar pruebas; por ejemplo, para desactivar temporalmente las acciones que deben tener lugar si se invoca [Camera.onActivity](#).
- Para determinar la cantidad de movimiento que está detectando la cámara actualmente, utilice la propiedad [Camera.activityLevel](#).

Los valores de sensibilidad al movimiento se corresponden directamente con los valores de actividad. La ausencia total de movimiento se indica mediante el valor de actividad 0. Un movimiento constante se indica mediante el valor de actividad 100. El valor de actividad es inferior al valor de sensibilidad si no se mueve; en el caso de moverse, los valores de actividad sobrepasan a menudo el valor de sensibilidad al movimiento.

Este método es parecido en su finalidad a `Microphone.setSilenceLevel()`; ambos métodos se utilizan para especificar cuándo debe invocarse el controlador de eventos `onActivity`. No obstante, estos métodos tienen un impacto notablemente distinto en los flujos que se publican:

- `Microphone.setSilenceLevel()` está concebido para optimizar el ancho de banda. Cuando se considera que el flujo de audio está silenciado, no se envían datos de audio. En su lugar, se envía un único mensaje en el que se indica que se ha iniciado el silencio.
- `Camera.setMotionLevel()` está concebido para detectar movimiento y no tiene ningún efecto sobre el uso del ancho de banda. Aun cuando un flujo de vídeo no detecte movimiento, se enviará vídeo.

### Ejemplo

En el ejemplo siguiente se envían mensajes al panel Salida cuando se inicia o detiene la actividad de vídeo. Cambie el valor de sensibilidad al movimiento de 30 a un valor superior o inferior para ver cómo distintos valores afectan a la detección del movimiento.

```
// Se presupone que hay un objeto Video denominado "myVideoObject" en el
// escenario
c = Camera.get();
x = 0;
function motion(mode)
{
    trace(x + ": " + mode);
    x++;
}
c.onActivity = function(mode) {motion(mode)};
c.setMotionLevel(30, 500);
myVideoObject.attachVideo(c);
```

### Véase también

[Camera.activityLevel](#), [Camera.motionLevel](#), [Camera.motionTimeout](#), [Camera.onActivity](#)

## Camera.setQuality()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
active_cam.setQuality(bandwidth, frameQuality)
```

### Parámetros

*bandwidth* Número entero que especifica la cantidad máxima de ancho de banda que puede utilizar la salida de vídeo actual, expresada en bytes por segundo. Para especificar que un vídeo Flash pueda utilizar todo el ancho de banda que sea necesario para mantener el valor de *frameQuality*, pase 0 para *bandwidth*. El valor predeterminado es 16384.

*frameQuality* Número entero que especifica el nivel necesario de calidad de la imagen, según lo determina la cantidad de compresión que se aplique a cada fotograma de vídeo. El rango de valores aceptables oscila entre 1 (calidad inferior, compresión máxima) y 100 (calidad superior, sin compresión). Para especificar que la calidad de la imagen puede variar según sea necesario para evitar que se sobrepase el ancho de banda, pase 0 para *frameQuality*. El valor predeterminado es 0.

### Valor devuelto

Ninguno.

### Descripción

Método; establece la cantidad máxima de ancho de banda por segundo o la calidad de imagen necesaria de la salida de vídeo actual. Este método generalmente sólo puede aplicarse si se transmite vídeo mediante Flash Communication Server.

Utilice este método para especificar qué elemento de salida de vídeo es más importante para la aplicación: el uso de ancho de banda o la calidad de la imagen.

- Para indicar que el uso del ancho de banda es prioritario, pase un valor para *bandwidth* y 0 para *frameQuality*. Flash transmitirá el vídeo a la calidad máxima posible utilizando el ancho de banda especificado. Si es necesario, Flash reducirá la calidad de la imagen para evitar que se sobrepase el ancho de banda especificado. Por lo general, cuando aumenta el movimiento, disminuye la calidad.
- Para indicar que la calidad es prioritaria, pase el valor 0 para *bandwidth* y un valor numérico para *frameQuality*. Flash utilizará todo el ancho de banda que sea necesario para mantener la calidad especificada. Si es necesario, Flash reducirá la velocidad de fotogramas para mantener la calidad de la imagen. Por lo general, cuando aumenta el movimiento, también aumenta el ancho de banda.
- Para especificar que el ancho de banda y la calidad tienen la misma importancia, pase un valor numérico para ambos parámetros. Flash transmitirá vídeo que cumpla la calidad especificada y que no sobrepase el ancho de banda indicado. Si es necesario, Flash reducirá la velocidad de fotogramas para mantener la calidad de la imagen sin sobrepasar el ancho de banda especificado.

### Ejemplo

En los ejemplos siguientes se ilustra cómo utilizar este método para controlar el uso del ancho de banda y la calidad de la imagen.

```
// Garantizar que no se utilizan más de 8192 (8K/segundo) para enviar vídeo
active_cam.setQuality(8192,0);
```

```
// Garantizar que no se utilizan más de 8192 (8K/segundo) para enviar vídeo
// con una calidad mínima de 50
active_cam.setQuality(8192,50);
```

```
// Garantizar una calidad mínima de 50, independientemente del ancho de banda
necesario
active_cam.setQuality(0,50);
```

### Véase también

[Camera.bandwidth](#), [Camera.quality](#)

# Camera.width

## Disponibilidad

Flash Player 6.

## Sintaxis

*active\_cam.width*

## Descripción

Propiedad de sólo lectura; anchura de la captura actual, expresada en píxeles. Para establecer el valor que desee utilizar para esta propiedad, utilice [Camera.setMode\(\)](#).

## Ejemplo

En la línea de código siguiente se actualiza un cuadro de texto en la interfaz de usuario con el valor de anchura actual.

```
myTextField.text=myCam.width;
```

Consulte también el ejemplo para [Camera.setMode\(\)](#).

## Véase también

[Camera.height](#)

# case

## Disponibilidad

Flash Player 4.

## Sintaxis

*case expression: statements*

## Parámetros

*expression*    Cualquier expresión.

*statements*    Cualquier número de sentencias.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; define una condición para la acción `switch`. Las sentencias del parámetro *statements* se ejecutan si el parámetro *expression* que sigue a la palabra clave `case` es igual al parámetro *expression* de la acción `switch` utilizando la igualdad estricta (`===`)

Si utiliza la acción `case` fuera de una sentencia `switch`, se genera un error y el script no se compila.

## Véase también

[break](#), [default](#), [===](#) (igualdad estricta), [switch](#)

# chr

## Disponibilidad

Flash Player 4. Esta función se eliminó en Flash 5 y se sustituyó por `String.fromCharCode()`.

## Sintaxis

`chr(number)`

## Parámetros

*number* Número de código ASCII.

## Valor devuelto

Ninguno.

## Descripción

Función de cadena; convierte los números de código ASCII en caracteres.

## Ejemplo

En el ejemplo siguiente el número 65 se convierte en la letra *A* y se asigna a la variable `myVar`.

```
myVar = chr(65);
```

## Véase también

`String.fromCharCode()`

# class

## Disponibilidad

Flash Player 6.

## Sintaxis

```
[dynamic] class className [ extends superClass ]  
                        [ implements interfaceName [, interfaceName... ] ]  
{  
    // la definición de clase  
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Parámetros

*className* El nombre completo de la clase.

*superClass* Opcional; el nombre de la clase de la que *className* se extiende (de la que se hereda).

*interfaceName* Opcional; el nombre de la interfaz cuyos métodos debe implementar *className*.



## Descripción

Sentencia; define una clase personalizada, que permite crear instancias de objetos que comparten métodos y propiedades que define el usuario. Por ejemplo, si está desarrollando un sistema de seguimiento de facturas, podría crear una clase de factura que definiera todos los métodos y las propiedades que cada factura debería tener. Luego podría usar el comando `new factura()` para crear objetos de factura.

El nombre de la clase debe ser el mismo que el del archivo externo que contiene la clase. Por ejemplo, si asigna a una clase el nombre *Student*, el archivo que define la clase debe llamarse *Student.as*.

El nombre de la clase debe estar completo en el archivo en el que está declarada; es decir, debe reflejar el directorio en el que está almacenada. Por ejemplo, si crea una clase llamada *RequiredClass*, que esté almacenada en el directorio `myClasses/education/curriculum`, debe declarar la clase en el archivo *RequiredClass.as* de esta forma:

```
class myClasses.education.curriculum.RequiredClass {  
}
```

Por este motivo, es aconsejable planificar una estructura de directorios antes de empezar a crear clases. De otro modo, si decide mover archivos de clases después de crearlos, deberá modificar las sentencias de declaración de clases para reflejar su nueva ubicación.

Las definiciones de clase no pueden anidarse; es decir, no pueden definirse clases adicionales en una definición de clase.

Para indicar que los objetos pueden añadir y acceder a propiedades dinámicas en tiempo de ejecución, ponga la palabra clave *dynamic* delante de la sentencia de clase. Para crear clases basadas en las interfaces, utilice la palabra clave *implements*. Para crear subclases de una clase, utilice la palabra clave *extends*. Una clase puede ampliar solamente una clase, pero puede implementar varias interfaces. Puede utilizar las palabras clave *implements* y *extends* en una única sentencia.

```
class C implements Interface_i, Interface_j    // Correcto  
class C extends Class_d implements Interface_i, Interface_j    // Correcto  
class C extends Class_d, Class_e    // Incorrecto
```

Para más información, consulte [“Creación y utilización de clases” en la página 167](#).

## Ejemplo

En el ejemplo siguiente se crea una clase denominada *Plant*. Su constructor toma dos parámetros.

```
// Nombre de archivo Plant.as  
class Plant {  
    // Definir los tipos y los nombres de propiedad  
    var leafType:String;  
    var bloomSeason:String;  
    // La línea siguiente es un constructor  
    // porque tiene el mismo nombre que la clase  
    function Plant (param_leafType:String, param_bloomSeason:String) {  
        // Asignar valores pasados a las propiedades cuando se crea el nuevo  
        // objeto Plant  
        leafType = param_leafType;  
        bloomSeason = param_bloomSeason;  
    }  
    // Crear métodos para volver a ejecutar los valores de propiedad, porque  
    // se recomienda no crear referencias directamente a una propiedad de una  
    // clase
```

```

    function getLeafType():String {return leafType};
    function getBloomSeason():String {return bloomSeason};
}

```

En un archivo de script externo o en el panel Acciones, utilice el operador `new` para crear un objeto `Plant`.

```

var pineTree:Plant = new Plant("Perenne","N/A");
// Confirmar que los parámetros se han pasado correctamente
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());

```

#### Véase también

[dynamic](#), [extends](#), [implements](#), [interface](#), [new](#)

## clearInterval()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
clearInterval( intervalID )
```

### Parámetros

*intervalID* Objeto devuelto de una llamada a [setInterval\(\)](#).

### Valor devuelto

Ninguno.

### Descripción

Función; borra una llamada a [setInterval\(\)](#).

### Ejemplo

En el ejemplo siguiente se establece, en primer lugar, una llamada de intervalo y, a continuación, se cancela:

```

function callback() {
    trace("intervalo llamado");
}
var intervalID;
intervalID = setInterval( callback, 1000 );

// posteriormente
clearInterval( intervalID );

```

#### Véase también

[setInterval\(\)](#)

# Clase Color

## Disponibilidad

Flash Player 5.

## Descripción

La clase Color permite establecer el valor de color RGB y de transformación del color de los clips de película y recuperar esos valores una vez establecidos.

Debe utilizar un constructor `new Color()` para crear un objeto Color antes de llamar a sus métodos.

## Resumen de métodos para la clase Color

Método	Descripción
<code>Color.getRGB()</code>	Devuelve el valor numérico RGB establecido por la última llamada <code>setRGB()</code> .
<code>Color.getTransform()</code>	Devuelve la información de transformación establecida por la última llamada <code>setTransform()</code> .
<code>Color.setRGB()</code>	Establece la representación hexadecimal del valor RGB para un objeto Color.
<code>Color.setTransform()</code>	Establece la transformación de color para un objeto Color.

## Constructor para la clase Color

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Color(target)
```

### Parámetros

*target* Nombre de instancia de un clip de película.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto Color para el clip de película especificado por el parámetro *target*. Puede utilizar los métodos de ese objeto Color para cambiar el color de todo el clip de película de destino.

### Ejemplo

En el ejemplo siguiente se crea un objeto Color denominado `my_color` para el clip de película `my_mc` y se establece su valor RGB:

```
my_color = new Color(my_mc);  
my_color.setRGB(0xff9933);
```

## Color.getRGB()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_color.getRGB()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número que representa el valor numérico RGB para el color especificado.

### Descripción

Método; devuelve los valores numéricos establecidos por la última llamada `setRGB()`.

### Ejemplo

En el código siguiente se recupera el valor RGB del objeto `Color my_color`, se convierte en una cadena hexadecimal y se le asigna una variable `value`.

```
value = my_color.getRGB().toString(16);
```

### Véase también

[Color.setRGB\(\)](#)

## Color.getTransform()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_color.getTransform()
```

### Parámetros

Ninguno.

### Valor devuelto

Un objeto cuyas propiedades contienen los valores de desplazamiento y de porcentaje actuales para el color especificado.

### Descripción

Método; devuelve el valor de transformación establecido por la última llamada

[Color.setTransform\(\)](#).

### Véase también

[Color.setTransform\(\)](#)

## Color.setRGB()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_color.setRGB(0xRRGGBB)
```

### Parámetros

*0xRRGGBB* Color hexadecimal o RGB que se va a establecer. *RR* (rojo), *GG* (verde) y *BB* (azul) constan de dos dígitos hexadecimales respectivamente que especifican el desplazamiento de cada componente de color. *0x* indica al compilador de ActionScript que el número es un valor hexadecimal.

### Descripción

Método; especifica un color RGB para un objeto Color. Al llamar a este método se suplantán los valores anteriores de [Color.setTransform\(\)](#).

### Valor devuelto

Ninguno.

### Ejemplo

En este ejemplo se establece el valor de color RGB para el clip de película *my\_mc*. Para ver este código en funcionamiento, coloque un clip de película en el escenario con el nombre de instancia *my\_mc*. A continuación, coloque el código siguiente en el fotograma 1 de la línea de tiempo principal y elija Control > Probar película.

```
my_color = new Color(my_mc);  
my_color.setRGB(0x993366);
```

### Véase también

[Color.setTransform\(\)](#)

## Color.setTransform()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_color.setTransform(colorTransformObject)
```

### Parámetros

*colorTransformObject* Objeto creado con el constructor `new Object`. Esta instancia de la [Clase Object](#) debe tener las propiedades siguientes que especifican los valores de transformación del color: *ra*, *rb*, *ga*, *gb*, *ba*, *bb*, *aa*, *ab*. Estas propiedades se explican a continuación.

### Valor devuelto

Ninguno.

## Descripción

Método; establece la información de transformación del color para un objeto Color. El parámetro *colorTransformObject* es un objeto genérico que se crea a partir del constructor `new Object`. Dispone de parámetros que especifican el porcentaje y los valores de desplazamiento de los componentes rojo (R), verde (G), azul (B) y transparencia alfa (A) de un color, especificados en formato *0xRRGGBBAA*.

Los parámetros de un objeto de transformación del color corresponden a la configuración del cuadro de diálogo Efecto avanzado y se definen del modo siguiente:

- *ra* es el porcentaje del componente rojo (de -100 a 100).
- *rb* es el desplazamiento del componente rojo (de -255 a 255).
- *ga* es el porcentaje del componente verde (de -100 a 100).
- *gb* es el desplazamiento del componente verde (de -255 a 255).
- *ba* es el porcentaje del componente azul (de -100 a 100).
- *bb* es el desplazamiento del componente azul (de -255 a 255).
- *aa* es el porcentaje de transparencia alfa (de -100 a 100).
- *ab* es el desplazamiento de transparencia alfa (de -255 a 255).

Para crear un parámetro *colorTransformObject* debe seguir el procedimiento que se muestra a continuación:

```
myColorTransform = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

También puede utilizar la sintaxis siguiente para crear un parámetro *colorTransformObject*:

```
myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112', ba: '12', bb:
'90', aa: '40', ab: '70' }
```

## Ejemplo

En este ejemplo se crea un nuevo objeto Color para un archivo SWF de destino, se crea un objeto genérico denominado *myColorTransform* con las propiedades definidas anteriormente y se utiliza el método `setTransform()` para pasar el parámetro *colorTransformObject* a un objeto Color. Para utilizar este código en un documento de Flash (FLA), colóquelo en el fotograma 1 de la línea de tiempo principal y coloque un clip de película en el escenario con el nombre de instancia *my\_mc*, como se indica en el código siguiente:

```
// Crear un objeto Color denominado my_color para el destino my_mc
my_color = new Color(my_mc);
// Crear un objeto de transformación del color denominado myColorTransform con
// el objeto genérico Object
myColorTransform = new Object();
// Establecer los valores de myColorTransform
myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112', ba: '12', bb:
'90', aa: '40', ab: '70' };
// Asociar el objeto de transformación del color con el objeto Color
// creado para my_mc
my_color.setTransform(myColorTransform);
```

# Clase ContextMenu

## Disponibilidad

Flash Player 7.

## Descripción

La clase `ContextMenu` proporciona el control de tiempo de ejecución de los elementos del menú contextual de Flash Player, que aparece al hacer clic con el botón derecho del ratón (Windows) o presionar la tecla Control y hacer clic (Macintosh) en Flash Player. Los métodos y las propiedades de la clase `ContextMenu` pueden utilizarse para agregar elementos de menú personalizados, controlar la visualización de los elementos de menú contextual incorporados (por ejemplo, Acercar e Imprimir) o crear copias de menús.

Puede asociarse un objeto `ContextMenu` a un botón, un clip de película o un objeto de campo de texto específico o a nivel de toda la película. Para ello, se utiliza la propiedad `menu` de las clases `Button`, `MovieClip` o `TextField`. Para más información sobre la propiedad `menu`, consulte [Button.menu](#), [MovieClip.menu](#) y [TextField.menu](#).

Para agregar nuevos elementos a un objeto `ContextMenu`, debe crear un objeto `ContextMenuItems` y, a continuación, agregar dicho objeto a la matriz `ContextMenu.customItems`. Para más información sobre la creación de elementos de menú contextual, consulte la entrada [Clase ContextMenuItems](#).

Flash Player tiene tres tipos de menús contextuales: el menú estándar (que aparece al hacer clic con el botón derecho del ratón en Flash Player), el menú de edición (que aparece al hacer clic con el botón derecho del ratón en un campo de texto seleccionable o editable) y un menú de error (que aparece cuando un archivo SWF no puede cargarse en Flash Player). Con la clase `ContextMenu` sólo pueden modificarse el menú estándar y el de edición.

Los elementos de menú personalizados siempre aparecen en la parte superior del menú contextual de Flash Player, por encima de cualquier elemento de menú incorporado visible; los elementos de menú incorporados y los personalizados se separan mediante una barra de división. Un menú contextual no puede tener más de 15 elementos de menú personalizados.

Debe utilizar el constructor `new ContextMenu()` para crear un objeto `ContextMenu` antes de llamar a sus métodos.

## Resumen de métodos para la clase ContextMenu

Método	Descripción
<a href="#">ContextMenu.copy()</a>	Devuelve una copia del objeto <code>ContextMenu</code> especificado.
<a href="#">ContextMenu.hideBuiltInItems()</a>	Oculto la mayoría de los elementos incorporados en el menú contextual de Flash Player.

## Resumen de propiedades para la clase `ContextMenu`

Propiedad	Descripción
<code>ContextMenu.builtInItems</code>	Objeto cuyos miembros corresponden a elementos incorporados del menú contextual.
<code>ContextMenu.customItems</code>	Matriz, que de forma predeterminada no está definida, que contiene objetos <code>ContextMenuItem</code> .

## Resumen del controlador de eventos para la clase `ContextMenu`

Propiedad	Descripción
<code>ContextMenu.onSelect</code>	Invocado antes de mostrar el menú.

## Constructor para la clase `ContextMenu`

### Disponibilidad

Flash Player 7.

### Sintaxis

```
new ContextMenu ([callbackFunction])
```

### Parámetros

*callbackFunction* Referencia a una función que se llama cuando el usuario hace clic con el botón derecho del ratón o hace clic con la tecla Control presionada antes de visualizar el menú. Este parámetro es opcional.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto `ContextMenu` nuevo. De manera opcional, puede especificarse un identificador para un controlador de eventos al crear el objeto. La función especificada se llama cuando el usuario invoca el menú contextual, pero *antes* de que el menú se visualice. Esto es muy útil para personalizar el contenido del menú en función del estado de la aplicación o del tipo de objeto (clip de película, campo de texto o botón) en el que el usuario hace clic con el botón derecho del ratón o hace clic con la tecla Control presionada. Para ver un ejemplo de creación de un controlador de eventos, véase [ContextMenu.onSelect](#).

### Ejemplo

En el ejemplo siguiente se ocultan los objetos incorporados del menú contextual. No obstante, los elementos Configuración y Acerca de siguen apareciendo, porque no pueden desactivarse.

```
var newMenu = new ContextMenu();
newMenu.hideBuiltInItems();
_root.menu = newMenu;
```



En este ejemplo, el controlador de eventos especificado, `menuHandler`, activa o desactiva el elemento de menú personalizado (utilizando la matriz `ContextMenu.customItems`) en función del valor de una variable booleana denominada `showItem`. Si el valor es `false`, el elemento de menú personalizado se desactiva; de lo contrario, se activa.

```
var showItem = false; // Cambiar este valor a true para ver su efecto
my_cm = new ContextMenu(menuHandler);
my_cm.customItems.push(new ContextMenuItem("Hola", itemHandler));
function menuHandler(obj, menuObj) {
    if (showItem == false) {
        menuObj.customItems[0].enabled = false;
    } else {
        menuObj.customItems[0].enabled = true;
    }
}
function itemHandler(obj, item) {
}
_root.menu = my_cm;
```

#### Véase también

[Button.menu](#), [ContextMenu.onSelect](#), [ContextMenu.customItems](#),  
[ContextMenu.hideBuiltInItems\(\)](#), [MovieClip.menu](#), [TextField.menu](#)

## ContextMenu.builtInItems

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_cm.builtInItems
```

### Descripción

Propiedad; objeto que tiene las propiedades booleanas siguientes: `save`, `zoom`, `quality`, `play`, `loop`, `rewind`, `forward_back` y `print`. Si se establecen estas variables en `false`, se eliminan los elementos de menú correspondientes del objeto `ContextMenu` especificado. Estas propiedades pueden enumerarse y están establecidas en `true` de forma predeterminada.

### Ejemplo

En este ejemplo, se desactivan los elementos de menú incorporados Calidad e Imprimir para el objeto `ContextMenu` denominado `my_cm`, que se ha asociado a la línea de tiempo raíz del archivo SWF.

```
var my_cm = new ContextMenu();
my_cm.builtInItems.quality=false;
my_cm.builtInItems.print=false;
_root.menu = my_cm;
```

En el ejemplo siguiente, un bucle `for...in` enumera todos los nombres y valores para los elementos de menú incorporados del objeto de menú `ContextMenu`, `my_cm`.

```
my_cm = new ContextMenu();
for(eachProp in my_cm.builtInItems) {
    var propName = eachProp;
    var propValue = my_cm.builtInItems[propName];
    trace(propName + ": " + propValue);
}
```

## ContextMenu.copy()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_cm.copy()
```

### Parámetros

Ninguno.

### Valor devuelto

Un objeto ContextMenu.

### Descripción

Método; crea una copia del objeto ContextMenu especificado. La copia hereda todas las propiedades del objeto de menú original.

### Ejemplo

En este ejemplo se crea una copia del objeto de menú ContextMenu denominado `my_cm` cuyos elementos de menú incorporados están ocultos, y se añade un elemento de menú con el texto “Guardar...”. A continuación, se crea una copia de `my_cm` y se la asigna a la variable `clone_cm`, que hereda todas las propiedades del menú original.

```
my_cm = new ContextMenu();
my_cm.hideBuiltInItems();
my_cm.customItems.push(new ContextMenuItem("Guardar...", saveHandler);
function saveHandler (obj, menuItem) {
    saveDocument(); // Función personalizada (no se muestra)
}
clone_cm = my_cm.copy();
```

## ContextMenu.customItems

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_cm.customItems
```

### Descripción

Propiedad; matriz de objetos ContextMenuItem. Cada objeto de la matriz representa un elemento de menú contextual que se ha definido. Utilice esta propiedad para añadir, eliminar o modificar los elementos de menú personalizados.

Para añadir elementos de menú, en primer lugar debe crear un objeto ContextMenuItem nuevo y, a continuación, añadirlo a la matriz `menu_mc.customItems` (utilizando `Array.push()`, por ejemplo). Para más información sobre cómo crear elementos de menú, consulte la entrada [Clase ContextMenuItem](#).

## Ejemplo

En el ejemplo siguiente se crea un nuevo elemento de menú personalizado denominado `menuItem_cm` con el texto “Enviar correo electrónico” y un controlador callback denominado `emailHandler` (no se muestra). A continuación, se añade el elemento de menú al objeto `ContextMenu`, `my_cm`, utilizando la matriz `customItems`. Por último, el nuevo menú se asocia a un clip de película denominado `email_mc`.

```
var my_cm = new ContextMenu();
var menuItem_cm = new ContextMenuItem("Enviar correo electrónico",
    emailHandler);
my_cm.customItems.push(menuItem_cm);
email_mc.menu = my_cm;
```

## Véase también

[Button.menu](#), [Clase ContextMenu](#), [MovieClip.menu](#), [TextField.menu](#)

# ContextMenu.hideBuiltInItems()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_cm.hideBuiltInItems()
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Método; oculta todos los elementos de menú incorporados (con la excepción de Configuración) en el objeto `ContextMenu` especificado. Si se está ejecutando el Reproductor de depuración de Flash, se muestra el elemento de menú Depuración, aunque aparece atenuado en archivos SWF en los que no se haya activado la depuración remota.

Este método oculta solamente los elementos de menú que aparecen en el menú contextual estándar; no tiene ningún efecto sobre los elementos que aparecen en el menú de edición o de error. Para más información sobre los distintos tipos de menú, consulte la entrada [Clase ContextMenu](#).

Este método establece todos los miembros booleanos de `my_cm.builtInItems` en `false`. Puede establecer de forma selectiva que un elemento incorporado pase a ser visible estableciendo su miembro correspondiente de `my_cm.builtInItems` en `true` (como se muestra en el ejemplo siguiente).

## Ejemplo

En el ejemplo siguiente se crea un objeto `ContextMenu` nuevo denominado `my_cm` cuyos elementos de menú incorporados están ocultos, con la excepción de Imprimir. El objeto de menú se asocia a la línea de tiempo raíz.

```
my_cm = new ContextMenu();
my_cm.hideBuiltInItems();
my_cm.builtInItems.print = true;
_root.menu = my_cm;
```

## ContextMenu.onSelect

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_cm.onSelect = function (item:Object, item_menu:ContextMenu) {
    // el código se escribe aquí
}
```

### Parámetros

*item* Referencia al objeto (clip de película, botón o campo de texto seleccionable) que se encontraba bajo el puntero del ratón al invocar el menú contextual de Flash Player y cuya propiedad `menu` está establecida en un objeto `ContextMenu` válido.

*item\_menu* Referencia al objeto `ContextMenu` asignado a la propiedad `menu` de *object*.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se llama a este controlador cuando un usuario invoca el menú contextual de Flash Player, pero antes de que se visualice el menú. De este modo es posible personalizar el contenido del menú contextual en función del estado actual de la aplicación.

También puede especificarse el controlador callback para un objeto `ContextMenu` al construir un objeto `ContextMenu` nuevo. Para más información, consulte la entrada [Clase ContextMenu](#).

## Ejemplo

En el ejemplo siguiente se determina sobre qué tipo de objeto se ha invocado el menú contextual.

```
my_cm = new ContextMenu();
menuHandler = function (obj:Object, menu:ContextMenu) {
    if(obj instanceof MovieClip) {
        trace("Clip de película: " + obj);
    }
    if(obj instanceof TextField) {
        trace("Campo de texto: " + obj);
    }
    if(obj instanceof Button) {
        trace("Botón: " + obj);
    }
}
my_cm.onSelect = menuHandler;
```

# Clase ContextMenuItem

## Disponibilidad

Flash Player 7.

## Descripción

La clase ContextMenuItem se utiliza para crear elementos de menú personalizados y visualizarlos en el menú contextual de Flash Player. Cada objeto ContextMenuItem tiene un título (texto) que se visualiza en el menú contextual y un controlador callback (una función) que se invoca al seleccionar el elemento de menú. Para añadir un nuevo elemento de menú contextual a un menú contextual, debe añadirlo a la matriz `customItems` de un objeto ContextMenu.

Puede activar o desactivar elementos de menú específicos, establecer que los elementos sean visibles o no, o cambiar un título o un controlador callback asociado a un elemento de menú.

Los elementos de menú personalizados aparecen en la parte superior del menú contextual, por encima de los elementos incorporados. Los elementos de menú personalizados siempre están separados de los elementos incorporados por una barra de división. En el menú contextual de Flash Player no pueden añadirse más de 15 elementos personalizados. Cada elemento debe contener como mínimo un carácter visible (los caracteres de control, las líneas nuevas y otros caracteres de espacio en blanco se pasan por alto). La longitud máxima de un elemento es de 100 caracteres. Los elementos que sean idénticos a algún elemento de menú incorporado, o a otro elemento de menú personalizado, se pasarán por alto, independientemente de si el elemento coincidente está visible. Los elementos de menú se comparan sin tener en cuenta las mayúsculas y minúsculas, la puntuación ni los espacios en blanco.

Un elemento personalizado no puede contener ninguna de las palabras siguientes: *Macromedia*, *Flash Player* o *Configuración*.

## Resumen de métodos para la clase ContextMenuItem

Método	Descripción
<code>ContextMenuItem.copy()</code>	Devuelve una copia del objeto ContextMenuItem especificado.

## Resumen de propiedades para la clase ContextMenuItem

Propiedad	Descripción
<code>ContextMenuItem.caption</code>	Especifica el texto que se visualiza en el elemento de menú.
<code>ContextMenuItem.enabled</code>	Especifica si el menú está activado o desactivado.
<code>ContextMenuItem.separatorBefore</code>	Especifica si debe aparecer una barra de división por encima del elemento de menú.
<code>ContextMenuItem.visible</code>	Especifica si el elemento de menú está visible o no.

## Resumen de controladores de eventos para la clase ContextMenuItem

Controlador de eventos	Descripción
<code>ContextMenuItem.onSelect</code>	Se invoca al seleccionar el elemento de menú.

## Constructor para la clase ContextMenuItem

### Disponibilidad

Flash Player 7.

### Sintaxis

```
new ContextMenuItem(caption, callbackFunction, [ separatorBefore, [ enabled, [ visible ] ] ] )
```

### Parámetros

*caption* Cadena que especifica el texto asociado a un elemento de menú.

*callbackFunction* Una función que haya definido, y a la que se llama cuando se selecciona el elemento de menú.

*separatorBefore* Valor booleano que indica si debe aparecer una barra de división por encima del elemento de menú en el menú contextual. Este parámetro es opcional; su valor predeterminado es `false`.

*enabled* Valor booleano que indica si el elemento de menú está activado o desactivado en el menú contextual. Este parámetro es opcional; su valor predeterminado es `true`.

*visible* Valor booleano que indica si el elemento de menú está visible o no. Este parámetro es opcional; su valor predeterminado es `true`.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto `ContextMenuItem` que puede añadirse a la matriz `ContextMenu.customItems`.

### Ejemplo

En este ejemplo se añaden los elementos de menú Iniciar y Detener, separados por una barra, al objeto `ContextMenu` denominado `my_cm`. Al seleccionar Iniciar en el menú contextual se llama a la función `startHandler()`; al seleccionar Detener, se llama a la función `stopHandler()`. El objeto `ContextMenu` se aplica a la línea de tiempo raíz.

```
my_cm = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Start", startHandler));
my_cm.customItems.push(new ContextMenuItem("Stop", stopHandler, true));
function stopHandler(obj, item) {
    trace("Deteniendo...");
}
function startHandler(obj, item) {
    trace("Iniciando...");
}
_root.menu = my_cm;
```

## ContextMenuItem.caption

### Disponibilidad

Flash Player 7.

### Sintaxis

*menuItem\_cmi.caption*

### Descripción

Propiedad; cadena que especifica el título (texto) del elemento de menú que se visualiza en el menú contextual.

### Ejemplo

En este ejemplo se muestra el título para el elemento de menú seleccionado (Pausar juego) en el panel Salida.

```
my_cm = new ContextMenu();
menuItem_cmi = new ContextMenuItem("Pausar juego", onPause);
my_cm.customItems.
function onPause(obj, menuItem) {
    trace("Ha seleccionado: " + menuItem.caption);
}
```

## ContextMenuItem.copy()

### Disponibilidad

Flash Player 7.

### Sintaxis

*menuItem\_cmi.copy()*;

### Valor devuelto

Un objeto ContextMenuItem.

### Descripción

Método; crea y devuelve una copia del objeto ContextMenuItem especificado. La copia incluye todas las propiedades del objeto original.

### Ejemplo

En este ejemplo se crea un objeto ContextMenuItem nuevo denominado *original\_cmi* con el texto Pausar y un controlador callback establecido para la función *onPause*. A continuación, el ejemplo crea una copia del objeto ContextMenuItem y se asigna a la variable *copy\_cmi*.

```
original_cmi = new ContextMenuItem("Pausar", onPause);
function onPause(obj, menu) {
    _root.stop();
}
original_cmi.visible = false;
copy_cmi = orig_cmi.copy();
```

## ContextMenuItem.enabled

### Disponibilidad

Flash Player 7.

### Sintaxis

```
menuItem_cmi.enabled
```

### Descripción

Propiedad; valor booleano que indica si el menú especificado está activado o desactivado. De forma predeterminada, el valor de esta propiedad es `true`.

### Ejemplo

En el ejemplo siguiente se crea un nuevo elemento de menú contextual y, a continuación, dicho elemento se desactiva.

```
var saveMenuItem = new ContextMenuItem("Guardar...", doSave);
saveMenuItem.enabled = false;
```

## ContextMenuItem.onSelect

### Disponibilidad

Flash Player 7.

### Sintaxis

```
menuItem_cmi.onSelect = function (obj, menuItem) {
    // las sentencias se escriben aquí
}
```

### Parámetros

*obj* Referencia al clip de película (o a la línea de tiempo), botón o campo de texto seleccionable (editable) en el que el usuario ha hecho clic con el botón derecho del ratón o con la tecla Control presionada.

*menuItem* Referencia al objeto ContextMenuItem seleccionado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca al seleccionar el elemento de menú especificado en el menú contextual de Flash Player. El controlador callback especificado recibe dos parámetros: *obj*, que es una referencia al objeto que se encuentra debajo del ratón cuando el usuario invoca el menú contextual de Flash Player, y *menuItem*, que es una referencia al objeto ContextMenuItem que representa el elemento de menú seleccionado.



## Ejemplo

En el ejemplo siguiente se asigna una función al controlador `onSelect` para un objeto `ContextMenuItem` denominado `start_cmi`. La función muestra el texto del elemento de menú seleccionado.

```
start_cmi.onSelect = function (obj, item) {  
    trace("Ha seleccionado: " + item.caption);  
}
```

## Véase también

[ContextMenu.onSelect](#)

# ContextMenuItem.separatorBefore

## Disponibilidad

Flash Player 7.

## Sintaxis

```
menuItem_cmi.separatorBefore
```

## Descripción

Propiedad; valor booleano que indica si debe aparecer una barra de división encima del elemento de menú especificado. De forma predeterminada, el valor de esta propiedad es `false`.

**Nota:** siempre aparece una barra de división entre los elementos de menú personalizados y los elementos incorporados.

## Ejemplo

En este ejemplo se crean tres elementos de menú: Abrir, Guardar e Imprimir. Una barra de división separa los elementos Guardar e Imprimir. A continuación, se añaden los elementos de menú a la matriz `customItems` del objeto `ContextMenu`. Por último, se asocia el menú a la línea de tiempo raíz del archivo SWF.

```
my_cm = new ContextMenu();  
open_cmi = new ContextMenuItem("Abrir", itemHandler);  
save_cmi = new ContextMenuItem("Guardar", itemHandler);  
print_cmi = new ContextMenuItem("Imprimir", itemHandler);  
print_cmi.separatorBefore = true;  
my_cm.customItems.push(open_cmi, save_cmi, print_cmi);  
function itemHandler(obj, menuItem) {  
    trace("Ha seleccionado: " + menuItem.caption);  
};  
_root.menu = my_cm;
```

## Véase también

[ContextMenu.onSelect](#)

## ContextMenuItem.visible

### Disponibilidad

Flash Player 7.

### Sintaxis

```
menuItem_cmi.visible
```

### Descripción

Propiedad; valor booleano que indica si el elemento de menú seleccionado está visible cuando se visualiza el menú contextual de Flash Player. De forma predeterminada, el valor de esta propiedad es `true`.

## continue

### Disponibilidad

Flash Player 4.

### Sintaxis

```
continue
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Sentencia; aparece dentro de varios tipos de sentencias de reproducción indefinida; funciona de forma distinta en cada tipo de reproducción.

En un bucle `while`, `continue` hace que el intérprete de Flash omita el resto del cuerpo de la reproducción y salte al principio, donde se comprueba la condición.

En un bucle `do while`, `continue` hace que el intérprete de Flash omita el resto del cuerpo del bucle y salte al final de la reproducción, donde se comprueba la condición.

En una reproducción `for`, `continue` hace que el intérprete de Flash omita el resto del cuerpo de la reproducción y salte al cálculo de la postexpresión de la reproducción `for`.

En una reproducción `for..in`, `continue` hace que el intérprete de Flash omita el resto del cuerpo de la reproducción y salte al principio del bucle, donde se procesa el siguiente valor de la enumeración.

### Véase también

`do while`, `for`, `for..in`, `while`

# Clase CustomActions

## Disponibilidad

Flash Player 6.

## Descripción

Los métodos de la clase CustomActions permiten que un archivo SWF que se reproduzca en la herramienta de edición de Flash gestione las acciones personalizadas que se registran con la herramienta de edición. Los archivos SWF pueden instalar y desinstalar acciones personalizadas, recuperar la definición XML de una acción personalizada y recuperar la lista de las acciones personalizadas registradas.

Puede utilizar estos métodos para crear archivos SWF que sean extensiones de la herramienta de edición de Flash. Este tipo de extensión podría, por ejemplo, utilizar el protocolo de aplicación de Flash para navegar por un repositorio UDDI y descargar servicios Web en la caja de herramientas Acciones.

## Resumen de métodos para la clase CustomActions

Método	Descripción
<code>CustomActions.get()</code>	Lee el contenido de un archivo de definición XML de acciones personalizadas.
<code>CustomActions.install()</code>	Instala un nuevo archivo de definición XML de acciones personalizadas.
<code>CustomActions.list()</code>	Devuelve una lista de todas las acciones personalizadas registradas.
<code>CustomActions.uninstall()</code>	Elimina un archivo de definición XML de acciones personalizadas.

## CustomActions.get()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
CustomActions.get(customActionsName)
```

### Parámetros

*customActionsName* Nombre de la definición de la acción personalizada que debe recuperarse.

### Valor devuelto

Si se localiza la definición XML de la acción personalizada, devuelve una cadena; de lo contrario, devuelve undefined.

### Descripción

Método; lee el contenido del archivo de definición XML de acciones personalizadas denominado *customActionsName*.

El nombre del archivo de definición debe ser un nombre simple, sin la extensión .xml y sin separadores de directorio (':', '/' o '\').

Si no se encuentra el archivo de definición especificado por *customActionsName*, se devuelve el valor *undefined*. Si se encuentra la definición XML de acciones personalizadas especificada por el parámetro *customActionsName*, se la lee completamente y se devuelve como cadena.

## CustomActions.install()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
CustomActions.install(customActionsName, customXMLDefinition)
```

### Parámetros

*customActionsName*    Nombre de la definición de la acción personalizada que debe instalarse.

*customXMLDefinition*    Texto de la definición XML que debe instalarse.

### Valor devuelto

Un valor booleano de *false* si se produce un error durante la instalación; en caso contrario, se devuelve un valor de *true* para indicar que se ha instalado correctamente la acción personalizada.

### Descripción

Método; instala un nuevo archivo de definición XML de acciones personalizadas indicada por el parámetro *customActionsName*. El contenido del archivo se especifica mediante la cadena *customXMLDefinition*.

El nombre del archivo de definición debe ser un nombre simple, sin la extensión .xml y sin separadores de directorio (':', '/' o '\').

Si ya existe un archivo de acciones personalizadas con el nombre *customActionsName*, se sobrescribirá.

Si no existe el directorio Configuration/ActionsPanel/CustomActions cuando se invoca este método, se creará.

## CustomActions.list()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
CustomActions.list()
```

### Parámetros

Ninguno.

### Valor devuelto

Una matriz.

### Descripción

Método; devuelve un objeto Array que contiene los nombres de todas las acciones personalizadas que se han registrado con la herramienta de edición de Flash. Los elementos de la matriz son nombres simples, sin la extensión de archivo .xml y sin separadores de directorio (por ejemplo, “:”, “/” o “\”). Si no existe ninguna acción personalizada registrada, `list()` devuelve una matriz de longitud cero. Si se produce un error, `list()` devuelve el valor `undefined`.

## CustomActions.uninstall()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
CustomActions.uninstall(customActionsName)
```

### Parámetros

*customActionsName*    Nombre de la definición de la acción personalizada que debe desinstalarse.

### Valor devuelto

Un valor booleano de `false` si no se encuentra ninguna acción personalizada llamada *customActionsName*. Si se han eliminado correctamente las acciones personalizadas, se devuelve el valor `true`.

### Descripción

Método; elimina el archivo de definición XML de acciones personalizadas *customActionsName*.

El nombre del archivo de definición debe ser un nombre simple, sin la extensión .xml y sin separadores de directorio (':', '/' o '\').

## Clase Date

### Disponibilidad

Flash Player 5.

### Descripción

La clase Date permite recuperar valores de fecha y hora relativos a la hora universal (hora de Greenwich, conocida como hora universal o UTC) o relativos al sistema operativo en el que se está ejecutando Flash Player. Los métodos de la clase Date no son estáticos, pero sólo se aplican al objeto Date individual especificado cuando se llama al método. El método `Date.UTC()` es una excepción, ya que se trata de un método estático.

La clase Date controla el horario de verano de forma distinta en función del sistema operativo y la versión de Flash Player. Flash Player 6 y las versiones posteriores manejan el horario de verano en los sistemas operativos siguientes como se indica a continuación:

- Windows: el objeto Date se ajusta automáticamente según el horario de verano. El objeto Date detecta si se emplea el horario de verano en el entorno regional actual y, si es así, detecta la fecha y la hora en que se pasa del horario estándar al horario de verano. Sin embargo, las fechas de transición que están en vigor en ese momento se aplican a fechas pasadas o futuras, de modo que puede que la diferencia horaria del horario de verano no se calcule correctamente en el caso de fechas pasadas si el entorno regional tiene fechas de transición diferentes.
- Mac OS X: el objeto Date se ajusta de forma automática para el horario de verano. La base de datos de información de zonas horarias de Mac OS X sirve para determinar si a las fechas y horas pasadas o futuras se les debe aplicar una diferencia horaria debida al horario de verano.

Flash Player 5 maneja el horario de verano en estos sistemas operativos del modo siguiente:

- Windows: la normativa de EE. UU. para el horario de verano siempre se aplica, lo que lleva a transiciones incorrectas en Europa y otras zonas que también tienen un horario de verano, pero cuyo momento de transición es distinto al de EE. UU. Flash detecta correctamente si el horario de verano se utiliza en el entorno regional actual.

Para llamar a los métodos de la clase Date, en primer lugar debe crear un objeto Date utilizando el constructor para la clase Date, que se describe más adelante en esta sección.

## Resumen de métodos para la clase Date

Método	Descripción
<code>Date.getDate()</code>	Devuelve el día del mes según la hora local.
<code>Date.getDay()</code>	Devuelve el día de la semana según la hora local.
<code>Date.getFullYear()</code>	Devuelve el año en formato de cuatro dígitos según la hora local.
<code>Date.getHours()</code>	Devuelve la hora según la hora local.
<code>Date.getMilliseconds()</code>	Devuelve los milisegundos según la hora local.
<code>Date.getMinutes()</code>	Devuelve los minutos según la hora local.
<code>Date.getMonth()</code>	Devuelve el mes según la hora local.
<code>Date.getSeconds()</code>	Devuelve los segundos según la hora local.
<code>Date.getTime()</code>	Devuelve el número de milisegundos desde la medianoche del 1 de enero de 1970, hora universal.
<code>Date.getTimezoneOffset()</code>	Devuelve la diferencia, en minutos, entre la hora local del sistema y la hora universal.
<code>Date.getUTCDate()</code>	Devuelve el día (fecha) del mes según la hora universal.
<code>Date.getUTCDay()</code>	Devuelve el día de la semana según la hora universal.
<code>Date.getUTCFullYear()</code>	Devuelve el año en formato de cuatro dígitos según la hora universal.
<code>Date.getUTCHours()</code>	Devuelve la hora según la hora universal.
<code>Date.getUTCMilliseconds()</code>	Devuelve los milisegundos según la hora universal.

Método	Descripción
<code>Date.getUTCMinutes()</code>	Devuelve los minutos según la hora universal.
<code>Date.getUTCMonth()</code>	Devuelve el mes según la hora universal.
<code>Date.getUTCSeconds()</code>	Devuelve los segundos según la hora universal.
<code>Date.getYear()</code>	Devuelve el año según la hora local.
<code>Date.setDate()</code>	Establece el día del mes según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setFullYear()</code>	Establece el año completo según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setHours()</code>	Establece la hora según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setMilliseconds()</code>	Establece los milisegundos según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setMinutes()</code>	Establece los minutos según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setMonth()</code>	Establece el mes según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setSeconds()</code>	Establece los segundos según la hora local. Devuelve la nueva hora en milisegundos.
<code>Date.setTime()</code>	Establece la fecha en milisegundos. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCDate()</code>	Establece la fecha según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCFullYear()</code>	Establece el año según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCHours()</code>	Establece la hora según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCMilliseconds()</code>	Establece los milisegundos según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCMinutes()</code>	Establece los minutos según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCMonth()</code>	Establece el mes según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setUTCSeconds()</code>	Establece los segundos según la hora universal. Devuelve la nueva hora en milisegundos.
<code>Date.setYear()</code>	Establece el año según la hora local.
<code>Date.toString()</code>	Devuelve un valor de cadena que representa la fecha y la hora almacenada en el objeto Date especificado.
<code>Date.UTC()</code>	Devuelve el número de milisegundos entre la medianoche del 1 de enero de 1970, la hora universal, y la hora especificada.

## Constructor para la clase Date

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Date()  
new Date(year, month [, date [, hour [, minute [, second [, millisecond ]]]])
```

### Parámetros

*year* Valor de 0 a 99 que indica de 1900 a 1999; de lo contrario, deben especificarse los 4 dígitos del año.

*month* Número entero de 0 (enero) a 11 (diciembre).

*date* Número entero de 1 a 31. Este parámetro es opcional.

*hour* Número entero de 0 (medianoche) a 23 (11 PM).

*minute* Número entero de 0 a 59. Este parámetro es opcional.

*second* Número entero de 0 a 59. Este parámetro es opcional.

*millisecond* Número entero de 0 a 999. Este parámetro es opcional.

### Valor devuelto

Ninguno.

### Descripción

Objeto; construye un nuevo objeto Date que contiene la fecha y la hora actual o la fecha especificada.

### Ejemplo

El ejemplo siguiente recupera la fecha y la hora actuales.

```
now_date = new Date();
```

En el ejemplo siguiente se crea un nuevo objeto Date para el día en que nació Gary: el 12 de agosto de 1974. El parámetro de mes es de base cero, por lo que para el mes se utiliza 7 en lugar de 8.

```
garyBirthdate_date = new Date (74, 7, 12);
```

En el ejemplo siguiente se crea un nuevo objeto Date, se concatenan los valores devueltos de `Date.getMonth()`, `Date.getDate()` y `Date.getFullYear()` y se visualizan en el campo de texto especificado por la variable `date_str`.

```
today_date = new Date();  
date_str = ((today_date.getMonth() + 1) + "/" + today_date.getDate() + "/" +  
    today_date.getFullYear());
```



## Date.getDate()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getDate()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el día del mes (un número entero de 1 a 31) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getDay()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getDay()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el día de la semana (0 para el domingo, 1 para el lunes, etc.) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getFullYear()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getFullYear()
```

### Parámetros

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el año completo (un número de cuatro dígitos, como por ejemplo, 2000) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

**Ejemplo**

En el ejemplo siguiente se utiliza el constructor para crear un nuevo objeto Date y enviar el valor devuelto por el método `getFullYear()` al panel Salida.

```
my_date = new Date();  
trace(my_date.getFullYear());
```

## Date.getHours()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getHours()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve la hora (un número entero de 0 a 23) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getMilliseconds()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getMilliseconds()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve los milisegundos (un número entero de 0 a 999) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getMinutes()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getMinutes()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve los minutos (un número entero de 0 a 59) del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getMonth()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getMonth()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el mes (0 para enero, 1 para febrero, etc.) del objeto Date especificado, según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getSeconds()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getSeconds()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve los segundos (un número entero de 0 a 59) del objeto Date especificado, según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.getTime()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getTime()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el número de milisegundos desde la medianoche del 1 de enero de 1970, hora universal, para el objeto Date especificado. Utilice este método para representar un instante específico en el tiempo cuando compare dos o más objetos Date.

## Date.getTimezoneOffset()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getTimezoneOffset()
```

### Parámetros

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve la diferencia, en minutos, entre la hora local del sistema y la hora universal.

**Ejemplo**

En el ejemplo siguiente se devuelve la diferencia entre el horario de verano local de San Francisco y la hora universal. El horario de verano se aplica al resultado devuelto sólo si la fecha definida en el objeto Date está comprendida en el periodo en el que se aplica el horario de verano.

```
trace(new Date().getTimezoneOffset());  
// Se visualiza 420 en el panel Salida  
// (7 horas * 60 minutos/hora = 420 minutos)  
// Este ejemplo es el horario de verano del Pacífico (PDT, GMT-0700).  
// El resultado variará según el entorno regional y la fecha del año.
```

## Date.getUTCDate()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getUTCDate()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el día del mes (un número entero de 1 a 31) en el objeto Date especificado, de acuerdo con la hora universal.

## Date.getUTCDay()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getUTCDay()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

### Descripción

Método; devuelve el día de la semana (0 para el domingo, 1 para el lunes, etc.) del objeto Date especificado según la hora universal.

## Date.getUTCFullYear()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getUTCFullYear()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el año en cuatro dígitos del objeto Date especificado según la hora universal.

## Date.getUTCHours()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getUTCHours()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve las horas del objeto Date especificado según la hora universal.

## Date.getUTCMilliseconds()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.getUTCMilliseconds()
```

### Parámetros

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve los milisegundos del objeto Date especificado según la hora universal.

## Date.getUTCMinutes()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getUTCMinutes()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve los minutos del objeto Date especificado según la hora universal.

## Date.getUTCMonth()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getUTCMonth()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el mes (0 para enero, 1 para febrero, etc.) del objeto Date especificado, según la hora universal.

## Date.getUTCSeconds()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getUTCSeconds()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve los segundos del objeto Date especificado según la hora universal.

## Date.getYear()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.getYear()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el año del objeto Date especificado según la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player. El año es el año completo menos 1900. Por ejemplo, el año 2000 se representa como 100.

**Véase también**

[Date.getFullYear\(\)](#)

## Date.setDate()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.setDate(date)
```

**Parámetros**

*date* Número entero del 1 al 31.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece el día del mes del objeto Date especificado, según la hora local, y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.



# Date.setFullYear()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_date.setFullYear(year [, month [, date]] )
```

## Parámetros

*year* Número de cuatro dígitos que especifica un año. Los números de dos dígitos no representan años; por ejemplo, 99 no es el año 1999, sino el año 99.

*month* Número entero de 0 (enero) a 11 (diciembre). Este parámetro es opcional.

*date* Número de 1 a 31. Este parámetro es opcional.

## Valor devuelto

Un número entero.

## Descripción

Método; establece el año del objeto Date especificado, según la hora local, y devuelve la nueva hora en milisegundos. Si se especifican los parámetros *month* y *date*, también se establecen en la hora local. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

Llamar a este método no modifica los otros campos del objeto Date especificado, pero los métodos `Date.getUTCDay()` y `Date.getDay()` pueden dar un nuevo valor si el día de la semana cambia como resultado de la llamada a este método.

# Date.setHours()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_date.setHours(hour)
```

## Parámetros

*hour* Número entero de 0 (medianoche) a 23 (11 PM).

## Valor devuelto

Un número entero.

## Descripción

Método; establece las horas del objeto Date especificado según la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.setMilliseconds()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setMilliseconds(milliseconds)
```

### Parámetros

*milliseconds* Número entero de 0 a 999.

### Valor devuelto

Un número entero.

### Descripción

Método; establece los milisegundos para el objeto Date especificado según la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.setMinutes()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setMinutes(minute)
```

### Parámetros

*minute* Número entero de 0 a 59.

### Valor devuelto

Un número entero.

### Descripción

Método; establece los minutos de un objeto Date especificado según la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.setMonth()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setMonth(month [, date ])
```

### Parámetros

*month* Número entero de 0 (enero) a 11 (diciembre).

*date* Número entero de 1 a 31. Este parámetro es opcional.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece el mes del objeto Date especificado en la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.setSeconds()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.setSeconds(second)
```

**Parámetros**

*second*    Número entero de 0 a 59.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece los segundos del objeto Date especificado en la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.setTime()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.setTime(milliseconds)
```

**Parámetros**

*milliseconds*    Valor entero donde 0 son las 0:00 GMT del 1 de enero de 1970.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece la fecha del objeto Date especificado en milisegundos desde la medianoche del 1 de enero de 1970 y devuelve la nueva hora en milisegundos.

## Date.setUTCDate()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCDate(date)
```

### Parámetros

*date* Número entero del 1 al 31.

### Valor devuelto

Un número entero.

### Descripción

Método; establece la fecha del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos. Llamar a este método no modifica los otros campos del objeto Date especificado, pero los métodos [Date.getUTCDay\(\)](#) y [Date.getDay\(\)](#) pueden dar un nuevo valor si el día de la semana cambia como resultado de llamar a este método.

## Date.setUTCFullYear()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCFullYear(year [, month [, date]])
```

### Parámetros

*year* Año especificado en formato de cuatro dígitos, como por ejemplo, 2000.

*month* Número entero de 0 (enero) a 11 (diciembre). Este parámetro es opcional.

*date* Número entero de 1 a 31. Este parámetro es opcional.

### Valor devuelto

Un número entero.

### Descripción

Método; establece el año del objeto Date especificado (*my\_date*) en la hora universal y devuelve la nueva hora en milisegundos.

De modo opcional, este método también puede establecer el mes y la fecha representados por el objeto Date especificado. Llamar a este método no modifica los otros campos del objeto Date especificado, pero los métodos [Date.getUTCDay\(\)](#) y [Date.getDay\(\)](#) pueden dar un nuevo valor si el día de la semana cambia como resultado de llamar a este método.

## Date.setUTCHours()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCHours(hour [, minute [, second [, millisecond]]])
```

### Parámetros

*hour* Número entero de 0 (medianoche) a 23 (11 PM).

*minute* Número entero de 0 a 59. Este parámetro es opcional.

*second* Número entero de 0 a 59. Este parámetro es opcional.

*millisecond* Número entero de 0 a 999. Este parámetro es opcional.

### Valor devuelto

Un número entero.

### Descripción

Método; establece la hora del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos.

## Date.setUTCMilliseconds()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCMilliseconds(millisecond)
```

### Parámetros

*millisecond* Número entero de 0 a 999.

### Valor devuelto

Un número entero.

### Descripción

Método; establece los milisegundos del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos.

## Date.setUTCMinutes()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCMinutes(minute [, second [, millisecond]])
```

### Parámetros

*minute* Número entero de 0 a 59.

*second* Número entero de 0 a 59. Este parámetro es opcional.

*millisecond* Número entero de 0 a 999. Este parámetro es opcional.

### Valor devuelto

Un número entero.

### Descripción

Método; establece el minuto del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos.

## Date.setUTCMonth()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCMonth(month [, date])
```

### Parámetros

*month* Número entero de 0 (enero) a 11 (diciembre).

*date* Número entero de 1 a 31. Este parámetro es opcional.

### Valor devuelto

Un número entero.

### Descripción

Método; establece el mes y, opcionalmente, el día (la *date*) del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos. Llamar a este método no modifica otros campos del objeto Date especificado, pero los métodos [Date.getUTCDay\(\)](#) y [Date.getDay\(\)](#) pueden notificar un nuevo valor si el día de la semana cambia como consecuencia de especificar un valor para el parámetro *date*.

## Date.setUTCSeconds()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_date.setUTCSeconds(second [, millisecond])
```

### Parámetros

*second* Número entero de 0 a 59.

*millisecond* Número entero de 0 a 999. Este parámetro es opcional.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece los segundos del objeto Date especificado en la hora universal y devuelve la nueva hora en milisegundos.

## Date.setYear()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.setYear(year)
```

**Parámetros**

*year* Si *year* es un número entero entre 0 y 99, `setYear` establece el año en 1900 + *year*; de lo contrario, el año es el valor del parámetro *year*.

**Valor devuelto**

Un número entero.

**Descripción**

Método; establece el año del objeto Date especificado en la hora local y devuelve la nueva hora en milisegundos. La hora local la determina el sistema operativo en el que se esté ejecutando Flash Player.

## Date.toString()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_date.toString()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Una cadena.

**Descripción**

Método; devuelve un valor de cadena del objeto Date especificado en un formato legible y devuelve la nueva hora en milisegundos.

## Ejemplo

En el ejemplo siguiente se devuelve la información del objeto Date denominado `dateOfBirth_date` como una cadena.

```
var dateOfBirth_date = new Date(74, 7, 12, 18, 15);  
trace (dateOfBirth_date.toString());
```

Salida (para la hora estándar del Pacífico):

```
Mon Aug 12 18:15:00 GMT-0700 1974
```

## Date.UTC()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Date.UTC(year, month [, date [, hour [, minute [, second [, millisecond ]]]]])
```

### Parámetros

*year* Número de cuatro dígitos, como por ejemplo, 2000.

*month* Número entero de 0 (enero) a 11 (diciembre).

*date* Número entero de 1 a 31. Este parámetro es opcional.

*hour* Número entero de 0 (medianoche) a 23 (11 PM).

*minute* Número entero de 0 a 59. Este parámetro es opcional.

*second* Número entero de 0 a 59. Este parámetro es opcional.

*millisecond* Número entero de 0 a 999. Este parámetro es opcional.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el número de milisegundos desde la medianoche del 1 de enero de 1970, hora universal, y la hora especificada en los parámetros. Este es un método estático que se invoca por medio del constructor del objeto Date, no por medio de un objeto Date específico. Este método permite crear un objeto Date al que se asigna la hora universal, mientras que al constructor Date se asigna la hora local.

## Ejemplo

En el ejemplo siguiente se crea un nuevo objeto Date denominado `garyBirthday_date` definido con la hora universal. Esta es la variación de la hora universal del ejemplo utilizado para el método constructor `newDate`.

```
garyBirthday_date = new Date(Date.UTC(1974, 7, 12));
```



## default

### Disponibilidad

Flash Player 6.

### Sintaxis

`default: statements`

### Parámetros

*statements*    Cualquier número de sentencias.

### Valor devuelto

Ninguno.

### Descripción

Sentencia; define el bloque case predeterminado para una acción `switch`. Las sentencias se ejecutan si el parámetro *expression* de la acción `switch` no equivale (utilizando la igualdad estricta) a ninguno de los parámetros *expression* que siguen a las palabras clave `case` de una acción `switch` determinada.

No es necesario que la acción `switch` tenga un bloque `default`. No es necesario que el bloque `default` sea el último de la lista. La utilización de una acción `default` fuera de una acción `switch` es un error que hace que el script no se compile.

### Ejemplo

En el ejemplo siguiente, la expresión A no equivale a las expresiones B o D, por lo que se ejecuta la sentencia que sigue a la palabra clave `default` y la acción `trace()` se envía al panel Salida.

```
switch ( A ) {  
    case B:  
        C;  
        break;  
    case D:  
        E;  
        break;  
    default:  
        trace ("no se ha detectado ningún caso específico");  
}
```

### Véase también

[switch](#), [case](#), [break](#)

## delete

### Disponibilidad

Flash Player 5.

### Sintaxis

`delete reference`

### Parámetros

*reference*    Nombre de la variable u objeto que se va a eliminar.

## Valor devuelto

Valor booleano.

## Descripción

Operador; destruye el objeto o la variable especificada por el parámetro *reference* y devuelve *true* si el objeto se ha eliminado correctamente; de lo contrario, devuelve el valor *false*. Este operador es útil para liberar memoria utilizada por scripts. Aunque *delete* es un operador, normalmente se utiliza como sentencia, como en el ejemplo siguiente:

```
delete x;
```

El operador *delete* puede fallar y devolver *false* si el parámetro *reference* no existe o no puede eliminarse. Los objetos y propiedades predefinidos y las variables declaradas con *var* no pueden eliminarse. No puede utilizar el operador *delete* para eliminar clips de película.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se crea un objeto, se utiliza y después se elimina cuando ya no es necesario.

```
account = new Object();
account.name = 'Jon';
account.balance = 10000;
```

```
delete account;
```

Sintaxis 2: el ejemplo siguiente elimina una propiedad de un objeto.

```
// crear el nuevo objeto "account"
account = new Object();
// asignar un nombre de propiedad a la cuenta
account.name = 'Jon';
// eliminar la propiedad
delete account.name;
```

Sintaxis 3: el ejemplo siguiente es otro ejemplo de la eliminación de una propiedad de un objeto.

```
// crear un elemento Array de longitud 0
my_array = new Array();
// añadir un elemento a la matriz. Array.length es ahora 1.
my_array[0] = "abc";
// añadir otro elemento a la matriz. Array.length es ahora 2.
my_array[1] = "def";
// añadir otro elemento a la matriz. Array.length es ahora 3.
my_array[2] = "ghi";
// se elimina my_array[2], pero Array.length no cambia
delete array[2];
trace(my_array.length);
```

Sintaxis 4: en el ejemplo siguiente se muestra el comportamiento de *delete* en referencias de objeto.

```
// crear un nuevo objeto y asignar la variable ref1
// para hacer referencia al objeto
ref1 = new Object();
ref1.name = "Jody";
// copiar la variable de referencia a la nueva variable
// y eliminar ref1
ref2 = ref1;
delete ref1;
```

Si `ref1` no se hubiera copiado en `ref2`, el objeto se habría eliminado al eliminar `ref1`, porque no habría referencias a él. Si elimina `ref2`, ya no habrá referencias al objeto; se destruirá y la memoria que utilizaba pasará a estar disponible.

**Véase también**

`var`

## do while

**Disponibilidad**

Flash Player 4.

**Sintaxis**

```
do {  
    statement(s)  
} while (condition)
```

**Parámetros**

*condition* Condición que se comprueba.

*statement(s)* Sentencia(s) que debe(n) ejecutarse mientras el parámetro *condition* tenga el valor `true`.

**Valor devuelto**

Ninguno.

**Descripción**

Sentencia; ejecuta las sentencias y después comprueba la condición de una reproducción indefinida mientras la condición sea `true`.

**Véase también**

`break`, `continue`

## duplicateMovieClip()

**Disponibilidad**

Flash Player 4.

**Sintaxis**

```
duplicateMovieClip(target, newname, depth)
```

**Parámetros**

*target* Ruta de destino del clip de película que debe duplicarse.

*newname* Identificador exclusivo para el clip de película duplicado.

*depth* Nivel de profundidad exclusivo del clip de película duplicado. El nivel de profundidad es un orden de apilamiento de los clips de película duplicados. Este orden de apilamiento se parece mucho al orden de apilamiento de las capas en la línea de tiempo; los clips de película con un nivel de profundidad menor permanecen ocultos debajo de los clips con un orden de apilamiento mayor. Debe asignar a cada clip de película duplicado un nivel de profundidad exclusivo para evitar que reemplace archivos SWF existentes en las profundidades ocupadas.

### Valor devuelto

Una referencia al clip de película duplicado.

### Descripción

Función; crea una instancia de un clip de película mientras se reproduce el archivo SWF. La cabeza lectora de los clips de película duplicados siempre empieza en el fotograma 1, sin tener en cuenta la posición de la cabeza lectora en el clip de película original (o principal). Las variables del clip de película principal no se copian en el clip de película duplicado. Si se elimina el clip de película principal también se elimina el clip de película duplicado. Utilice la acción o el método `removeMovieClip()` para eliminar una instancia de clip de película creada con `duplicateMovieClip()`.

### Véase también

`MovieClip.duplicateMovieClip()`, `removeMovieClip()`, `MovieClip.removeMovieClip()`

## dynamic

### Disponibilidad

Flash Player 6.

### Sintaxis

```
dynamic class className [ extends superClass ]  
                        [ implements interfaceName [, interfaceName... ] ]  
{  
    // la definición de clase  
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

### Descripción

Palabra clave; especifica que los objetos basados en la clase especificada pueden añadir y acceder a propiedades dinámicas en tiempo de ejecución.

La verificación de tipos en estas clases no es tan estricta como en las clases no dinámicas, ya que los miembros a los que se ha accedido en la definición de clase y en las instancias de clase no se comparan con los definidos en el alcance de clase. No obstante, es posible que el tipo de las funciones de miembro de clase se verifique para el tipo de devolución y los tipos de parámetro. Este comportamiento es especialmente útil cuando se trabaja con objetos `MovieClip`, en los que hay varias formas de añadir propiedades y objetos a un clip de película de forma dinámica, como `MovieClip.createEmptyMovieClip()` y `MovieClip.createTextField()`.

Subclasses of dynamic classes are also dynamic.

Para más información, consulte [“Creación de clases dinámicas” en la página 180](#).

## Ejemplo

En el ejemplo siguiente, se ha marcado la clase B como dinámica, por lo que si se llama a alguna función no declarada en dicha clase no se generará ningún error durante la compilación.

```
// en B.as
dynamic class B extends class_A {
  function B() {
    /*this is the constructor*/
  }
  function m():Number {return 25;}
  function o(s:String):Void {trace(s);}
}

// en C.as
class C extends class_A {
  function C() {
    /*this is the constructor*/
  }
  function m():Number {return 25;}
  function o(s:String):Void {trace(s);}
}

// en otro script
var var1 = B.n();    // correcto
var var2 = C.n()     // incorrecto, ya que no existe ninguna función n en C.as
```

## Véase también

[class](#), [extends](#)

# else

## Disponibilidad

Flash Player 4.

## Sintaxis

```
if (condition){
  statement(s);
} else (condition){
  statement(s);
}
```

## Parámetros

*condition*    Expresión que da como resultado true o false.

*statement(s)*    Serie alternativa de sentencias que debe ejecutarse si la condición especificada en la sentencia if es false.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; especifica las sentencias que deben ejecutarse si la condición de la sentencia if devuelve el valor false.

## Véase también

[if](#)

# else if

## Disponibilidad

Flash Player 4.

## Sintaxis

```
if (condition){  
    statement(s);  
} else if (condition){  
    statement(s);  
}
```

## Parámetros

*condition* Expresión que da como resultado `true` o `false`.

*statement(s)* Serie alternativa de sentencias que debe ejecutarse si la condición especificada en la sentencia `if` es `false`.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; calcula el valor de una condición y especifica las sentencias que deben ejecutarse si la condición de la sentencia `if` inicial es `false`. Si la condición `else if` es `true`, el intérprete de Flash ejecuta las sentencias que van después de la condición entre llaves (`{}`). Si la condición `else if` es `false`, Flash pasa por alto las sentencias entre llaves y ejecuta las sentencias que van después de las llaves. Utilice la acción `else if` para definir lógica de ramificación en los scripts.

## Ejemplo

En el ejemplo siguiente se utilizan acciones `else if` para comprobar si los lados de un objeto se encuentran dentro de un límite específico:

```
// si el objeto sobrepasa los límites,  
// devolverlo e invertir su velocidad de desplazamiento  
if (this._x>rightBound) {  
    this._x = rightBound;  
    xInc = -xInc;  
} else if (this._x<leftBound) {  
    this._x = leftBound;  
    xInc = -xInc;  
} else if (this._y>bottomBound) {  
    this._y = bottomBound;  
    yInc = -yInc;  
} else if (this._y<topBound) {  
    this._y = topBound;  
    yInc = -yInc;  
}
```

## Véase también

[if](#)

## #endinitclip

### Disponibilidad

Flash Player 6.

### Sintaxis

```
#endinitclip
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Directiva del compilador; indica el final de un bloque de acciones de inicialización.

### Ejemplo

```
#initclip  
...aquí se indican las acciones de inicialización...  
#endinitclip
```

### Véase también

[#initclip](#)

## eq (igual; específico para cadenas)

### Disponibilidad

Flash Player 4. Este operador se eliminó en Flash 5 y se sustituyó por el operador `==` ([igualdad](#)).

### Sintaxis

```
expression1 eq expression2
```

### Parámetros

*expression1*, *expression2*    Números, cadenas o variables.

### Valor devuelto

Ninguno.

### Descripción

Operador de comparación; compara si dos expresiones son iguales y devuelve el valor `true` si la representación de la cadena de *expression1* es igual a la representación de la cadena de *expression2*; de lo contrario, la operación devuelve el valor `false`.

### Véase también

[==](#) ([igualdad](#))

# Clase Error

## Disponibilidad

Flash Player 7.

## Descripción

Contiene información sobre el error que se ha producido en un script. Para crear un objeto Error, se utiliza la función constructora `Error`. Generalmente, se “emite” un nuevo objeto Error desde un bloque de código `try` que, a continuación, es “capturado” por un bloque de código `catch` o `finally`.

También puede crear una subclase de la clase Error y emitir instancias de dicha subclase.

## Resumen de métodos para la clase Error

Método	Descripción
<code>Error.toString()</code>	Devuelve la representación de cadena de un objeto Error.

## Resumen de propiedades para la clase Error

Propiedad	Descripción
<code>Error.message</code>	Cadena que contiene un mensaje de error asociado a un error.
<code>Error.name</code>	Cadena que contiene el nombre del objeto Error.

## Constructor para la clase Error

### Disponibilidad

Flash Player 7.

### Sintaxis

```
new Error([message])
```

### Parámetros

*message* Cadena asociada al objeto Error; este parámetro es opcional.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto Error. Si se especifica un *message*, su valor se asigna a la propiedad `Error.message` del objeto.

### Ejemplo

En el ejemplo siguiente, una función emite un error (con el mensaje especificado) si se pasan dos cadenas que no son idénticas.

```
function compareStrings(string_1, string_2) {  
    if(string_1 != string_2) {  
        throw new Error("Las cadenas no coinciden.");  
    }  
}
```



```
    }  
  }  
  try {  
    compareStrings("Dog","dog");  
  } catch (e) {  
    trace(e.toString());  
  }  
}
```

#### Véase también

[throw](#), [try..catch..finally](#)

## Error.message

#### Disponibilidad

Flash Player 7.

#### Sintaxis

*myError*.message

#### Descripción

Propiedad; contiene el mensaje asociado con el objeto Error. De forma predeterminada, el valor de esta propiedad es "Error". Cuando cree un nuevo objeto Error, puede especificar una propiedad `message` pasando la cadena del error a la función constructora `Error`.

#### Véase también

[throw](#), [try..catch..finally](#)

## Error.name

#### Disponibilidad

Flash Player 7.

#### Sintaxis

*myError*.name

#### Descripción

Propiedad; contiene el nombre del objeto Error. De forma predeterminada, el valor de esta propiedad es "Error".

#### Véase también

[throw](#), [try..catch..finally](#)

## Error.toString()

#### Disponibilidad

Flash Player 7.

#### Sintaxis

*my\_err*.toString()

**Valor devuelto**

Una cadena.

**Descripción**

Método; devuelve la cadena "Error" de forma predeterminada, o el valor contenido en `Error.message`, si se ha definido.

**Véase también**

`Error.message`, `throw`, `try..catch..finally`

## escape

**Disponibilidad**

Flash Player 5.

**Sintaxis**

`escape(expression)`

**Parámetros**

*expresión* Expresión que se va a convertir en una cadena y que se va a codificar con un formato URL.

**Valor devuelto**

Ninguno.

**Descripción**

Función; convierte el parámetro en una cadena y lo codifica en formato URL, en el que todos los caracteres no alfanuméricos se convierten en secuencias hexadecimales % de escape.

**Ejemplo**

La ejecución del código siguiente da como resultado `Ho1a%7B%5BMundo%5D%7D`.

```
escape("Ho1a{[Mundo]}");
```

**Véase también**

`unescape`

## eval()

**Disponibilidad**

Flash Player 5 o posterior para disponer de todas las funciones. Puede utilizar la función `eval()` cuando realice una exportación a Flash Player 4, pero debe utilizar la notación usando barras inclinadas y sólo puede acceder a variables y no a propiedades ni objetos.

**Sintaxis**

`eval(expression)`

## Parámetros

*expression* Cadena que contiene el nombre de una variable, propiedad, objeto o clip de película que debe recuperarse.

## Valor devuelto

Un valor, referencia a un objeto o clip de película o `undefined`.

## Descripción

Función; accede a variables, propiedades, objetos o clips de película por nombre. Si *expression* es una variable o una propiedad, se devuelve el valor de la variable o de la propiedad. Si *expression* es un objeto o un clip de película, se devuelve una referencia al objeto o al clip de película. Si el elemento nombrado en *expression* no puede encontrarse, se devuelve `undefined`.

En Flash 4, se utilizaba `eval()` para simular matrices; en Flash 5 o en versiones posteriores, es aconsejable utilizar para ello la clase `Array`.

En Flash 4, también puede utilizar `eval()` para definir y recuperar dinámicamente el valor de una variable o de un nombre de instancia. Sin embargo, también puede llevar esto a cabo con el operador de acceso de matriz (`[]`).

En Flash 5 y versiones posteriores, no puede utilizar `eval()` para definir dinámicamente y recuperar el valor de una variable o de un nombre de instancia, ya que no puede utilizar `eval()` en la parte izquierda de una ecuación. Por ejemplo, sustituya el código:

```
eval ("var" + i) = "first";
```

por el código siguiente:

```
this["var"+i] = "first"
```

o por éste:

```
set ("var" + i, "first");
```

## Ejemplo

En el ejemplo siguiente se utiliza `eval()` para determinar el valor de la expresión `"piece" + x`. Puesto que el resultado es un nombre de variable, `piece3`, la función `eval()` devuelve el valor de la variable y lo asigna a `y`:

```
piece3 = "peligroso";  
x = 3;  
  
y = eval("piece" + x);  
trace(y);  
  
// Resultado: peligroso
```

## Véase también

[Clase Array](#)

## extends

### Disponibilidad

Flash Player 6.

## Sintaxis

```
class className extends otherClassName {}  
interface interfaceName extends otherInterfaceName {}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Parámetros

*className* El nombre de la clase que se está definiendo.  
*otherClassName* El nombre de la clase en que se basa *className*.  
*interfaceName* El nombre de la interfaz que se está definiendo.  
*otherInterfaceName* El nombre de la interfaz en la que se basa *interfaceName*.

## Descripción

Palabra clave; define una clase o interfaz que es una subclase de otra clase o interfaz; esta última es la superclase. La subclase hereda todos los métodos, las propiedades, las funciones, etc. que se han definido en la superclase.

Para más información, consulte [“Creación de subclases” en la página 169](#).

## Ejemplo

En la clase B que se define a continuación, se insertará automáticamente y como primera sentencia de la función constructora B una llamada al constructor de la clase A, ya que todavía no existe ninguna llamada (se comenta en el ejemplo.)

```
class B extends class A  
{  
    function B() { // éste es el constructor  
        // super(); // opcional; se inserta durante la compilación si se ha omitido  
    }  
    function m():Number {return 25;}  
    function o(s:String):Void {trace(s);}  
}
```

## Véase también

[class](#), [implements](#), [interface](#)

# false

## Disponibilidad

Flash Player 5.

## Sintaxis

```
false
```

## Descripción

Constante; valor booleano exclusivo que representa lo contrario de `true`.

## Véase también

[true](#)

## **\_focusrect**

### **Disponibilidad**

Flash Player 4.

### **Sintaxis**

```
_focusrect = booleano;
```

### **Descripción**

Propiedad (global); especifica si aparece un rectángulo amarillo alrededor de un botón o de un clip de película que se selecciona mediante el teclado. El valor predeterminado, `true`, muestra un rectángulo amarillo alrededor del botón o campo de texto seleccionado a medida que el usuario presiona la tecla de tabulación para desplazarse por los objetos de un archivo SWF. Especifique `false` si no desea visualizar el rectángulo amarillo. Ésta es una propiedad global que puede sustituirse para instancias específicas.

### **Véase también**

`Button._focusrect`, `MovieClip._focusrect`

## **for**

### **Disponibilidad**

Flash Player 5.

### **Sintaxis**

```
for(init; condition; next) {  
    statement(s);  
}
```

### **Parámetros**

*init* Expresión que debe comprobarse antes de que comience la secuencia de reproducción indefinida. Suele tratarse de una expresión de asignación. También se permite una sentencia `var` para este parámetro.

*condition* Expresión que da como resultado `true` o `false`. La condición se comprueba antes de cada repetición de reproducción indefinida; la reproducción finaliza cuando la condición da como resultado `false`.

*next* Expresión que debe comprobarse después de cada repetición de reproducción indefinida; normalmente se trata de una expresión de asignación que utiliza los operadores `++` (incremento) o `--` (decremento).

*statement(s)* Instrucción o instrucciones que deben ejecutarse en el cuerpo de la reproducción indefinida.

### **Descripción**

Sentencia; construcción de reproducción indefinida que comprueba la expresión *init* (inicializar) una vez y después comienza una secuencia de reproducción indefinida por medio de la cual, siempre que la *condition* dé como resultado `true`, se ejecuta la *statement* y se comprueba la siguiente expresión.

Las acciones `for` o `for..in` no pueden enumerar algunas propiedades. Por ejemplo, los métodos incorporados de la clase `Array` (como `Array.sort()` y `Array.reverse()`) no se incluyen en la enumeración de un objeto `Array`, y las propiedades de clip de película, como `_x` y la propiedad `_y`, no se enumeran. En archivos de clase externos, los miembros de instancia no son numerables; únicamente lo son los miembros dinámicos y estáticos.

## Ejemplo

En el ejemplo siguiente se utiliza `for` para agregar los elementos en una matriz:

```
my_array=new Array();
for(i=0; i<10; i++) {
    my_array [i] = (i + 5)*10;
    trace(my_array[i]);
}
```

Los resultados siguientes se visualizan en el panel Salida:

```
50
60
70
80
90
100
110
120
130
140
```

A continuación, se muestra un ejemplo de la utilización de `for` para realizar la misma acción repetidamente. En el código siguiente, la reproducción indefinida `for` añade los números de 1 a 100:

```
var sum = 0;
for (var i=1; i<=100; i++) {
    sum = sum + i;
}
```

## Véase también

`++ (incremento)`, `-- (decremento)`, `for..in`, `var`

## for..in

### Disponibilidad

Flash Player 5.

### Sintaxis

```
for(variableIterant in object){
    statement(s);
}
```

### Parámetros

*variableIterant* Nombre de una variable que actúa como repetidor, haciendo referencia a cada propiedad de un objeto o elemento de una matriz.

*object* Nombre de un objeto que debe repetirse.

*statement(s)* Instrucción que debe ejecutarse para cada repetición.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; realiza una reproducción indefinida por las propiedades de un objeto o elemento de una matriz y ejecuta la *statement* para cada propiedad de un objeto.

Las acciones `for` o `for..in` no pueden enumerar algunas propiedades. Por ejemplo, los métodos incorporados de la clase `Array` (como `Array.sort()` y `Array.reverse()`) no se incluyen en la enumeración de un objeto `Array`, y las propiedades de clip de película, como `_x` y la propiedad `_y`, no se enumeran. En archivos de clase externos, los miembros de instancia no son numerables; únicamente lo son los miembros dinámicos y estáticos.

La sentencia `for..in` se repite en las propiedades de los objetos de la cadena prototipo del objeto repetido. Si el prototipo del objeto secundario es `principal`, repetir las propiedades del secundario con `for..in`, también repetirá las propiedades heredadas del `principal`.

La acción `for..in` enumera todos los objetos de la cadena prototipo de un objeto. Primero se enumeran las propiedades del objeto, después las propiedades del prototipo inmediato, las propiedades del prototipo del prototipo y así sucesivamente. La acción `for..in` no enumera dos veces el mismo nombre de propiedad. Si el objeto secundario tiene un prototipo principal y ambos contienen la propiedad `prop`, la acción `for..in` llamada en el secundario enumerará `prop` en el secundario, pero pasará por alto la de principal.

## Ejemplo

A continuación se muestra un ejemplo de la utilización de `for..in` para repetir las propiedades de un objeto:

```
myObject = { name:'Tara', age:27, city:'San Francisco' };
for (name in myObject) {
    trace ("myObject." + name + " = " + myObject[name]);
}
```

La salida de este ejemplo es la que se muestra a continuación:

```
myObject.name = Tara
myObject.age = 27
myObject.city = San Francisco
```

A continuación se muestra un ejemplo de la utilización del operador `typeof` con `for..in` para repetir un tipo concreto de secundario:

```
for (name in my_mc) {
    if (typeof (my_mc[name]) = "clip de película") {
        trace ("Tengo un clip de película secundario llamado " + name);
    }
}
```

El ejemplo siguiente enumera los elementos secundarios de un clip de película y envía cada uno al fotograma 2 de sus respectivas líneas de tiempo. El clip de película `RadioButtonGroup` es principal con varios secundarios, `_RedRadioButton_`, `_GreenRadioButton_` y `_BlueRadioButton_`.

```
for (var name in RadioButtonGroup) {
    RadioButtonGroup[name].gotoAndStop(2);
}
```

# fscommand()

## Disponibilidad

Flash Player 3.

## Sintaxis

```
fscommand("command", "parameters")
```

## Parámetros

*command* Cadena pasada a la aplicación host para cualquier uso o comando pasado a Flash Player.

*parameters* Cadena pasada a la aplicación host para cualquier uso o valor pasado a Flash Player.

## Valor devuelto

Ninguno.

## Descripción

Función; permite que el archivo SWF se comuniquen con Flash Player o con el programa que alberga Flash Player, como un navegador Web. También puede utilizar la acción `fscommand` para pasar mensajes a Macromedia Director o a Visual Basic, Visual C++ y otros programas que pueden aceptar controles de ActiveX.

Sintaxis 1: para enviar un mensaje a Flash Player, debe utilizar comandos y parámetros predefinidos. La tabla siguiente muestra los valores que puede especificar para los parámetros *command* y *parameters* de la acción `fscommand` a fin de controlar un archivo SWF que se ejecuta en el reproductor Flash Player (incluidos los proyectores):

Comando	Parámetros	Propósito
<code>quit</code>	Ninguno	Cierra el proyector.
<code>fullscreen</code>	<code>true</code> o <code>false</code>	Si se especifica <code>true</code> , Flash Player se establece en el modo de pantalla completa. Si se especifica <code>false</code> , el reproductor vuelve a la vista de menú normal.
<code>allowscale</code>	<code>true</code> o <code>false</code>	Si se especifica <code>false</code> , el reproductor se establece para que el archivo SWF se dibuje siempre a su tamaño original y nunca se cambie su escala. Si se especifica <code>true</code> , se obliga al archivo SWF a cambiar su escala al 100% del reproductor.
<code>showmenu</code>	<code>true</code> o <code>false</code>	Si se especifica <code>true</code> , se activa el conjunto completo de elementos de menú contextual. Si se especifica <code>false</code> , se atenúan todos los elementos de menú contextual excepto Acerca de Flash Player.
<code>exec</code>	Ruta de acceso a la aplicación	Ejecuta una aplicación desde el proyector.
<code>trapallkeys</code>	<code>true</code> o <code>false</code>	Si se especifica <code>true</code> envía todos los eventos de teclas, incluidos las teclas aceleradoras, al controlador <code>onClipEvent(keyDown/keyUp)</code> de Flash Player.



El comando `exec` sólo puede contener los caracteres A–Z, a–z, 0–9, punto (.) y subrayado (\_). El comando `exec` se ejecuta solamente en el subdirectorio `fscommand`. Es decir, si utiliza el comando `fscommand exec` para llamar a una aplicación, la aplicación debe residir en un subdirectorio denominado `fscommand`.

Sintaxis 2: para utilizar la acción `fscommand` y enviar un mensaje en un lenguaje de creación de scripts, como por ejemplo JavaScript en un navegador Web, puede pasar dos parámetros que desee en los parámetros *command* y *parameters*. Estos argumentos pueden ser cadenas o expresiones y se utilizan en una función de JavaScript que “captura” o controla la acción `fscommand`.

En un navegador Web, la acción `fscommand` llama a la función JavaScript `movienome_DoFSCommand` en la página HTML que contiene el archivo SWF. En `movienome` se incluye el nombre de Flash Player asignado mediante el atributo `NAME` de la etiqueta `EMBED` o la propiedad `ID` de la etiqueta `OBJECT`. Si asigna a Flash Player el nombre `myDocument`, la función de JavaScript se denomina `myDocument_DoFSCommand`.

Sintaxis 3: la acción `fscommand` puede enviar mensajes a Macromedia Director que son interpretados por Lingo como cadenas, eventos o código Lingo ejecutable. Si el mensaje es una cadena o un evento, debe escribir el código Lingo para recibir el mensaje de la acción `fscommand` y llevar a cabo una acción en Director. Para obtener más información, consulte el Centro de Soporte de Director en la dirección [www.macromedia.com/support/director](http://www.macromedia.com/support/director).

Sintaxis 4: en Visual Basic, Visual C++, y otros programas para los controles ActiveX, `fscommand` envía un evento VB con dos cadenas que pueden gestionarse en el lenguaje de programación del entorno empleado. Para obtener más información, utilice las palabras clave *Flash method* para buscar el Centro de Soporte Flash en [http://www.macromedia.com/go/flash\\_support\\_sp](http://www.macromedia.com/go/flash_support_sp).

## Ejemplo

Sintaxis 1: en el ejemplo siguiente, la acción `fscommand` hace que Flash Player ajuste la escala del archivo SWF a pantalla completa al soltar el botón.

```
on(release) {  
    fscommand("fullscreen", true);  
}
```

Sintaxis 2: en el ejemplo siguiente se utiliza la acción `fscommand` aplicada a un botón de Flash para abrir un cuadro de mensaje JavaScript en una página HTML. El mensaje se envía a JavaScript como parámetro `fscommand`.

Debe agregar una función a la página HTML que contiene el archivo SWF. Esta función, `myDocument_DoFSCommand`, se establece en la página HTML y espera una acción `fscommand` en Flash. Cuando se activa `fscommand` en Flash (por ejemplo, cuando un usuario presiona el botón), se pasan las cadenas *command* y *parameter* a la función `myDocument_DoFSCommand`. Puede utilizar las cadenas pasadas en el código JavaScript o VBScript como desee. En este ejemplo, la función contiene una sentencia condicional `if` que comprueba si la cadena de comando es “messagebox”. Si lo es, se abrirá un cuadro de aviso de JavaScript (o “message box”) y se mostrará el contenido de la cadena de *parameter*.

```
function myDocument_DoFSCommand(command, args) {  
    if (command == "messagebox") {  
        alert(args);  
    }  
}
```

En el documento Flash, agregue la acción `fscommand` a un botón:

```
fscommand("messagebox", "Es un cuadro de mensajes invocado desde Flash.")
```

También puede utilizar expresiones para la acción `fscommand` y sus parámetros, como en el ejemplo siguiente:

```
fscommand("cuadro de mensajes", "Hola, " + name + ", bienvenido/a a nuestra  
página Web.")
```

Para probar la película, elija Archivo > Vista previa de publicación > HTML.

**Nota:** si publica su archivo SWF utilizando Flash con la plantilla `FSCommand` en Configuración de publicación en HTML, la función `myDocument_DoFSCommand` se inserta automáticamente. Los atributos `NAME` e `ID` del archivo SWF serán el nombre del archivo. Por ejemplo, para el archivo `myDocument.fla`, los atributos se establecerían en `myDocument`.

## function

### Disponibilidad

Flash Player 5.

### Sintaxis

```
function functionname ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}  
function ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}
```

### Parámetros

*functionname*    Nombre de la nueva función.

*parameter*    Identificador que representa un parámetro que debe pasarse a la función. Estos parámetros son opcionales.

*statement(s)*    Instrucción de `ActionScript` definida para el cuerpo de `function`.

### Valor devuelto

Ninguno.

### Descripción

Sentencia; conjunto de sentencias que define para realizar una determinada tarea. Puede *declarar*, o definir, una función en una única ubicación y llamarla, o invocarla, desde diferentes scripts de un archivo SWF. Cuando defina una función, también puede especificar sus parámetros. Los parámetros son marcadores de posición para valores en los que opera la función. Puede pasar diferentes parámetros a una función cada vez que la llama. Esto permite volver a utilizar una función en muchas situaciones diferentes.

Utilice la acción `return` en las *sentencias* de una función para hacer que la función devuelva o genere un valor.

Sintaxis 1: declara una `function` con el *functionname*, los *parameters* y las *statement(s)* especificados. Cuando se llama a una función, se invoca la declaración de función. Se permite la referencia hacia delante; dentro de la misma lista de acciones, puede declararse una función después de que haya sido llamada. Una declaración de función sustituye a cualquier declaración anterior de la misma función. Puede utilizar esta sintaxis siempre que esté permitida una declaración.

Sintaxis 2: crea una función anónima y la devuelve. Esta sintaxis se utiliza en expresiones y es muy útil para instalar métodos en objetos.

### Ejemplo

Sintaxis 1: en el ejemplo siguiente se define la función `sqr`, que acepta un parámetro y devuelve el valor `square(x*x)` del parámetro. Si la función se declara y utiliza en el mismo script, la declaración de función puede aparecer tras la utilización de la función.

```
y=sqr(3);  
  
function sqr(x) {  
    return x*x;  
}
```

Sintaxis 2: en la siguiente función se define un objeto `Circle`:

```
function Circle(radius) {  
    this.radius = radius;  
}
```

En la sentencia siguiente se define una función anónima que calcula el área de un círculo y la asocia al objeto `Circle` como un método:

```
Circle.prototype.area = function () {return Math.PI * this.radius *  
    this.radius}
```

## Clase Function

### Disponibilidad

Flash Player 6.

### Resumen de métodos para la clase Function

Método	Descripción
<code>Function.apply()</code>	Activa el código de ActionScript para llamar a una función.
<code>Function.call()</code>	Invoca la función representada por un objeto Function.

### Resumen de propiedades para la clase Function

Propiedad	Descripción
<code>Function.prototype</code>	Hace referencia a un objeto que es el prototipo de una clase.

## Function.apply()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myFunction.apply(thisObject, argumentsObject)
```

## Parámetros

*thisObject* Objeto al que se aplica *myFunction*.

*argumentsObject* Matriz de elementos que se pasan a *myFunction* como parámetros.

## Valor devuelto

El valor que especifique la función llamada.

## Descripción

Método; especifica el valor de *this* que debe utilizarse con la función a la que llama *ActionScript*. Este método también especifica parámetros que deben pasarse a la función llamada. Puesto que *apply()* es un método de la clase *Function*, también es un método para cada objeto de función de *ActionScript*.

Los parámetros se especifican como un objeto *Array*. Suele ser útil cuando no se conoce el número de parámetros que se van a pasar hasta que se ejecuta el script.

## Ejemplo

Las llamadas de función siguientes son equivalentes:

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

Puede crear un archivo SWF que contenga campos de entrada que permitan al usuario especificar el nombre de una función a la que se debe llamar, y cero o más parámetros que se pasarán a la función. Si se presiona el botón “Llamar” se utilizará el método *apply* para llamar a la función especificando los parámetros.

En este ejemplo, el usuario especifica un nombre de función en un campo de introducción de texto denominado *functionName*. El número de parámetros se especifica en un campo de introducción de texto denominado *numParameters*. Se especifican hasta 10 parámetros en campos de texto denominados *parameter1*, *parameter2*, hasta *parameter10*.

```
on(release) {
    callTheFunction();
}
...
function callTheFunction()
{
    var theFunction = eval(functionName.text);
    var n = Number(numParameters);
    var parameters = [];
    for (var i = 0; i < n; i++) {
        parameters.push(eval("parameter" + i));
    }
    theFunction.apply(null, parameters);
}
```

# Function.call()

## Disponibilidad

Flash Player 6.

## Sintaxis

*myFunction.call(thisObject, parameter1, ..., parameterN)*

## Parámetros

*thisObject* Especifica el valor de *this* en el cuerpo de la función.

*parameter1* Parámetro que se pasa a *myFunction*. Puede especificar cero o más parámetros.

*parameterN*

## Valor devuelto

Ninguno.

## Descripción

Método; invoca la función representada por un objeto *Function*. Cada función de *ActionScript* se representa mediante un objeto *Function*, de modo que todas las funciones admiten este método.

En la mayoría de los casos, puede utilizarse el operador de llamada de función (*()*) en lugar de este método. El operador de llamada de función produce código preciso y fácil de leer. Este método es de gran utilidad cuando debe controlarse de forma explícita el parámetro *this* de la invocación de la función. Normalmente, si se invoca una función como método de un objeto, *this* se establece en *myObject* dentro del cuerpo de la función como el caso siguiente:

```
myObject.myMethod(1, 2, 3);
```

En algunas situaciones, es posible que desee que *this* haga referencia a otro elemento; por ejemplo, si debe invocarse una función como un método de un objeto, pero en realidad no se almacena como método de dicho objeto.

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

Puede pasar el valor *null* para el parámetro *thisObject* para invocar una función como función regular y no como método de un objeto. Por ejemplo, las llamadas de función siguientes son equivalentes:

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

## Ejemplo

En este ejemplo se utiliza *Function.call()* para hacer que una función se comporte como método de otro objeto, sin almacenar la función en el objeto.

```
function MyObject() {
}
function MyMethod(obj) {
    trace("this == obj? " + (this == obj));
}
var obj = new MyObject();
MyMethod.call(obj, obj);
```

La acción *trace()* envía el código siguiente al panel Salida:

```
this == obj? true
```

# Function.prototype

## Disponibilidad

Flash Player 5. Si utiliza ActionScript 2.0, no necesita usar esta propiedad; refleja la implementación de herencia de ActionScript 1.

## Sintaxis

*myFunction.prototype*

## Descripción

Propiedad; en una función constructora ActionScript 1, la propiedad `prototype` hace referencia a un objeto que es el prototipo de la clase construida. Cada instancia de la clase que crea la función constructora hereda todas las propiedades y los métodos del objeto prototipo.

# ge (mayor o igual que; específico para cadenas)

## Disponibilidad

Flash Player 4. Este operador se eliminó en Flash 5 y se sustituyó por el operador `>=` (mayor o igual que).

## Sintaxis

*expression1 ge expression2*

## Parámetros

*expression1, expression2* Números, cadenas o variables.

## Valor devuelto

Ninguno.

## Descripción

Operador (comparación); compara la representación de cadena de *expression1* con la representación de cadena de *expression2* y devuelve `true` si *expression1* es mayor o igual que *expression2*; de lo contrario, devuelve `false`.

## Véase también

`>=` (mayor o igual que)

# get

## Disponibilidad

Flash Player 6.

## Sintaxis

```
function get property() {  
    // las sentencias se escriben aquí  
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

### Parámetros

*property* Palabra que desee utilizar para hacer referencia a la propiedad a la que va a acceder `get`; este valor debe ser el mismo que el utilizado en el comando `set` correspondiente.

### Valor devuelto

El valor de la propiedad especificado por *propertyName*.

### Descripción

Palabra clave; permite "obtener" de manera implícita propiedades asociadas a objetos basados en clases definidos en archivos de clase externos. El uso de métodos `get` implícitos permite acceder a las propiedades de los objetos sin acceder directamente a los objetos. los métodos `get/set` implícitos son la abreviatura sintáctica del método `Object.addProperty()` en ActionScript 1.

Para más información, consulte [“Métodos `get/set` implícitos” en la página 179](#).

### Véase también

`Object.addProperty()`, `set`

## getProperty

### Disponibilidad

Flash Player 4.

### Sintaxis

```
getProperty(my_mc, property)
```

### Parámetros

*my\_mc* Nombre de instancia de un clip de película para el que se recupera la propiedad.

*property* Propiedad de un clip de película.

### Valor devuelto

El valor de la propiedad especificada.

### Descripción

Función; devuelve el valor de la propiedad especificada para el clip de película *my\_mc*.

### Ejemplo

En el ejemplo siguiente se recupera la coordenada de eje horizontal (`_x`) del clip de película *my\_mc* y se asigna a la variable *my\_mc\_x*:

```
my_mc_x = getProperty(_root.my_mc, _x);
```

## getTimer

### Disponibilidad

Flash Player 4.

### Sintaxis

```
getTimer()
```

### Parámetros

Ninguno.

### Valor devuelto

El número de milisegundos que han transcurrido desde que se inició la reproducción del archivo SWF.

### Descripción

Función; devuelve el número de milisegundos que han transcurrido desde que se inició la reproducción del archivo SWF.

## getURL()

### Disponibilidad

Flash 2. Las opciones GET y POST están disponibles solamente en Flash Player 4 y versiones posteriores del reproductor.

### Sintaxis

```
getURL(url [, window [, "variables"]])
```

### Parámetros

*url* Dirección URL de la que se obtiene el documento.

*window* Argumento opcional que especifica la ventana o el fotograma HTML en el que debería cargarse el documento. Puede especificar el nombre de una ventana específica o seleccionar uno de los nombres de destino reservados siguientes:

- `_self` especifica el fotograma actual de la ventana activa.
- `_blank` especifica una nueva ventana.
- `_parent` especifica el fotograma principal del fotograma actual.
- `_top` especifica el fotograma de nivel superior de la ventana actual.

*variables* Método GET o POST para el envío de variables. Si no hay variables, omita este parámetro. El método GET adjunta las variables al final de la dirección URL y sirve para pequeñas cantidades de variables. El método POST envía las variables en un encabezado HTTP separado y se usa para el envío de cadenas largas de variables.

### Valor devuelto

Ninguno.



## Descripción

Función; carga un documento de una URL específica en una ventana o pasa variables a otra aplicación en una URL definida. Para probar esta acción, asegúrese de que el archivo que se va a cargar se encuentra en la ubicación especificada. Para utilizar una dirección URL absoluta (por ejemplo, *http://www.myserver.com*), necesita una conexión de red.

## Ejemplo

En este ejemplo se carga una nueva URL en una ventana del navegador vacía. La acción `getURL()` utiliza la variable `incomingAd` como el parámetro *url* para que pueda cambiar la URL cargada sin tener que editar el archivo SWF. El valor de la variable `incomingAd` se pasa antes a Flash en el archivo SWF que utiliza una acción `loadVariables()`.

```
on(release) {  
    getURL(incomingAd, "_blank");  
}
```

## Véase también

[loadVariables\(\)](#), [XML.send\(\)](#), [XML.sendAndLoad\(\)](#), [XMLSocket.send\(\)](#)

# getVersion

## Disponibilidad

Flash Player 5.

## Sintaxis

```
getVersion()
```

## Parámetros

Ninguno.

## Valor devuelto

Una cadena que contiene información sobre la versión de Flash Player y la plataforma.

## Descripción

Función, devuelve una cadena que contiene la información de la versión de Flash Player y de la plataforma.

La función `getVersion` sólo devuelve información para Flash Player 5 o versiones posteriores.

## Ejemplo

A continuación, se muestra un ejemplo de una cadena devuelta por la función `getVersion`:

```
WIN 5,0,17,0
```

Esto indica que la plataforma es Microsoft Windows y el número de la versión de Flash Player es versión mayor 5, versión menor 17 (5.0r17).

## Véase también

[System.capabilities.os](#), [System.capabilities.version](#)

## **\_global object**

### **Disponibilidad**

Flash Player 6.

### **Sintaxis**

`_global.identifier`

### **Parámetros**

Ninguno.

### **Valor devuelto**

Referencia al objeto global que contiene las clases básicas de ActionScript, como por ejemplo, String, Object, Math y Array.

### **Descripción**

Identificador; crea objetos, clases o variables globales. Por ejemplo, puede crear una biblioteca que se presente como objeto global de ActionScript, del mismo modo que un objeto Math o Date. A diferencia de las variables y las funciones declaradas en la línea de tiempo o localmente, las variables y las funciones globales están visibles para cada línea de tiempo y ámbito en el archivo SWF, siempre que no haya identificadores con los mismos nombres en ámbitos interiores que los anulen.

### **Ejemplo**

En el ejemplo siguiente se crea una función `factorial()` de nivel superior, disponible en cada línea de tiempo y ámbito de un archivo SWF:

```
_global.factorial = function (n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return n * factorial(n-1);  
    }  
}
```

### **Véase también**

[var](#), [set variable](#)

## **gotoAndPlay()**

### **Disponibilidad**

Flash 2.

### **Sintaxis**

`gotoAndPlay([scene,] frame)`

### **Parámetros**

*scene* Cadena opcional que especifica el nombre de la escena a la que se envía la cabeza lectora.

*frame* Un número que representa el número de fotograma, o una cadena que represente la etiqueta del fotograma, al que se envía la cabeza lectora.

### Valor devuelto

Ninguno.

### Descripción

Función; envía la cabeza lectora al fotograma especificado en una escena y se reproduce a partir de ese fotograma. Si no se especifica escena, la cabeza lectora va al fotograma especificado en la escena actual.

### Ejemplo

Cuando el usuario hace clic en un botón al que se ha asignado `gotoAndPlay()`, se envía la cabeza lectora al fotograma 16 de la escena actual y se inicia la reproducción.

```
on(release) {  
    gotoAndPlay(16);  
}
```

### Véase también

[MovieClip.gotoAndPlay\(\)](#)

## gotoAndStop()

### Disponibilidad

Flash 2.

### Sintaxis

```
gotoAndStop([scene], frame)
```

### Parámetros

*scene* Cadena opcional que especifica el nombre de la escena a la que se envía la cabeza lectora.

*frame* Un número que representa el número de fotograma, o una cadena que represente la etiqueta del fotograma, al que se envía la cabeza lectora.

### Valor devuelto

Ninguno.

### Descripción

Función; envía la cabeza lectora al fotograma especificado en una escena y lo detiene. Si no se especifica ninguna escena, la cabeza lectora se envía al fotograma en la escena actual.

### Ejemplo

Cuando el usuario hace clic en un botón al que se ha asignado `gotoAndStop()`, se envía la cabeza lectora al fotograma 5 de la escena actual y el archivo SWF deja de reproducirse.

```
on(release) {  
    gotoAndStop(5);  
}
```

### Véase también

[stop\(\)](#)

## gt (mayor que; específico para cadenas)

### Disponibilidad

Flash Player 4. Este operador se eliminó en Flash 5 y se sustituyó por el nuevo operador `>` (mayor que).

### Sintaxis

*expression1* gt *expression2*

### Parámetros

*expression1*, *expression2*    Números, cadenas o variables.

### Descripción

Operador (comparación); compara la representación de cadena de *expression1* con la representación de cadena de *expression2* y devuelve `true` si *expression1* es mayor que *expression2*; de lo contrario, devuelve `false`.

### Véase también

`>` (mayor que)

## \_highquality

### Disponibilidad

Flash Player 4; se ha eliminado a cambio de `_quality`.

### Sintaxis

`_highquality`

### Descripción

Propiedad eliminada (global); especifica el nivel de suavizado aplicado al archivo SWF actual. Especifique 2 (calidad óptima) para aplicar alta calidad con el suavizado de mapa de bits siempre activado. Especifique 1 (alta calidad) para aplicar suavizado; esto suavizará los mapas de bits si el archivo SWF no contiene animación. Especifique 0 (baja calidad) para evitar el suavizado.

### Ejemplo

```
_highquality = 1;
```

### Véase también

`_quality`, `toggleHighQuality()`

## if

### Disponibilidad

Flash Player 4.

### Sintaxis

```
if(condition) {  
    statement(s);  
}
```

## Parámetros

*condition* Expresión que da como resultado `true` o `false`.

*statement(s)* Instrucciones que se deben ejecutar si o cuando la condición da como resultado `true`.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; calcula el resultado de una condición para determinar la siguiente acción en un archivo SWF. Si la condición es `true`, Flash ejecuta las sentencias entre llaves (`{}`) que van a continuación de la condición. Si la condición es `false`, Flash omite las sentencias entre llaves y ejecuta las sentencias que siguen a dichas llaves. Utilice la acción `if` para crear lógica de ramificación en sus scripts.

## Ejemplo

En el ejemplo siguiente, la condición entre paréntesis comprueba si la variable `name` contiene el valor literal “Erica”. Si es así, se ejecuta la acción `play()` entre llaves.

```
if(name == "Erica"){  
    play();  
}
```

El siguiente ejemplo utiliza una acción `if` para comprobar si el usuario suelta en el archivo SWF un objeto arrastrable. Si el objeto se suelta antes de que transcurran 300 milisegundos tras arrastrarlo, la condición da como resultado `true` y se ejecutan las sentencias entre llaves. Estas sentencias definen variables para almacenar la nueva ubicación del objeto y para determinar con qué fuerza se soltó y la velocidad a la que se hizo. También se restablece la variable `timePressed`. Si el objeto se suelta una vez transcurridos 300 milisegundos desde que se arrastró, la condición da como resultado `false` y no se ejecuta ninguna sentencia.

```
if (getTimer()<timePressed+300) {  
    // si la condición es verdadera,  
    // el objeto se ha soltado.  
    // ¿cuál es la nueva ubicación de este objeto?  
    xNewLoc = this._x;  
    yNewLoc = this._y;  
    // ¿con qué fuerza se soltó?  
    xTravel = xNewLoc-xLoc;  
    yTravel = yNewLoc-yLoc;  
    // definición de la velocidad del objeto en función de  
    // la distancia que ha recorrido  
    xInc = xTravel/2;  
    yInc = yTravel/2;  
    timePressed = 0;  
}
```

## Véase también

`else`

# ifFrameLoaded

## Disponibilidad

Flash Player 3. La acción `ifFrameLoaded` se eliminó en Flash 5; Macromedia recomienda utilizar la propiedad `MovieClip._framesloaded`.

## Sintaxis

```
ifFrameLoaded([scene,] frame) {  
    statement(s);  
}
```

## Parámetros

*scene* Cadena opcional que especifica el nombre de la escena que debe cargarse.

*frame* El número de fotograma o la etiqueta de fotograma que debe cargarse antes de ejecutar la sentencia siguiente.

*statement(s)* Instrucciones que se deben ejecutar si se carga la escena especificada o la escena y el fotograma indicados.

## Valor devuelto

Ninguno.

## Descripción

Acción desfasada; comprueba si el contenido de un fotograma específico está disponible localmente. Utilice `ifFrameLoaded` para empezar a reproducir una animación sencilla mientras se descarga el resto del archivo SWF en el sistema local. La diferencia entre utilizar `_framesloaded` y `ifFrameLoaded` es que `_framesloaded` le permite agregar sus propias sentencias `if` o `else`.

## Véase también

[MovieClip.\\_framesloaded](#)

# implements

## Disponibilidad

Flash Player 6.

## Sintaxis

```
myClass implements interface01 [, interface02, ...]
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Descripción

Palabra clave; define una clase que debe proporcionar implementaciones para todos los métodos definidos en la interfaz (o interfaces) que se implementa. Para más información, consulte [“Interfaces como tipos de datos” en la página 174](#).

## Ejemplo

Véase [interface](#).

## Véase también

[class](#), [extends](#), [interface](#)

# import

## Disponibilidad

Flash Player 6.

## Sintaxis

```
import className
```

```
import packageName.*
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Se admite esta sentencia en el panel Acciones, así como en archivos de clase externos.

## Parámetros

*className* El nombre completo de una clase que se ha definido en un archivo de clase externo.

*packageName* Un directorio en el que se han almacenado archivos de clase relacionados.

## Descripción

Palabra clave; permite acceder a clases sin especificar sus nombres completos. Por ejemplo, si desea utilizar la clase `macr.util.users.UserClass.as` en un script, debe referirse a ella por su nombre completo o bien importarla; si la importa, puede referirse a ella por el nombre de clase:

```
// antes de importarla
var myUser:UserClass = new macr.util.users.UserClass();
// después de importarla
import macr.util.users.UserClass;
var myUser:UserClass = new UserClass();
```

Si hay varias clases de archivos en el directorio al que desea acceder, puede importarlos todos mediante una sola sentencia:

```
import macr.util.users.*;
```

Debe emitir la sentencia `import` antes de intentar acceder a la clase importada sin especificar su nombre por completo.

Si importa una clase pero no la utiliza en el script, la clase no se exporta como parte del archivo SWF. Esto significa que puede importar paquetes de gran tamaño sin preocuparse del tamaño del archivo SWF; el código de bytes asociado a una clase sólo se incluye en un archivo SWF si esa clase se utiliza realmente.

La sentencia `import` sólo se aplica al script actual (fotograma u objeto) en el que se llama. Por ejemplo, supongamos que en el fotograma 1 de un documento de Flash importa todas las clases en el paquete `macr.util`. En ese fotograma, puede hacer referencia a las clases del paquete mediante sus nombres simples.

```
// En el fotograma 1 de un FLA:  
import macr.util.*;  
  
var myFoo:foo = new foo();
```

Sin embargo, en otro script de fotograma se necesita hacer referencia a las clases de ese paquete con sus nombres completos (`var myFoo:foo = new macr.util.foo();`) o bien deberá añadir una sentencia `import` al otro fotograma que importa las clases de ese paquete.

Para más información sobre la importación, consulte [“Importación de clases” en la página 178](#) y [“Utilización de paquetes” en la página 177](#).

## #include

### Disponibilidad

Flash Player 4.

### Sintaxis

```
#include "[path] filename.as"
```

**Nota:** no utilice un punto y coma (;) al final de la línea que contiene la sentencia `#include`.

### Parámetros

*[path] filename.as* Nombre de archivo y ruta opcional para el script que debe añadirse al panel Accciones; *.as* es la extensión de archivo recomendada.

### Valor devuelto

Ninguno.

### Descripción

Directiva del compilador: incluye el contenido del archivo especificado, como si los comandos del archivo formasen parte del propio script que llama. La directiva `#include` se invoca durante la compilación. Por lo tanto, si realiza cambios en un archivo externo, debe guardarlo y volver a compilar todos los archivos FLA que lo utilizan.

Si utiliza el botón Revisar sintaxis con un script que contiene sentencias `#include`, también se comprueba la sintaxis de los archivos incluidos.

Puede utilizar `#include` en los archivos FLA y en archivos de script externos, pero no en archivos de clase `ActionScript 2.0`.

Puede optar por no especificar ninguna ruta, especificar una ruta relativa o una ruta absoluta para el archivo que debe incluirse.

- Si no especifica ninguna ruta, el archivo AS debe residir en el mismo directorio que el archivo FLA o el script que contiene la sentencia `#include`.
- Para especificar una ruta para el archivo AS relativa al archivo FLA o al script, utilice un punto (.) para indicar el directorio actual, dos puntos (..) para indicar el directorio principal y barras diagonales (/). Consulte los ejemplos que aparecen a continuación.
- Para especificar una ruta absoluta para el archivo AS, utilice el formato que admite la plataforma (Macintosh o Windows). Consulte los ejemplos que aparecen a continuación. No obstante, este uso no se recomienda ya que requiere que la estructura de directorios sea la misma en cada uno de los equipos que se utilicen para compilar el script.



## Ejemplo

En los ejemplos siguientes se muestran distintas formas de especificar una ruta o un archivo para incluirlo en el script.

```
// Tenga en cuenta que las sentencias #include no finalizan con punto y coma
// (;)
// El archivo AS se encuentra en el mismo directorio que el archivo FLA o el
// script
#include "init_script.as"

// El archivo AS se encuentra en un subdirectorio del directorio
// que contiene el archivo FLA o el script
// El subdirectorio se denomina "FLA_includes"
#include "FLA_includes/init_script.as"

// El archivo AS se encuentra en un directorio del mismo nivel que el del
// archivo FLA o el script
// El directorio se denomina "ALL_includes"
#include "../ALL_includes/init_script.as"

// El archivo AS se especifica mediante una ruta absoluta en Windows
// Observe el uso de barras diagonales, no de barras inversas
#include "C:/Flash_scripts/init_script.as"

// El archivo AS se especifica mediante una ruta absoluta en Macintosh
#include "Mac HD:Flash_scripts:init_script.as"
```

## Véase también

[import](#)

## Infinity

### Disponibilidad

Flash Player 5.

### Sintaxis

Infinity

### Descripción

Constante; especifica el valor IEEE-754 que representa el infinito positivo. El valor de esta constante es el mismo que [Number.POSITIVE\\_INFINITY](#).

## -Infinity

### Disponibilidad

Flash Player 5.

### Sintaxis

-Infinity

### Descripción

Constante; especifica el valor IEEE-754 que representa el infinito negativo. El valor de esta constante es el mismo que [Number.NEGATIVE\\_INFINITY](#).

## #initclip

### Disponibilidad

Flash Player 6.

### Sintaxis

```
#initclip order
```

### Parámetros

*order* Número entero que especifica el orden de ejecución de los bloques del código `#initclip`. Este parámetro es opcional.

### Descripción

Directiva del compilador; indica el inicio de un bloque de acciones de inicialización. Si se inicializan varios clips al mismo tiempo, se puede utilizar el parámetro *order* para especificar cuál es la inicialización que se llevará a cabo en primer lugar. Las acciones de inicialización se ejecutan cuando se define un símbolo de clip de película. Si el clip de película es un símbolo exportado, las acciones de inicialización se ejecutarán antes que las acciones del fotograma 1 del archivo SWF. En caso contrario, se ejecutarán inmediatamente antes de las acciones del fotograma que contiene la primera instancia del símbolo de clip de película asociado.

Las acciones de inicialización se ejecutan sólo una vez durante la reproducción de un archivo SWF; deben utilizarse para inicializaciones que sólo se ejecutan una vez, como la definición y el registro de clases.

### Véase también

[#endinitclip](#)

## instanceof

### Disponibilidad

Flash Player 6.

### Sintaxis

```
object instanceof class
```

### Parámetros

*object* Objeto de `ActionScript`.

*class* Referencia a una función constructora de `ActionScript`, como `String` o `Date`.

### Valor devuelto

Si *objeto* es una instancia de *clase*, `instanceof` devuelve `true`; de lo contrario, `instanceof` devuelve `false`. Asimismo, `_global instanceof Object` devuelve `false`.

### Descripción

Operador; determina si un objeto pertenece a una clase específica. Comprueba si *object* es una instancia de *class*.

El operador `instanceof` no convierte los tipos primitivos en objetos envolventes. Por ejemplo, el código siguiente devuelve `true`:

```
new String("Hola") instanceof String;
```

Mientras que el código siguiente devuelve `false`:

```
"Hola" instanceof String;
```

#### Véase también

[typeof](#)

## int

#### Disponibilidad

Flash Player 4. Esta función se eliminó en Flash 5 y se sustituyó por [Math.round\(\)](#).

#### Sintaxis

```
int(value)
```

#### Parámetros

*value* Número que se va a redondear en un número entero.

#### Valor devuelto

Ninguno.

#### Descripción

Función; convierte un número decimal en el valor del número entero más cercano.

#### Véase también

[Math.floor\(\)](#)

## interface

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
interface InterfaceName {}  
interface InterfaceName [extends InterfaceName [, InterfaceName ...]] {}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Descripción

Palabra clave; define una interfaz. Una interfaz es parecida a una clase, con las diferencias importantes que se indican a continuación:

- Las interfaces sólo contienen declaraciones de los métodos, no de su implementación. Es decir que cada clase que implementa una interfaz debe proporcionar una implementación para cada uno de los métodos declarados en la interfaz.
- En una definición de interfaz sólo pueden utilizarse miembros públicos. Además, no se permiten miembros de clase e instancia.
- No se permiten las sentencias `get` y `set` en las definiciones de interfaz.

Para más información, consulte [“Creación y utilización de interfaces” en la página 173](#).

## Ejemplo

En el ejemplo siguiente se muestran varias formas de definir e implementar interfaces.

(archivos `.as` `Ia`, `B`, `C`, `Ib`, `D`, `Ic`, `E` del paquete de nivel superior)

```
// archivo Ia.as
interface Ia
{
    function k():Number;           // declaración de método únicamente
    function n(x:Number):Number; // sin implementación
}
// archivo B.as
class B implements Ia
{
    function k():Number {return 25;}
    function n(x:Number):Number {return x+5;}
}
// script externo o panel Acciones
mvar = new B();
trace(B.k());    // 25
trace(B.n(7));   // 12

// archivo c.as
class C implements Ia
{
    function k():Number {return 25;}
} // error: la clase debe implementar todos los métodos de interfaz

// archivo Ib.as
interface Ib
{
    function o():Void;
}
class D implements Ia, Ib
{
    function k():Number {return 15;}
    function n(x:Number):Number {return x*x;}
    function o():Void {trace("o");}
}

// script externo o panel Acciones
mvar = new D();
trace(D.k());    // 15
trace(D.n(7));   // 49
trace(D.o());    // "o"
```

```

interface Ic extends Ia
{
    function p():Void;
}
class E implements Ib, Ic
{
    function k():Number {return 25;}
    function n(x:Number):Number {return x+5;}
    function o():Void {trace("o");}
    function p():Void {trace("p");}
}

```

#### Véase también

[class](#), [extends](#), [implements](#)

## isFinite

### Disponibilidad

Flash Player 5.

### Sintaxis

`isFinite(expression)`

### Parámetros

*expression* Valor booleano, variable u otra expresión cuyo resultado se debe calcular.

### Valor devuelto

Valor booleano.

### Descripción

Función; calcula *expression* y devuelve `true` si se trata de un número finito o `false` se trata de un valor infinito o infinito negativo. La presencia de infinito o de infinito negativo indica una condición de error matemático como una división entre 0.

### Ejemplo

A continuación, se muestran ejemplos de los valores devueltos por `isFinite`:

```

isFinite(56)
// devuelve true

isFinite(Number.POSITIVE_INFINITY)
// devuelve false

```

## isNaN()

### Disponibilidad

Flash Player 5.

### Sintaxis

`isNaN(expression)`

### Parámetros

*expression* Valor booleano, variable u otra expresión cuyo resultado se debe calcular.

### Valor devuelto

Valor booleano.

### Descripción

Función; calcula el resultado del parámetro y devuelve `true` si el valor no es un número (NaN), lo que indica la presencia de errores matemáticos.

### Ejemplo

El código siguiente muestra los valores de retorno de la función `isNaN`.

```
isNaN("Tree")
// devuelve true

isNaN(56)
// devuelve false

isNaN(Number.POSITIVE_INFINITY)
// devuelve false
```

### Véase también

[NaN](#), [Number.NaN](#)

## Clase Key

### Disponibilidad

Flash Player 6.

### Descripción

La clase `Key` es una clase de nivel superior cuyos métodos y propiedades pueden utilizarse sin necesidad de un constructor. Utilice los métodos de la clase `Key` para crear una interfaz que pueda ser controlada por cualquier usuario con un teclado estándar. Las propiedades de la clase `Key` son constantes que representan las teclas que se utilizan con mayor frecuencia para controlar juegos. Para obtener una lista completa de los valores de códigos de tecla, véase el [Apéndice C, “Teclas del teclado y valores de códigos de tecla”](#), en la página 791.

## Resumen de métodos para la clase Key

Método	Descripción
<a href="#">Key.addListener()</a>	Registra un objeto para que reciba una notificación cuando se invoquen los métodos <code>onKeyDown</code> y <code>onKeyUp</code> .
<a href="#">Key.getAscii()</a>	Devuelve el valor ASCII de la última tecla presionada.
<a href="#">Key.getCode()</a>	Devuelve el código de tecla virtual de la última tecla presionada.
<a href="#">Key.isDown()</a>	Devuelve <code>true</code> si se presiona la tecla especificada en el parámetro.

Método	Descripción
<code>Key.isToggled()</code>	Devuelve <code>true</code> si está activada la tecla Bloq Num o Bloq Mayús.
<code>Key.removeListener()</code>	Elimina un objeto registrado previamente con <code>Key.addListener()</code> .

## Resumen de propiedades para la clase Key

Todas las propiedades de la clase Key son constantes.

Propiedad	Descripción
<code>Key.BACKSPACE</code>	Constante asociada con el valor del código de tecla para la tecla Retroceso (8).
<code>Key.CAPSLCK</code>	Constante asociada con el valor del código de tecla para la tecla Bloq Mayús (20).
<code>Key.CONTROL</code>	Constante asociada con el valor del código de tecla para la tecla Control (17).
<code>Key.DELETEKEY</code>	Constante asociada con el valor del código de tecla para la tecla Supr (46).
<code>Key.DOWN</code>	Constante asociada con el valor del código de tecla para la tecla Flecha abajo (40).
<code>Key.END</code>	Constante asociada con el valor del código de tecla para la tecla Fin (35).
<code>Key.ENTER</code>	Constante asociada con el valor del código de tecla para la tecla Intro (13).
<code>Key.ESCAPE</code>	Constante asociada con el valor del código de tecla para la tecla Escape (27).
<code>Key.HOME</code>	Constante asociada con el valor del código de tecla para la tecla Inicio (36).
<code>Key.INSERT</code>	Constante asociada con el valor del código de tecla para la tecla Insert (45).
<code>Key.LEFT</code>	Constante asociada con el valor del código de tecla para la tecla Flecha izquierda (37).
<code>Key.PGDN</code>	Constante asociada con el valor del código de tecla para la tecla Av Pág (34).
<code>Key.PGUP</code>	Constante asociada con el valor del código de tecla para la tecla Re Pág (33).
<code>Key.RIGHT</code>	Constante asociada con el valor del código de tecla para la tecla Flecha derecha (39).
<code>Key.SHIFT</code>	Constante asociada con el valor del código de tecla para la tecla Mayús (16).
<code>Key.SPACE</code>	Constante asociada con el valor del código de tecla para la Barra espaciadora (32).
<code>Key.TAB</code>	Constante asociada con el valor del código de tecla para la tecla Tabulador (9).
<code>Key.UP</code>	Constante asociada con el valor del código de tecla para la tecla Flecha arriba (38).

## Resumen de detectores de la clase **Key**

Método	Descripción
<code>Key.onKeyDown</code>	Recibe notificación cuando se presiona una tecla.
<code>Key.onKeyUp</code>	Recibe notificación cuando se suelta una tecla.

## Key.addListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Key.addListener (newListener)
```

### Parámetros

`newListener` Objeto con los métodos `onKeyDown` y `onKeyUp`.

### Valor devuelto

Ninguno.

### Descripción

Método; registra un objeto para que reciba una notificación de `onKeyDown` y `onKeyUp`. Al presionar o soltar una tecla, sin tener en cuenta la selección de entrada, se invoca el método `onKeyDown` u `onKeyUp` de todos los objetos de detección registrados con `addListener()`. Varios objetos pueden detectar notificaciones de teclado. Si el detector `newListener` ya está registrado, no se produce ningún cambio.

### Ejemplo

En este ejemplo se crea un nuevo objeto detector y define una función para `onKeyDown` y `onKeyUp`. La última línea utiliza el método `addListener()` para registrar el detector con el objeto `Key` para que pueda recibir notificación de los eventos presionar tecla y soltar tecla.

```
myListener = new Object();
myListener.onKeyDown = function () {
    trace ("Ha presionado una tecla.");
}
myListener.onKeyUp = function () {
    trace ("Ha soltado una tecla.");
}
Key.addListener(myListener);
```

En el ejemplo siguiente se asigna el método abreviado de teclado `Control+7` a un botón cuyo nombre de instancia es `myButton` y se ofrece información sobre el método abreviado disponible para los lectores de pantalla (consulte [\\_accProps](#)). En este ejemplo, cuando presiona `Control+7` la función `myOnPress` muestra el texto "hola" en el panel Salida; en el archivo, puede crear una función que realice una tarea de más relevancia.

```
function myOnPress() {
    trace( "hola" );
}

function myOnKeyDown() {
```



```

        if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) // 55 is key code for 7
        {
            Selection.setFocus( myButton );
            myButton.onPress();
        }
    }

    var myListener = new Object();
    myListener.onKeyDown = myOnKeyDown;
    Key.addListener( myListener );

    myButton.onPress = myOnPress;
    myButton._accProps.shortcut = "Ctrl+F"
    Accessibility.updateProperties();

```

#### Véase también

[Key.getCode\(\)](#), [Key.isDown\(\)](#), [Key.onKeyDown](#), [Key.onKeyUp](#), [Key.removeListener\(\)](#)

## Key.BACKSPACE

#### Disponibilidad

Flash Player 5.

#### Sintaxis

`Key.BACKSPACE`

#### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Retroceso (8).

## Key.CAPSLOCK

#### Disponibilidad

Flash Player 5.

#### Sintaxis

`Key.CAPSLOCK`

#### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Bloq Mayús (20).

## Key.CONTROL

#### Disponibilidad

Flash Player 5.

#### Sintaxis

`Key.CONTROL`

#### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Control (17).

## Key.DELETEKEY

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.DELETEKEY`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Supr (46).

## Key.DOWN

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.DOWN`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Flecha abajo (40).

## Key.END

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.END`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Fin (35).

## Key.ENTER

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.ENTER`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Intro (13).

# Key.ESCAPE

## Disponibilidad

Flash Player 5.

## Sintaxis

`Key.ESCAPE`

## Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Escape (27).

# Key.getAscii()

## Disponibilidad

Flash Player 5.

## Sintaxis

`Key.getAscii();`

## Parámetros

Ninguno.

## Valor devuelto

Número entero que representa el valor ASCII de la última tecla presionada.

## Descripción

Método; devuelve el código ASCII de la última tecla presionada o soltada. Los valores ASCII devueltos son valores del teclado inglés. Por ejemplo, si presiona Mayús+2, `Key.getAscii()` devuelve @ en un teclado japonés, igual que en un teclado inglés.

# Key.getCode()

## Disponibilidad

Flash Player 5.

## Sintaxis

`Key.getCode();`

## Parámetros

Ninguno.

## Valor devuelto

Número entero que representa el código de tecla de la última tecla presionada.

## Descripción

Método; devuelve el código de tecla virtual de la última tecla presionada. Para establecer una correspondencia entre el código de tecla devuelto y la tecla de un teclado estándar, véase el [Apéndice C](#), “Teclas del teclado y valores de códigos de tecla”, en la página 791.

## Key.HOME

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.HOME`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Inicio (36).

## Key.INSERT

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.INSERT`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Insert (45).

## Key.isDown()

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.isDown(keycode)`

### Parámetros

*keycode* Valor de código de tecla asignado a una tecla específica o a una propiedad de la clase `Key` asociada con una tecla específica. Para establecer una correspondencia entre el código de tecla devuelto y la tecla de un teclado estándar, véase el [Apéndice C](#), “Teclas del teclado y valores de códigos de tecla”, en la página 791.

### Valor devuelto

Valor booleano.

### Descripción

Método; devuelve `true` si se presiona la tecla especificada en *keycode* o `false`, si no se presiona. En Macintosh, los valores de código de tecla para las teclas Bloq Mayús y Bloq Num son idénticos.

## Ejemplo

El script siguiente permite al usuario controlar la ubicación de un clip de película.

```
onClipEvent(enterFrame) {  
    if(Key.isDown(Key.RIGHT)) {  
        this._x=_x+10;  
    } else if (Key.isDown(Key.DOWN)) {  
        this._y=_y+10;  
    }  
}
```

## Key.isToggled()

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.isToggled(keycode)`

### Parámetros

*keycode*    Código de tecla de Bloq Mayús (20) o Bloq Num (144).

### Valor devuelto

Valor booleano.

### Descripción

Método; devuelve `true` si la tecla Bloq Mayús o Bloq Num está activada o `false` si está desactivada. En Macintosh, los valores de código de tecla para las teclas Bloq Mayús y Bloq Num son idénticos.

## Key.LEFT

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.LEFT`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Flecha izquierda (37).

## Key.onKeyDown

### Disponibilidad

Flash Player 6.

### Sintaxis

`someListener.onKeyDown`

### Descripción

Detector, se notifica cuando se presiona una tecla. Para utilizar `onKeyDown`, debe crear un objeto detector. A continuación, puede definir una función para `onKeyDown` y utilizar el método `addListener()` para registrar el detector con el objeto `Key`, como en el caso siguiente:

```
somelister = new Object();
somelister.onKeyDown = function () { ... };
Key.addListener(somelister);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

### Véase también

[Key.addListener\(\)](#)

## Key.onKeyUp

### Disponibilidad

Flash Player 6.

### Sintaxis

```
somelister.onKeyUp
```

### Descripción

Detector; recibe notificación cuando se suelta una tecla. Para utilizar `onKeyUp` debe crear un objeto detector. A continuación, puede definir una función para `onKeyUp` y utilizar el método `addListener()` para registrar el detector con el objeto `Key`, como en el ejemplo siguiente:

```
somelister = new Object();
somelister.onKeyUp = function () { ... };
Key.addListener(somelister);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

### Véase también

[Key.addListener\(\)](#)

## Key.PGDN

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Key.PGDN
```

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Av Pág (34).

## Key.PGUP

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.PGUP`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Re Pág (33).

## Key.removeListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

`Key.removeListener (listener)`

### Parámetros

*listener* Un objeto.

### Valor devuelto

Si *listener* se ha eliminado correctamente, el método devuelve `true`. Si *listener* no se ha eliminado correctamente, por ejemplo, si *listener* no se encontraba en la lista de detectores del objeto `Key`, el método devuelve `false`.

### Descripción

Método; elimina un objeto previamente registrado con `Key.addListener()`.

## Key.RIGHT

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.RIGHT`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Flecha derecha (39).

## Key.SHIFT

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.SHIFT`

### Descripción

Propiedad; constante asociada con el valor del código de tecla para la tecla Mayús (16).

## Key.SPACE

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.SPACE`

### Descripción

Propiedad; constante asociada con el valor del código de tecla de la Barra espaciadora (32).

## Key.TAB

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.TAB`

### Descripción

Propiedad; constante asociada con el valor del código de tecla de la tecla Tabulador (9).

## Key.UP

### Disponibilidad

Flash Player 5.

### Sintaxis

`Key.UP`

### Descripción

Propiedad; constante asociada con el valor del código de tecla de la tecla Flecha arriba (38).

## le (menor o igual que; específico para cadenas)

### Disponibilidad

Flash Player 4. Este operador se eliminó en Flash 5 y se sustituyó por el operador `<=` (menor o igual que).

### Sintaxis

*expression1 le expression2*

### Parámetros

*expression1, expression2*    Números, cadenas o variables.



### Valor devuelto

Ninguno.

### Descripción

Operador (de comparación); compara *expression1* con *expression2* y devuelve un valor `true` si *expression1* es menor o igual que *expression2*; en caso contrario, devuelve un valor `false`.

### Véase también

`<=` (menor o igual que)

## length

### Disponibilidad

Flash Player 4. Esta función, junto con todas las funciones de cadena, se eliminó en Flash 5. Macromedia recomienda utilizar los métodos de la clase `String` y la propiedad `String.length` para realizar las mismas operaciones.

### Sintaxis

```
length(expression)  
length(variable)
```

### Parámetros

*expression*    Una cadena.  
*variable*    Nombre de una variable.

### Valor devuelto

La longitud de la cadena o el nombre de variable especificado.

### Descripción

Función de cadena; devuelve la longitud de la cadena o nombre de variable especificado.

### Ejemplo

El ejemplo siguiente devuelve el valor de la cadena "Hola".  
`length("Hola");`  
El resultado es 5.

### Véase también

`" "` (delimitador de cadena), [Clase String](#), [String.length](#)

## \_level

### Disponibilidad

Flash Player 4.

### Sintaxis

```
_levelN
```

## Descripción

Identificador; una referencia a la línea de tiempo raíz de `_levelN`. Debe utilizar la acción `loadMovieNum()` para cargar archivos SWF en Flash Player antes de usar la propiedad `_level` para utilizarlas. También puede utilizar `_levelN` para usar un archivo SWF cargado en el nivel asignado por *N*.

El archivo SWF inicial cargado en una instancia de Flash Player se carga automáticamente en `_level0`. El archivo SWF de `_level0` establece la velocidad y el tamaño de los fotogramas, así como el color de fondo para todos los demás archivos SWF cargados posteriormente. Éstos se apilan en niveles superiores al del archivo SWF de `_level0`.

Debe asignar un nivel a cada archivo SWF que cargue en Flash Player mediante `loadMovieNum()`. Puede asignar niveles en cualquier orden. Si asigna un nivel que ya contiene un archivo SWF (incluido `_level0`), el archivo SWF de ese nivel se descargará y se sustituirá por el nuevo archivo SWF.

## Ejemplo

El ejemplo siguiente detiene la cabeza lectora de la línea de tiempo principal del archivo SWF de `_level9`.

```
_level9.stop();
```

En el ejemplo siguiente se envía la cabeza lectora de la línea de tiempo principal del archivo SWF de `_level14` al fotograma 5. El archivo SWF de `_level14` debe haberse cargado anteriormente con una acción `loadMovieNum()`.

```
_level14.gotoAndStop(5);
```

## Véase también

`loadMovie()`, `MovieClip.swapDepths()`

# loadMovie()

## Disponibilidad

Flash Player 3.

## Sintaxis

```
loadMovie("url",target [, method])
```

## Parámetros

*url* URL absoluta o relativa del archivo SWF o JPEG que se debe cargar. Una ruta relativa debe ser relativa respecto al archivo SWF del nivel 0. Los URL absolutos deben incluir la referencia al protocolo, como `http://` o `file:///`.

*target* Ruta de acceso a un clip de película de destino. El clip de película de destino se sustituirá por la imagen o el archivo SWF cargado.

*method* Parámetro opcional que especifica un método HTTP para enviar variables. El parámetro debe ser la cadena `GET` o `POST`. Si no hay ninguna variable para enviar, no incluya este parámetro. El método `GET` adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método `POST` envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

## Valor devuelto

Ninguno.

## Descripción

Función; carga un archivo SWF o JPEG en Flash Player mientras se reproduce el archivo SWF original.

**Sugerencia:** si desea supervisar el progreso de la operación de descarga de un archivo, utilice `MovieClipLoader.loadClip()` en lugar de esta función.

La función `loadMovie()` permite ver varios archivos SWF al mismo tiempo y pasar de uno a otro sin cargar otro documento HTML. Sin la función `loadMovie()` Flash Player muestra un solo archivo SWF y después se cierra.

Si desea cargar un archivo SWF o JPEG en un nivel específico, utilice `loadMovieNum()` en lugar de `loadMovie()`.

Si un archivo SWF se carga en un clip de película de destino, puede utilizar la ruta de destino de dicho clip de película para acceder al archivo SWF cargado. Los archivos SWF y las imágenes que se cargan en un destino heredan las propiedades de posición, rotación y escala del clip de película de destino. La esquina superior izquierda de la imagen o archivo SWF cargado se alinea con el punto de registro del clip de película de destino. Como alternativa, si el destino es la línea de tiempo `_root`, la esquina superior izquierda de la imagen o del archivo SWF se alineará con la esquina superior izquierda del escenario.

Utilice `unloadMovie()` para eliminar archivos SWF que se hayan cargado con `loadMovie()`.

## Ejemplo

La sentencia `loadMovie()` siguiente está asociada al botón de navegación con la etiqueta Productos. Hay un clip de película invisible en el escenario con el nombre de instancia `dropZone`. La función `loadMovie()` utiliza este clip de película como parámetro de destino para cargar los productos del archivo SWF en la posición correcta del escenario.

```
on(release) {  
    loadMovie("products.swf",_root.dropZone);  
}
```

En el ejemplo siguiente se carga una imagen JPEG que se encuentra en el mismo directorio que el archivo SWF que llama a la función `loadMovie()`:

```
loadMovie("image45.jpeg", "ourMovieClip");
```

## Véase también

`_level`, `loadMovieNum()`, `MovieClipLoader.loadClip()`, `unloadMovie()`

# loadMovieNum()

## Disponibilidad

Flash Player 4. Los archivos de Flash 4 abiertos en Flash 5 o en una versión posterior se convierten para utilizar la sintaxis correcta.

## Sintaxis

```
loadMovieNum("url",level [, variables])
```

## Parámetros

*url* URL absoluta o relativa del archivo SWF o JPEG que se debe cargar. Una ruta relativa debe ser relativa al archivo SWF del nivel 0. Para utilizarlos en el reproductor Flash Player independiente o para realizar pruebas en el modo de probar película en la aplicación de edición Flash, todos los archivos SWF deben estar almacenados en la misma carpeta, y los nombres de archivo no pueden incluir especificaciones de carpeta ni de unidad de disco.

*level* Número entero que especifica el nivel de Flash Player en el que se cargará el archivo SWF.

*variables* Parámetro opcional que especifica un método HTTP para enviar variables. El parámetro debe ser la cadena GET o POST. Si no hay ninguna variable para enviar, no incluya este parámetro. El método GET adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método POST envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

## Valor devuelto

Ninguno.

## Descripción

Función; carga un archivo SWF o JPEG en un nivel de Flash Player mientras se reproduce el archivo SWF cargado originalmente.

**Sugerencia:** si desea supervisar el progreso de la operación de descarga de un archivo, utilice `MovieClipLoader.loadClip()` en lugar de esta función.

Por lo general, Flash Player muestra un solo archivo SWF y después se cierra. La acción `loadMovieNum()` permite ver varios archivos SWF al mismo tiempo y pasar de uno a otro sin cargar otro documento HTML.

Si desea especificar un destino en lugar de un nivel, utilice `loadMovie()` en lugar de `loadMovieNum()`.

Flash Player tiene un orden de apilamiento de niveles que empieza en el nivel 0. Estos niveles son como capas de acetato; es decir, son transparentes excepto para los objetos de cada nivel. Si utiliza la acción `loadMovieNum()`, debe especificar el nivel de Flash Player en el que se cargará el archivo SWF. Una vez que se ha cargado la película SWF en un nivel, puede utilizar la sintaxis `_levelN`, siendo *N* el número de nivel, para especificar el destino del archivo SWF.

Al cargar un archivo SWF, puede especificar cualquier número de nivel y puede cargar archivos SWF en un nivel en el que ya se haya cargado un archivo SWF. Al hacerlo, el nuevo archivo SWF sustituye al existente. Si carga un archivo SWF en el nivel 0, todos los demás niveles se descargan y la película del nivel 0 se sustituirá por el nuevo archivo. El archivo SWF del nivel 0 establece la velocidad y el tamaño de los fotogramas y el color de fondo para todos los demás archivos SWF cargados.

La acción `loadMovieNum()` también permite cargar archivos JPEG en un archivo SWF durante su reproducción. Tanto en las imágenes como en los archivos SWF, la esquina superior izquierda de la imagen se alinea con la esquina superior izquierda del escenario al cargar el archivo. En ambos casos, el archivo cargado también hereda las propiedades de rotación y escala, y el contenido original se sobrescribe.

Utilice `unloadMovieNum()` para eliminar archivos SWF o imágenes que se hayan cargado con `loadMovieNum()`.

## Ejemplo

En este ejemplo se carga la imagen JPEG `image45.jpg` en el nivel 2 de Flash Player.

```
loadMovieNum("http://www.blag.com/image45.jpg", 2);
```

## Véase también

```
loadMovie(), unloadMovieNum(), _level
```

# loadVariables()

## Disponibilidad

Flash Player 4; comportamiento modificado en Flash Player 7.

## Sintaxis

```
loadVariables ("url" , target [, variables])
```

## Parámetros

*url* URL absoluta o relativa donde están ubicadas las variables. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*target* Ruta de destino a un clip de película que recibe las variables cargadas.

*variables* Parámetro opcional que especifica un método HTTP para enviar variables. El parámetro debe ser la cadena GET o POST. Si no hay ninguna variable para enviar, no incluya este parámetro. El método GET adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método POST envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

## Valor devuelto

Ninguno.

## Descripción

Función; lee los datos de un archivo externo, como un archivo de texto o texto generado por un script CGI, Active Server Pages (ASP) o Personal Home Page (PHP), o un script Perl, y establece los valores para las variables de un clip de película de destino. Esta acción también puede utilizarse para actualizar variables en el archivo SWF activo con nuevos valores.

El texto de la URL especificada debe tener el formato MIME estándar `application/x-www-form-urlencoded` (formato estándar que se utiliza en los scripts CGI). Se puede especificar cualquier número de variables. Por ejemplo, la siguiente frase define varias variables:

```
empresa=Macromedia&dirección=600+Townsend&ciudad=San+Francisco&zip=94103
```

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables desde archivos SWF que también estén en `www.someDomain.com`. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Para cargar variables en un nivel específico, utilice `loadVariablesNum()` en lugar de `loadVariables()`.

### Ejemplo

Este ejemplo carga información de un archivo de texto en los campos de texto del clip de película `varTarget` de la línea de tiempo principal. Los nombres de variables de los campos de texto deben coincidir con los nombres de variables del archivo `data.txt`.

```
on(release) {  
    loadVariables("data.txt", "_root.varTarget");  
}
```

### Véase también

[loadVariablesNum\(\)](#), [loadMovie\(\)](#), [loadMovieNum\(\)](#), [getURL\(\)](#), [MovieClip.loadMovie\(\)](#), [MovieClip.loadVariables\(\)](#)

## loadVariablesNum()

### Disponibilidad

Flash Player 4. Los archivos de Flash 4 abiertos en Flash 5 o en una versión posterior se convertirán para utilizar la sintaxis correcta. Comportamiento modificado en Flash Player 7.

### Sintaxis

```
loadVariablesNum ("url" ,level [, variables])
```

### Parámetros

*url* URL absoluta o relativa donde están ubicadas las variables. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*level* Número entero que especifica el nivel de Flash Player para recibir las variables.

*variables* Parámetro opcional que especifica un método HTTP para enviar variables. El parámetro debe ser la cadena `GET` o `POST`. Si no hay ninguna variable para enviar, no incluya este parámetro. El método `GET` adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método `POST` envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

### Valor devuelto

Ninguno.

## Descripción

Función; lee los datos de un archivo externo, como un archivo de texto o texto generado por un script CGI, Active Server Pages (ASP), Personal Home Page (PHP) o un script Perl, y establece los valores para las variables en un nivel de Flash Player. También puede utilizar esta función para actualizar las variables del archivo SWF activo con nuevos valores.

El texto de la URL especificada debe tener el formato MIME estándar *application/x-www-form-urlencoded* (formato estándar que se utiliza en los scripts CGI). Se puede especificar cualquier número de variables. Por ejemplo, la siguiente frase define varias variables:

```
empresa=Macromedia&dirección=600+Townsend&ciudad=San+Francisco&zip=94103
```

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de [www.someDomain.com](http://www.someDomain.com) puede cargar variables desde un archivo SWF de [store.someDomain.com](http://store.someDomain.com) porque ambos archivos se encuentran en el mismo superdominio que [someDomain.com](http://someDomain.com).

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de [www.someDomain.com](http://www.someDomain.com) sólo puede cargar variables desde archivos SWF que también estén en [www.someDomain.com](http://www.someDomain.com). Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Para cargar las variables en un clip de película de destino, utilice `loadVariables()` en lugar de `loadVariablesNum()`.

## Ejemplo

Este ejemplo carga información de un archivo de texto en los campos de texto de la línea de tiempo principal del archivo SWF en el nivel 0 de Flash Player. Los nombres de variables de los campos de texto deben coincidir con los nombres de variables del archivo `data.txt`.

```
on(release) {  
    loadVariablesNum("data.txt", 0);  
}
```

## Véase también

[getURL\(\)](#), [loadMovie\(\)](#), [loadMovieNum\(\)](#), [loadVariables\(\)](#), [MovieClip.loadMovie\(\)](#), [MovieClip.loadVariables\(\)](#)

# Clase LoadVars

## Disponibilidad

Flash Player 6.

## Descripción

La clase `LoadVars` es una alternativa a la función `loadVariables()` para transferir variables entre una aplicación Flash y un servidor.

Puede utilizar la clase `LoadVars` para obtener la verificación de los procesos de carga de datos realizada correctamente, de las indicaciones de progreso y datos de flujo durante el proceso de descarga. La clase `LoadVars` funciona de forma parecida a la [Clase XML](#); utiliza los métodos `load()`, `send()` y `sendAndLoad()` para establecer comunicación con un servidor. La diferencia principal entre la clase `LoadVars` y la clase `XML` consiste en que `LoadVars` transfiere pares de nombre y valor de `ActionScript`, en lugar de un árbol XML DOM almacenado en el objeto `XML`.

La clase `LoadVars` sigue las mismas restricciones de seguridad que la clase `XML`.

## Resumen de métodos para la clase `LoadVars`

Método	Descripción
<code>LoadVars.setRequestHeader()</code>	Añade o cambia los encabezados HTTP para las operaciones <code>POST</code> .
<code>LoadVars.getBytesLoaded()</code>	Devuelve el número de bytes descargados por <code>LoadVars.load()</code> o <code>LoadVars.sendAndLoad()</code> .
<code>LoadVars.getBytesTotal()</code>	Devuelve el número total de bytes que se descargarán mediante el método <code>load</code> o <code>sendAndLoad</code> .
<code>LoadVars.load()</code>	Descarga variables de una URL especificada.
<code>LoadVars.send()</code>	Publica variables de un objeto <code>LoadVars</code> en una URL.
<code>LoadVars.sendAndLoad()</code>	Publica variables de un objeto <code>LoadVars</code> en una URL y descarga la respuesta del servidor en un objeto de destino.
<code>LoadVars.toString()</code>	Devuelve una cadena con codificación URL que contiene todas las variables enumerables del objeto <code>LoadVars</code> .

## Resumen de propiedades para la clase `LoadVars`

Propiedad	Descripción
<code>LoadVars.contentType</code>	Indica el tipo MIME de los datos.
<code>LoadVars.loaded</code>	Valor booleano que indica si se ha realizado una operación <code>load</code> o <code>sendAndLoad</code> .

## Resumen de controladores de eventos para la clase `LoadVars`

Controlador de eventos	Descripción
<code>LoadVars.onData</code>	Se invoca cuando los datos se han descargado completamente del servidor, o cuando se produce un error durante el proceso de descarga de datos de un servidor.
<code>LoadVars.onLoad</code>	Se invoca cuando se ha realizado una operación <code>load</code> o <code>sendAndLoad</code> .

## Constructor para la clase `LoadVars`

### Disponibilidad

Flash Player 6.



### Sintaxis

```
new LoadVars()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto LoadVars. A continuación, puede utilizar los métodos del objeto LoadVars para enviar y cargar datos.

### Ejemplo

En el ejemplo siguiente se crea un objeto LoadVars denominado `my_lv`:

```
var my_lv = new LoadVars();
```

## LoadVars.setRequestHeader()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.setRequestHeader(headerName, headerValue)  
my_lv.setRequestHeader(["headerName_1", "headerValue_1" ... "headerName_n",  
    "headerValue_n"])
```

### Parámetros

*headerName* Nombre de encabezado de una solicitud HTTP.

*headerValue* Valor asociado con *headerName*.

### Valor devuelto

Ninguno.

### Descripción

Método; añade o cambia encabezados de solicitud HTTP (tales como Content-Type o SOAPAction) enviados con acciones POST. En la primera sintaxis, se pasan dos cadenas al método: *headerName* y *headerValue*. En la segunda sintaxis, se pasa una matriz de cadenas, en las que se alternan los nombres y los valores de encabezado.

Si se realizan varias llamadas para establecer el mismo nombre de encabezado, cada valor sucesivo sustituirá el valor establecido en la llamada anterior.

Con este método *no se pueden* añadir ni cambiar los encabezados HTTP estándar siguientes: Accept-Ranges, Age, Allow, Allowed, Connection, Content-Length, Content-Location, Content-Range, ETag, Host, Last-Modified, Locations, Max-Forwards, Proxy-Authenticate, Proxy-Authorization, Public, Range, Retry-After, Server, TE, Trailer, Transfer-Encoding, Upgrade, URI, Vary, Via, Warning y WWW-Authenticate.

## Ejemplo

En este ejemplo se añade un encabezado HTTP personalizado denominado `SOAPAction` con el valor `Foo` al objeto `my_lv`.

```
my_lv.setRequestHeader("SOAPAction", "'Foo'");
```

En el ejemplo siguiente, se crea una matriz denominada `headers` que contiene dos encabezados HTTP alternos y los valores asociados correspondientes. La matriz se pasa como un argumento a `addRequestHeader()`.

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];
my_lv.setRequestHeader(headers);
```

## Véase también

[XML.setRequestHeader\(\)](#)

# LoadVars.contentType

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_lv.contentType
```

## Descripción

Propiedad; tipo MIME que se envía al servidor al llamar a [LoadVars.send\(\)](#) o [LoadVars.sendAndLoad\(\)](#). El valor predeterminado es `application/x-www-form-urlencoded`.

## Véase también

[LoadVars.send\(\)](#), [LoadVars.sendAndLoad\(\)](#)

# LoadVars.getBytesLoaded()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_lv.getBytesLoaded()
```

## Parámetros

Ninguno.

## Valor devuelto

Un número entero.

## Descripción

Método; devuelve el número de bytes descargados por [LoadVars.load\(\)](#) o [LoadVars.sendAndLoad\(\)](#). Este método devuelve `undefined` si no se está realizando ninguna operación de carga o si todavía no se ha iniciado dicha operación.

## LoadVars.getBytesTotal()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.getBytesTotal()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el número total de bytes descargados por `LoadVars.load()` o `LoadVars.sendAndLoad()`. Este método devuelve `undefined` si no se está realizando ninguna operación de carga o si todavía no se ha iniciado dicha operación. Este método también devuelve `undefined` si no se puede determinar el número total de bytes; por ejemplo, si la descarga se ha iniciado pero el servidor no ha transmitido ningún encabezado HTTP `content-length`.

## LoadVars.load()

### Disponibilidad

Flash Player 6; comportamiento modificado en Flash Player 7.

### Sintaxis

```
my_lv.load(url)
```

### Parámetros

*url* Dirección URL desde la que se descargan las variables. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

### Valor devuelto

Una cadena.

### Descripción

Método; descarga variables de la URL especificada, analiza los datos de las variables y coloca las variables resultantes en *my\_lv*. Se sobrescriben todas las propiedades de *my\_lv* que tengan los mismos nombres que las variables descargadas. Las propiedades de *my\_lv* que posean nombres distintos a los de las variables descargadas no se eliminan. Se trata de una acción asíncrona.

Los datos descargados deben estar en el tipo de contenido MIME *application/x-www-form-urlencoded*. Es el mismo formato que utiliza la acción `loadVariables()`.

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables desde archivos SWF que también estén en `www.someDomain.com`. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Asimismo, en los archivos publicados para Flash Player 7, la distinción entre mayúsculas y minúsculas (véase [“Distinción entre mayúsculas y minúsculas” en la página 31](#)) se admite para las variables externas cargadas mediante `LoadVars.load()`.

Este método es parecido a `XML.load()`.

## LoadVars.loaded

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.loaded
```

### Descripción

Propiedad; el valor predeterminado es `undefined`. Cuando se inicia una operación `LoadVars.load()` o `LoadVars.sendAndLoad()`, la propiedad `loaded` se establece en `false`; cuando la operación finaliza, la propiedad `loaded` se establece en `true`. Si la operación todavía no ha concluido o se produce un error, la propiedad `loaded` permanece con el valor `false`.

Esta propiedad es parecida a la propiedad `XML.loaded`.

## LoadVars.onData

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.onData = function(src) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*src* Datos sin formato (sin analizar) de una llamada de método `LoadVars.load()` o `LoadVars.sendAndLoad()`.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando los datos se han descargado completamente del servidor, o cuando se produce un error durante el proceso de descarga de datos de un servidor. Este controlador se invoca antes de que se analicen los datos y, por consiguiente, puede utilizarse para llamar a una rutina de análisis personalizada en lugar de llamar a una incorporada en Flash Player. El valor del parámetro *src* que se pasa a la función asignada a `LoadVars.onData` puede ser `undefined` o una cadena que contenga los pares de nombre y valor con formato URL descargados del servidor. Si el valor devuelto es `undefined`, significa que se ha producido un error al descargar datos del servidor.

La implementación predeterminada de `LoadVars.onData` invoca a `LoadVars.onLoad`. Es posible sobrescribir esta implementación predeterminada asignando una función personalizada a `LoadVars.onData`, pero no volverá a llamarse a `LoadVars.onLoad` a menos que la llamada se realice en la implementación de `LoadVars.onData`.

## LoadVars.onLoad

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.onLoad = function(success) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*success* Este parámetro indica si la operación de carga se ha realizado correctamente (`true`) o no (`false`).

### Valor devuelto

Valor booleano.

### Descripción

Controlador de eventos; se invoca cuando finaliza una operación `LoadVars.load()` o `LoadVars.sendAndLoad()`. Si la operación se ha realizado correctamente, *my\_lv* se llena con las variables descargadas por la operación, y dichas variables están disponibles al invocar este controlador de eventos.

El valor predeterminado de este controlador es `undefined`.

Este método es parecido a `XML.onLoad()`.

## LoadVars.send()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.send(url [, target, method])
```

## Parámetros

*url* URL en la que se deben cargar las variables.

*target* Ventana de la trama del navegador en la que se visualizará cualquier respuesta.

*method* Método GET o POST del protocolo HTTP.

## Valor devuelto

Una cadena.

## Descripción

Método; envía las variables del objeto *my\_lv* a la URL especificada. Todas las variables enumerables de *my\_lv* se concatenan en una cadena cuyo formato predeterminado es *application/x-www-form-urlencoded*, y la cadena se envía a la URL mediante el método POST de HTTP. Es el mismo formato que utiliza la acción `loadVariables()`. El tipo de contenido MIME que se envía en los encabezados de solicitud HTTP es el valor de *my\_lv.contentType* o el valor predeterminado *application/x-www-form-urlencoded*. A menos que se especifique el método GET, se utiliza el método POST.

Si se especifica el parámetro *target*, la respuesta del servidor aparece en la ventana del fotograma del navegador denominada destino. Si se omite el parámetro *target*, se descarta la respuesta del servidor.

Este método es parecido a `XML.send()`.

# LoadVars.sendAndLoad()

## Disponibilidad

Flash Player 6; comportamiento modificado en Flash Player 7.

## Sintaxis

```
my_lv.sendAndLoad(url, targetObject[, method])
```

## Parámetros

*url* URL en la que se deben cargar las variables. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*targetObject* Objeto LoadVars que recibe las variables descargadas.

*method* Método GET o POST del protocolo HTTP.

## Valor devuelto

Una cadena.

## Descripción

Método; publica las variables del objeto *my\_lv* en la URL especificada. La respuesta del servidor se descarga, se analiza como datos de variable y las variables resultantes se colocan en el objeto *targetObject*.

Las variables se envían del mismo modo que `LoadVars.send()`. Las variables se descargan en el *targetObject* del mismo modo que `LoadVars.load()`.

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables desde archivos SWF que también estén en `www.someDomain.com`. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Este método es parecido a `XML.sendAndLoad()`.

## LoadVars.toString()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_lv.toString()
```

### Parámetros

Ninguno.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve una cadena que contiene todas las variables enumerables de *my\_lv*, con la codificación de contenido MIME *application/x-www-form-urlencoded*.

### Ejemplo

```
var myVars = new LoadVars();
myVars.name = "Gary";
myVars.age = 26;
trace (myVars.toString());
//Daría como salida
//name=Gary&age=26
```

## Clase LocalConnection

### Disponibilidad

Flash Player 6.

## Descripción

La clase `LocalConnection` permite desarrollar archivos SWF que pueden enviarse instrucciones entre sí sin utilizar `fscommand()` o JavaScript. Los objetos `LocalConnection` sólo pueden comunicarse entre archivos SWF que estén ejecutándose en el mismo equipo cliente, pero pueden estar ejecutándose en dos aplicaciones distintas. Por ejemplo, un archivo SWF que se está ejecutando en un navegador y otro que se está ejecutando en un proyector. Puede utilizar objetos `LocalConnection` para enviar y recibir datos de un mismo archivo SWF, pero no se trata de una implementación estándar; en todos los ejemplos de esta sección se muestra la comunicación entre distintos archivos SWF.

Los principales métodos utilizados para enviar y recibir datos son `LocalConnection.send()` y `LocalConnection.connect()`. En su estado más básico, el código implementará los comandos siguientes; observe que ambos comandos, `LocalConnection.send()` y `LocalConnection.connect()`, especifican el mismo nombre de conexión, `lc_name`:

```
// Código de la película receptora
receiving_lc = new LocalConnection();
receiving_lc.methodToExecute = function(parámetro1, parámetro2)
{
    // Código que debe ejecutarse
}
receiving_lc.connect("lc_name");
// Código de la película emisora
sending_lc = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", dataItem1, dataItem2)
```

La forma más sencilla de utilizar un objeto `LocalConnection` es permitir la comunicación solamente entre los objetos `LocalConnection` situados en el mismo dominio, ya que de este modo se evitan problemas relacionados con la seguridad. No obstante, si debe permitir la comunicación entre dominios, hay varias formas de implementar medidas de seguridad. Para más información, consulte la descripción del parámetro `connectionName` en `LocalConnection.send()`, así como las entradas `LocalConnection.allowDomain` y `LocalConnection.domain()`.

## Resumen de métodos para la clase `LocalConnection`

Método	Descripción
<code>LocalConnection.close()</code>	Cierra (desconecta) el objeto <code>LocalConnection</code> .
<code>LocalConnection.connect()</code>	Prepara el objeto <code>LocalConnection</code> para recibir comandos de un comando <code>LocalConnection.send()</code> .
<code>LocalConnection.domain()</code>	Devuelve una cadena que representa el superdominio de la ubicación del archivo SWF actual.
<code>LocalConnection.send()</code>	Invoca un método en un objeto <code>LocalConnection</code> especificado.



## Resumen de controladores de eventos para la clase LocalConnection

Controlador de eventos	Descripción
<code>LocalConnection.allowDomain</code>	Se invoca cuando el objeto LocalConnection (receptor) actual recibe una solicitud para invocar un método desde un objeto LocalConnection emisor.
<code>LocalConnection.allowInsecureDomain()</code>	Se invoca cuando el objeto LocalConnection (receptor) actual, que se encuentra en un archivo SWF que está albergado en un dominio que utiliza un protocolo seguro (HTTPS), recibe una solicitud para invocar un método desde un objeto LocalConnection emisor que se encuentra en un archivo SWF que está albergado en un protocolo que no es seguro.
<code>LocalConnection.onStatus</code>	Se invoca después de que un objeto LocalConnection emisor intente enviar un comando a un objeto LocalConnection receptor.

## Constructor para la clase LocalConnection

### Disponibilidad

Flash Player 6.

### Sintaxis

```
new LocalConnection()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto LocalConnection.

### Ejemplo

En el ejemplo siguiente se muestra cómo los archivos SWF receptores y emisores crean objetos LocalConnection. Observe que los dos archivos SWF pueden utilizar el mismo nombre o nombres distintos para sus objetos LocalConnection respectivos. En este ejemplo, utilizan el mismo nombre: `my_lc`.

```
// Código del archivo SWF receptor
my_lc = new LocalConnection();
my_lc.someMethod = function() {
    // las sentencias se escriben aquí
}
my_lc.connect("connectionName");

// Código del archivo SWF emisor
my_lc = new LocalConnection();
my_lc.send("connectionName", "someMethod");
```

Véase también

[LocalConnection.connect\(\)](#), [LocalConnection.send\(\)](#)

## LocalConnection.allowDomain

### Disponibilidad

Flash Player 6; comportamiento modificado en Flash Player 7.

### Sintaxis

```
receiving_lc.allowDomain = function([sendingDomain]) {  
    // Estas sentencias devuelven true o false  
}
```

### Parámetros

*sendingDomain*    Parámetro opcional que especifica el dominio del archivo SWF que contiene el objeto LocalConnection emisor.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando *receiving\_lc* recibe una solicitud para invocar un método de un objeto LocalConnection emisor. Flash espera que el código que se implementa en este controlador devuelva el valor booleano *true* o *false*. Si el controlador no devuelve *true*, la solicitud del objeto emisor se pasa por alto y el método no se invoca.

Utilice este comando para permitir de forma explícita que los objetos LocalConnection de los dominios especificados, o de cualquier dominio, ejecuten métodos del objeto LocalConnection receptor. Si no declara el parámetro *sendingDomain*, probablemente deberá aceptar comandos de cualquier dominio, y el código del controlador será simplemente *return true*. Si declara el parámetro *sendingDomain*, deberá comparar el valor del *sendingDomain* con los dominios desde los que desee aceptar comandos. En los ejemplos siguientes se muestran ambas implementaciones.

En los archivos que se ejecuten en Flash Player 6, el parámetro *sendingDomain* contiene el superdominio del que llama. En los archivos que se ejecuten en Flash Player 7 o posterior, el parámetro *sendingDomain* contiene el dominio exacto del que llama. En este último caso, para permitir el acceso a archivos SWF albergados en *www.domain.com* o *store.domain.com*, debe explícitamente permitir el acceso desde esos dos dominios.

```
// Para Flash Player 6  
receiving_lc.allowDomain = function(sendingDomain) {  
    return(sendingDomain=="domain.com");  
}  
// Comandos correspondientes para permitir el acceso de los archivos SWF  
// que se están ejecutando en Flash Player 7 o posterior  
receiving_lc.allowDomain = function(sendingDomain) {  
    return(sendingDomain=="www.domain.com" ||  
           sendingDomain=="store.domain.com");  
}
```

Asimismo, para los archivos que se ejecuten en Flash Player 7 o posterior, no puede utilizar este método para hacer que los archivos SWF albergados mediante un protocolo seguro (HTTPS) puedan permitir el acceso desde archivos SWF albergados en protocolos que no son seguros; en tal caso, debe utilizar el controlador de eventos `LocalConnection.allowInsecureDomain()`.

## Ejemplo

En el ejemplo siguiente se muestra cómo un objeto `LocalConnection` de un archivo SWF receptor puede permitir que archivos SWF de otro dominio invoquen sus métodos. Compare este ejemplo con el de `LocalConnection.connect()`, en el que sólo los archivos SWF del mismo dominio pueden invocar el método `Trace` en el archivo SWF receptor. Para más información sobre el uso del carácter de subrayado (`_`) en el nombre de la conexión, consulte `LocalConnection.send()`.

```
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}

aLocalConnection.allowDomain = function() {
    // Cualquier dominio puede invocar métodos de este objeto LocalConnection
    return true;
}

aLocalConnection.connect("_trace");
```

En el ejemplo siguiente, el archivo SWF receptor acepta comandos únicamente de los archivos SWF que se encuentran en `thisDomain.com` o `thatDomain.com`.

```
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}

aLocalConnection.allowDomain = function(sendingDomain)
{
    return(sendingDomain=="thisDomain.com" || sendingDomain"thatDomain.com");
}

aLocalConnection.connect("_trace");
```

## Véase también

`LocalConnection.connect()`, `LocalConnection.domain()`, `LocalConnection.send()`

# LocalConnection.allowInsecureDomain()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
receiving_lc.allowInsecureDomain = function([sendingDomain]) {
    // Estas sentencias devuelven true o false
}
```

## Parámetros

*sendingDomain* Parámetro opcional que especifica el dominio del archivo SWF que contiene el objeto LocalConnection emisor.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca siempre que *receiving\_lc*, que se encuentra en un archivo SWF albergado en un dominio que utiliza un protocolo seguro (HTTPS), recibe una solicitud para invocar un método desde un objeto LocalConnection emisor, que se encuentra en un archivo SWF albergado en un protocolo que no es seguro. Flash espera que el código que se implementa en este controlador devuelva el valor booleano *true* o *false*. Si el controlador no devuelve *true*, la solicitud del objeto emisor se pasa por alto y el método no se invoca.

De forma predeterminada, a los archivos SWF albergados mediante el protocolo HTTPS sólo puede acceder otro archivo SWF albergado mediante el protocolo HTTPS. Esta implementación mantiene la integridad que se proporciona mediante el protocolo HTTPS.

No se recomienda el uso de este método para suplantar el comportamiento predeterminado, ya que supone un riesgo para la seguridad de HTTPS. Sin embargo, es posible que deba hacerlo en caso de que, por ejemplo, necesite permitir el acceso a archivos HTTPS publicados para Flash Player 7 o posterior desde archivos HTTP publicados para Flash Player 6.

Un archivo SWF publicado para Flash Player 6 puede utilizar el controlador de eventos `LocalConnection.allowDomain` para permitir el acceso de HTTP a HTTPS. Sin embargo, debido a que la seguridad se implementa de forma diferente en Flash Player 7, debe utilizar el método `LocalConnection.allowInsecureDomain()` para permitir ese acceso en los archivos SWF publicados para Flash Player 7 o posterior.

## Véase también

`LocalConnection.allowDomain`, `LocalConnection.connect()`

# LocalConnection.close()

## Disponibilidad

Flash Player 6.

## Sintaxis

*receiving\_lc.close*

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Método; cierra (desconecta) un objeto `LocalConnection`. Emita este comando cuando desee que el objeto ya no acepte más comandos, por ejemplo, cuando desee emitir un comando `LocalConnection.connect()` con el mismo parámetro *connectionName* en otro archivo SWF.

## Véase también

`LocalConnection.connect()`

# LocalConnection.connect()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
receiving_lc.connect(connectionName)
```

## Parámetros

*connectionName* Cadena que corresponde al nombre de conexión especificado en el comando `LocalConnection.send()` que desea comunicarse con *receiving\_lc*.

## Valor devuelto

El valor booleano `true` si ningún otro proceso que se ejecute en el mismo equipo cliente ha emitido este comando utilizando el mismo valor para el parámetro *connectionName*; de lo contrario, devuelve `false`.

## Descripción

Método; prepara un objeto `LocalConnection` para que reciba comandos de un comando `LocalConnection.send()` (denominado el “objeto `LocalConnection` emisor”). El objeto que se utiliza con este comando se denomina el “objeto `LocalConnection` receptor”. Los objetos receptores y emisores deben ejecutarse en el mismo equipo cliente.

Debe definir los métodos asociados a *receiving\_lc* antes de llamar a este método, como se muestra en todos los ejemplos de esta sección.

De forma predeterminada, Flash Player resuelve *connectionName* con el valor “*superdomain:connectionName*”, siendo *superdomain* el superdominio del archivo SWF que contiene el comando `LocalConnection.connect()`. Por ejemplo, si el archivo SWF que contiene el objeto `LocalConnection` receptor se encuentra en `www.someDomain.com`, *connectionName* se resuelve en “`someDomain.com:connectionName`”. Si un archivo SWF se encuentra en el equipo cliente, el valor asignado a *superdomain* es “`localhost`”.

De forma predeterminada, Flash Player también deja que el objeto `LocalConnection` receptor acepte comandos solamente de los objetos `LocalConnection` emisores cuyo nombre de conexión también se resuelva con un valor “*superdomain:connectionName*”. De este modo, Flash facilita la comunicación entre los archivos SWF que se encuentran en el mismo dominio.

Si implementa la comunicación solamente entre archivos SWF del mismo dominio, especifique una cadena para *connectionName* que no empiece con el carácter de subrayado (`_`) y que no especifique un nombre de dominio (por ejemplo, “`myDomain:connectionName`”). Utilice la misma cadena en el comando `LocalConnection.connect(connectionName)`.

Si implementa la comunicación entre archivos SWF de distintos dominios, consulte la descripción de *connectionName* en `LocalConnection.send()`, así como las entradas `LocalConnection.allowDomain` y `LocalConnection.domain()`.

## Ejemplo

En el ejemplo siguiente se muestra cómo un archivo SWF de un dominio determinado puede invocar un método denominado `Trace` de un archivo SWF receptor del mismo dominio. El archivo SWF receptor funciona como una ventana de rastreo para los archivos SWF emisores; contiene dos métodos a los que los otros archivos SWF pueden llamar: `Trace` y `Clear`. Los botones que se presionan en los archivos SWF emisores llaman a estos métodos con los parámetros especificados.

```
// Archivo SWF receptor
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}
aLocalConnection.Clear = function()
{
    aTextField = "";
}
aLocalConnection.connect("trace");
stop();
```

El archivo SWF 1 contiene el código siguiente asociado a un botón denominado `Presionar`. Al presionar este botón, aparece la frase “Se ha presionado el botón” en el archivo SWF receptor.

```
on (press)
{
    var lc = new LocalConnection();
    lc.send("trace", "Trace", "Se ha presionado el botón.");
    delete lc;
}
```

El archivo SWF 2 contiene un cuadro de introducción de texto con el nombre de variable `myText`, y el código siguiente asociado a un botón denominado `Copiar`. Al escribir texto y presionar el botón, el texto escrito aparece en el archivo SWF receptor.

```
on (press)
{
    _parent.lc.send("trace", "Trace", _parent.myText);
    _parent.myText = "";
}
```

El archivo SWF 3 contiene el código siguiente asociado a un botón denominado `Borrar`. Al presionar el botón, se borra el contenido de la ventana de rastreo del archivo SWF receptor.

```
on (press)
{
    var lc = new LocalConnection();
    lc.send("trace", "Clear");
    delete lc;
}
```

## Véase también

`LocalConnection.send()`

# LocalConnection.domain()

## Disponibilidad

Flash Player 6; comportamiento modificado en Flash Player 7.

## Sintaxis

```
my_lc.domain()
```

## Parámetros

Ninguno.

## Valor devuelto

Cadena que representa el dominio de la ubicación del archivo SWF actual; para más información, consulte la descripción que aparece a continuación.

## Descripción

Método; devuelve una cadena que representa el dominio de la ubicación del archivo SWF actual.

En los archivos SWF publicados para Flash Player 6, la cadena devuelta es el superdominio del archivo SWF actual. Por ejemplo, si el archivo SWF actual se encuentra en `www.macromedia.com`, este comando devuelve `"macromedia.com"`.

En los archivos SWF publicados para Flash Player 7, la cadena que se devuelve es el dominio exacto del archivo SWF actual. Por ejemplo, si el archivo SWF actual se encuentra en `www.macromedia.com`, este comando devuelve `"www.macromedia.com"`.

Si el archivo SWF actual es un archivo local que reside en el equipo cliente, este comando devuelve `"localhost"`.

La forma más común de utilizar este comando es incluir el nombre del dominio del objeto LocalConnection emisor como parámetro para el método que se desea invocar en el objeto LocalConnection receptor, o junto con `LocalConnection.allowDomain` para aceptar comandos del dominio especificado. Si activa la comunicación solamente entre los objetos LocalConnection que se encuentran en el mismo dominio, probablemente no será necesario que utilice este comando.

## Ejemplo

En el ejemplo siguiente, un archivo SWF receptor acepta comandos solamente de los archivos SWF que se encuentran en el mismo dominio o en `macromedia.com`.

```
my_lc = new LocalConnection();
my_lc.allowDomain = function(sendingDomain)
{
    return (sendingDomain==this.domain() || sendingDomain=="macromedia.com");
}
```

En el ejemplo siguiente, un archivo SWF emisor situado en `yourdomain.com` invoca un método de un archivo SWF receptor situado en `mydomain.com`. El archivo SWF emisor incluye su nombre de dominio como parámetro para el método que invoca, de modo que el archivo SWF receptor pueda devolver un valor de respuesta al objeto LocalConnection del dominio correcto. El archivo SWF emisor también especifica que sólo aceptará comandos de los archivos SWF de `mydomain.com`.

Los números de línea se indican a modo de referencia. La secuencia de eventos es la siguiente:

- El archivo SWF receptor se prepara para recibir comandos en una conexión denominada "sum" (línea 11). Flash Player resuelve el nombre de esta conexión con "mydomain.com:sum" (véase [LocalConnection.connect\(\)](#)).
- El archivo emisor se prepara para recibir una respuesta del objeto LocalConnection denominado "result" (línea 58). También especifica que sólo aceptará comandos de los archivos SWF que se encuentren en mydomain.com (líneas 51 a 53).
- El archivo SWF emisor invoca el método aSum de una conexión denominada "mydomain.com:sum" (línea 59) y pasa los parámetros siguientes: su dominio (lc.domain()), el nombre de la conexión que debe recibir la respuesta ("result") y los valores que debe utilizar aSum (123 y 456).
- El método aSum (línea 6) se invoca con los valores siguientes: sender = "mydomain.com:result", replyMethod = "aResult", n1 = 123 y n2 = 456. Por consiguiente, ejecuta la línea de código siguiente:  
`this.send("mydomain.com:result", "aResult", (123 + 456));`
- El método aResult (línea 54) muestra el valor devuelto por aSum (579).

```
// El archivo SWF receptor de http://www.mydomain.com/folder/movie.swf
// contiene el código siguiente
```

```
1  var aLocalConnection = new LocalConnection();
2  aLocalConnection.allowDomain = function()
3  {
4      // Permitir conexiones de cualquier dominio
5      return true;
6  }
6  aLocalConnection.aSum = function(sender, replyMethod, n1, n2)
7  {
8      this.send(sender, replyMethod, (n1 + n2));
9  }
10
11  aLocalConnection.connect("sum");
```

```
// El archivo SWF emisor de http://www.yourdomain.com/folder/movie.swf
// contiene el código siguiente
```

```
50  var lc = new LocalConnection();
51  lc.allowDomain = function(aDomain) {
52      // Permitir conexiones únicamente de mydomain.com
53      return (aDomain == "mydomain.com");
54  }
54  lc.aResult = function(aParam) {
55      trace("La suma es " + aParam);
56  }
57
58  lc.connect("result");
59  lc.send("mydomain.com:sum", "aSum", lc.domain() + ':' + "result",
59      "aResult", 123, 456);
```

## Véase también

[LocalConnection.allowDomain](#)



# LocalConnection.onStatus

## Disponibilidad

Flash Player 6.

## Sintaxis

```
sending_lc.onStatus = function(infoObject) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*infoObject* Parámetro definido de acuerdo con el mensaje de estado. Para más información sobre este parámetro, consulte la descripción que aparece a continuación.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca después de que un objeto LocalConnection intente enviar un comando a un objeto LocalConnection receptor. Si desea responder a este controlador de eventos, debe crear una función para procesar el objeto de información enviado por el objeto LocalConnection.

Si el objeto de información devuelto por este controlador de eventos contiene el valor `level` para "Status", indicará que Flash ha enviado correctamente el comando al objeto LocalConnection receptor. No significa que Flash haya invocado correctamente el método especificado del objeto LocalConnection receptor, sino simplemente que Flash ha podido enviar el comando. El método, por ejemplo, no se invoca si el objeto LocalConnection receptor no permite conexiones del dominio emisor o si el método no existe. La única forma de garantizar que el método se ha invocado es hacer que el objeto receptor envíe una respuesta al objeto emisor.

Si el objeto de información devuelto por este controlador de eventos contiene el valor `level` para "Error", indicará que Flash no ha podido enviar el comando a un objeto LocalConnection receptor, muy probablemente porque no existe ningún objeto LocalConnection receptor conectado cuyo nombre corresponda al nombre especificado en el comando `sending_lc.send()` que ha invocado este controlador.

Además de este controlador `onStatus`, Flash también proporciona una función de superclase denominada `System.onStatus`. Si se invoca `onStatus` para un objeto determinado y no existe ninguna función asignada para responderle, Flash procesará una función asignada a `System.onStatus` si existe.

En la mayoría de los casos, este controlador sólo se implementará para responder a condiciones de error, como se muestra en el ejemplo siguiente.

## Ejemplo

En el ejemplo siguiente se muestra información sobre una conexión que ha fallado en el panel Salida:

```
sending_lc = new LocalConnection();
sending_lc.onStatus = function(objetoInfo)
{
    if (infoObject.level == "Error")
    {
        trace("Error de conexión.");
    }
}
sending_lc.send("receiving_lc", "methodName");
```

## Véase también

[LocalConnection.send\(\)](#), [System.onStatus](#)

# LocalConnection.send()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
sending_lc.send (connectionName, method [, p1,...,pN])
```

## Parámetros

*connectionName* Cadena que corresponde al nombre de conexión especificado en el comando [LocalConnection.connect\(\)](#) que desea establecer comunicación con *sending\_lc*.

*method* Cadena que especifica el nombre del método que debe invocarse en el objeto [LocalConnection](#) receptor. Los nombres de método siguientes harán que el comando no pueda ejecutarse: `send`, `connect`, `close`, `domain`, `onStatus` y `allowDomain`.

*p1,...,pN* Parámetros opcionales que deben pasarse al método especificado.

## Valor devuelto

El valor booleano `true` si Flash puede llevar a cabo la solicitud; de lo contrario, devuelve el valor `false`.

**Nota:** un valor de retorno `true` no indica necesariamente que Flash se haya conectado correctamente a un objeto [LocalConnection](#) receptor; sólo indica que la sintaxis del comando es correcta. Para determinar si la conexión ha sido correcta, consulte [LocalConnection.onStatus](#).

## Descripción

Método; invoca el método denominado *method* en una conexión abierta con el comando [LocalConnection.connect\(connectionName\)](#) (denominado “objeto [LocalConnection](#) receptor”). El objeto utilizado con este comando se denomina “objeto [LocalConnection](#) emisor”. Los archivos SWF que contienen los objetos emisor y receptor deben ejecutarse en el mismo equipo cliente.

Hay un límite en la cantidad de datos que pueden pasarse como parámetros a este comando. Si el comando devuelve `false` pero la sintaxis es correcta, intente dividir las solicitudes de [LocalConnection.send\(\)](#) en varios comandos.

Como se ha explicado en la entrada [LocalConnection.connect\(\)](#), de forma predeterminada, Flash añade el superdominio actual a `connectionName`. Si implementa la comunicación entre distintos dominios, deberá definir `connectionName` tanto en el objeto LocalConnection emisor como en el receptor, de modo que Flash no añada el superdominio actual a `connectionName`. Puede hacerlo de dos formas:

- Utilice un carácter de subrayado (`_`) al principio de `connectionName` en los objetos LocalConnection emisor y receptor. En el archivo SWF que contiene el objeto receptor, utilice [LocalConnection.allowDomain](#) para especificar que se aceptarán conexiones de cualquier dominio. Esta implementación permite almacenar los archivos SWF emisor y receptor en cualquier dominio.
- Incluya el superdominio de `connectionName` en el objeto LocalConnection emisor; por ejemplo, `myDomain.com:myConnectionName`. En el objeto receptor, utilice [LocalConnection.allowDomain](#) para especificar que se aceptarán conexiones del superdominio especificado (en este caso, `myDomain.com`), o que se aceptarán conexiones de cualquier dominio.

**Nota:** no puede especificar un superdominio en `connectionName` del objeto LocalConnection receptor, sólo puede especificarlo en el objeto LocalConnection emisor.

### Ejemplo

Para obtener un ejemplo de comunicación entre objetos LocalConnection ubicados en un mismo dominio, consulte [LocalConnection.connect\(\)](#). Para obtener un ejemplo de comunicación entre objetos LocalConnection ubicados en cualquier dominio, véase [LocalConnection.allowDomain](#). Para obtener un ejemplo de comunicación entre objetos LocalConnection ubicados en dominios especificados, véase [LocalConnection.allowDomain](#) y [LocalConnection.domain\(\)](#).

### Véase también

[LocalConnection.allowDomain](#), [LocalConnection.connect\(\)](#),  
[LocalConnection.domain\(\)](#), [LocalConnection.onStatus](#)

## lt (menor que; específico para cadenas)

### Disponibilidad

Flash Player 4. Este operador se eliminó en Flash 5 y se sustituyó por el nuevo operador `<` ([menor que](#)).

### Sintaxis

`expression1 lt expression2`

### Parámetros

`expression1`, `expression2`    Números, cadenas o variables.

### Descripción

Operador (de comparación); compara la `expression1` con la `expression2` y devuelve `true` si la `expresión1` es menor que la `expression2`; en caso contrario, devuelve `false`.

### Véase también

`<` ([menor que](#))

# Clase Math

## Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

## Descripción

La clase Math es una clase de nivel superior cuyos métodos y propiedades pueden utilizarse sin necesidad de utilizar un constructor.

Utilice los métodos y propiedades de esta clase para acceder a constantes y funciones matemáticas y manipularlas. Todas las propiedades y métodos de la clase Math son estáticos y deben llamarse utilizando la sintaxis `Math.method(parameter)` o `Math.constant`. En ActionScript, las constantes se definen con la máxima precisión de números con coma flotante IEEE-754 de doble precisión.

Algunos de los métodos de la clase Math utilizan radianes de un ángulo como parámetro. Puede utilizar la ecuación siguiente para calcular valores en radianes, o sencillamente pasar la ecuación (introduciendo un valor para grados) para el parámetro en radianes.

Para calcular un valor en radianes, utilice esta fórmula:

`radián = Math.PI/180 * grado`

A continuación, se muestra un ejemplo para pasar una ecuación como parámetro para calcular el seno de un ángulo de 45 grados:

`Math.SIN(Math.PI/180 * 45)` es lo mismo que `Math.SIN(.7854)`

La clase Math está totalmente admitida en Flash Player 5. En Flash Player 4, puede utilizar los métodos de la clase Math, pero se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas que se utilizan en Flash Player 5.

## Resumen de métodos para la clase Math

Método	Descripción
<code>Math.abs()</code>	Calcula un valor absoluto.
<code>Math.acos()</code>	Calcula un arco coseno.
<code>Math.asin()</code>	Calcula un arco seno.
<code>Math.atan()</code>	Calcula un arco tangente.
<code>Math.atan2()</code>	Calcula un ángulo desde el eje x hasta el punto.
<code>Math.ceil()</code>	Redondea un número al entero más cercano hacia arriba.
<code>Math.cos()</code>	Calcula un coseno.
<code>Math.exp()</code>	Calcula un valor exponencial.
<code>Math.floor()</code>	Redondea un número al entero más cercano hacia abajo.
<code>Math.log()</code>	Calcula un logaritmo natural.
<code>Math.max()</code>	Devuelve el mayor de dos números enteros.

Método	Descripción
<code>Math.min()</code>	Devuelve el menor de dos números enteros.
<code>Math.pow()</code>	Calcula $x$ elevado a la potencia de $y$ .
<code>Math.random()</code>	Devuelve un número pseudoaleatorio ente 0,0 y 1,1.
<code>Math.round()</code>	Redondea al número entero más cercano.
<code>Math.sin()</code>	Calcula un seno.
<code>Math.sqrt()</code>	Calcula una raíz cuadrada.
<code>Math.tan()</code>	Calcula una tangente.

## Resumen de propiedades para la clase Math

Todas las propiedades de la clase Math son constantes.

Propiedad	Descripción
<code>Math.E</code>	La constante de Euler y la base de los logaritmos naturales (aproximadamente 2,718).
<code>Math.LN2</code>	El logaritmo natural de 2 (aproximadamente 0,693).
<code>Math.LOG2E</code>	El logaritmo en base 2 de $e$ (aproximadamente 1,442).
<code>Math.LN2</code>	Logaritmo natural de 10 (aproximadamente 2,302).
<code>Math.LOG10E</code>	Logaritmo en base 10 de $e$ (aproximadamente 0,434).
<code>Math.PI</code>	La relación entre la circunferencia de un círculo y su diámetro (aproximadamente 3,14159).
<code>Math.SQRT1_2</code>	El recíproco de la raíz cuadrada de $1/2$ (aproximadamente 0,707).
<code>Math.SQRT2</code>	La raíz cuadrada de 2 (aproximadamente 1,414).

## Math.abs()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.abs(x)`

### Parámetros

$x$  Un número.

### Valor devuelto

Un número.

### Descripción

Método; calcula y devuelve el valor absoluto del número especificado por el parámetro  $x$ .

## Math.acos()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.acos(x)
```

### Parámetros

*x* Un número de -1,0 a 1,0.

### Valor devuelto

Ninguno.

### Descripción

Método; calcula y devuelve el arco coseno del número especificado en el parámetro *x*, en radianes.

## Math.asin()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.asin(x);
```

### Parámetros

*x* Un número de -1,0 a 1,0.

### Valor devuelto

Un número.

### Descripción

Método; calcula y devuelve el arco seno del número especificado en el parámetro *x*, en radianes.

## Math.atan()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.atan(x)
```

### Parámetros

$x$  Un número.

### Valor devuelto

Un número.

### Descripción

Método; calcula y devuelve el arco tangente del número especificado en el parámetro  $x$ . El valor devuelto está entre  $\pi$  negativo dividido por 2 y  $\pi$  positivo dividido por 2.

## Math.atan2()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.atan2( $y$ ,  $x$ )`

### Parámetros

$x$  Número que especifica la coordenada  $x$  del punto.

$y$  Número que especifica la coordenada  $y$  del punto.

### Valor devuelto

Un número.

### Descripción

Método; calcula y devuelve el arco tangente de  $y/x$  en radianes. El valor devuelto representa el ángulo contrario al ángulo opuesto de un triángulo rectángulo, donde  $x$  es la longitud del cateto adyacente e  $y$  es la longitud de la hipotenusa.

## Math.ceil()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.ceil( $x$ )`

### Parámetros

$x$  Un número o una expresión.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el límite del número o expresión especificados. El límite de un número es el número entero más cercano que es mayor o igual que el número.

## Math.cos()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.cos(x)`

### Parámetros

$x$     Ángulo medido en radianes.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el coseno (un valor de -1,0 a 1,0) del ángulo especificado por el parámetro  $x$ . El ángulo  $x$  debe especificarse en radianes. Utilice la información descrita en la entrada [Clase Math](#) para calcular un radián.

## Math.E

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.E`

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constante; una constante matemática para la base de los logaritmos naturales, expresada como  $e$ . El valor aproximado de  $e$  es 2,71828.



## Math.exp()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.exp(x)
```

### Parámetros

*x* Exponente; un número o una expresión.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el valor de la base del logaritmo natural (*e*) elevado a la potencia del exponente especificado en el parámetro *x*. La constante `Math.E` proporciona el valor de *e*.

## Math.floor()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.floor(x)
```

### Parámetros

*x* Un número o una expresión.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el límite inferior de la expresión o número especificado en el parámetro *x*. Este valor es el número entero más cercano que es menor o igual que la expresión o número especificado.

### Ejemplo

El código siguiente devuelve el valor 12.

```
Math.floor(12.5);
```

## Math.log()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.log(x)
```

### Parámetros

*x* Número o expresión con un valor mayor que 0.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el logaritmo del parámetro *x*.

## Math.LN2

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.LN2
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constante; una constante matemática para el logaritmo natural en base 2, expresada como  $\log_2 2$ , con un valor aproximado de 0,69314718055994528623.

## Math.LN10

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.LN10
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Constante; constante matemática para el logaritmo natural en base 10, expresada como  $\log_e 10$ , con un valor aproximado de 2,3025850929940459011.

## Math.LOG2E

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

`Math.LOG2E`

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Constante; constante matemática para el logaritmo en base 2 de la constante  $e$  (`Math.E`), expresado como  $\log_e 2$ , con un valor aproximado de 1,442695040888963387.

## Math.LOG10E

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

`Math.LOG10E`

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

### Descripción

Constante; una constante matemática para el logaritmo en base 10 de la constante  $e$  (`Math.E`), expresado como  $\log_{10}e$ , con un valor aproximado de 0,43429448190325181667.

## Math.max()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase `Math` se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.max(x , y)
```

### Parámetros

*x*    Un número o una expresión.

*y*    Un número o una expresión.

### Valor devuelto

Un número.

### Descripción

Método; calcula el resultado de *x* e *y* y devuelve el valor mayor.

## Math.min()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase `Math` se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

```
Math.min(x , y)
```

### Parámetros

*x*    Un número o una expresión.

*y*    Un número o una expresión.

### Valor devuelto

Un número.

### Descripción

Método; calcula el resultado de *x* e *y* y devuelve el valor menor.

# Math.PI

## Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

## Sintaxis

`Math.PI`

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Constante; una constante matemática que expresa la relación entre la circunferencia de un círculo y su diámetro, expresada como pi, con el valor 3,14159265358979.

# Math.pow()

## Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

## Sintaxis

`Math.pow(x , y)`

## Parámetros

- x* Número que se va a elevar a una potencia.
- y* Número que especifica una potencia a la que se eleva el parámetro *x*.

## Valor devuelto

Un número.

## Descripción

Método; calcula y devuelve *x* elevado a la potencia de *y*:  $x^y$ .

# Math.random()

## Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

```
Math.random()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Un número.

**Descripción**

Método; devuelve  $n$ , donde  $0 \leq n < 1$ .

**Véase también**

[random](#)

## Math.round()

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

```
Math.round(x)
```

**Parámetros**

$x$  Un número.

**Valor devuelto**

Un número.

**Descripción**

Método; redondea el valor del parámetro  $x$  hacia arriba o hacia abajo al número entero más cercano y devuelve el valor.

## Math.sin()

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

```
Math.sin(x)
```

**Parámetros**

$x$  Ángulo medido en radianes.

**Valor devuelto**

Número; el seno del ángulo especificado (entre -1,0 y 1,0).

**Descripción**

Método; calcula y devuelve el seno del ángulo especificado en radianes. Utilice la información descrita en la entrada [Clase Math](#) para calcular un radián.

## Math.sqrt()

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

```
Math.sqrt(x)
```

**Parámetros**

*x* Número o expresión mayor o igual que 0.

**Valor devuelto**

Un número.

**Descripción**

Método; calcula y devuelve la raíz cuadrada del número especificado.

## Math.SQRT1\_2

**Disponibilidad**

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

**Sintaxis**

```
Math.SQRT1_2
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Constante; constante matemática para la raíz cuadrada de un medio.

## Math.SQRT2

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.SQRT2`

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constante; una constante matemática para la raíz cuadrada de 2, con un valor aproximado de 1,414213562373.

## Math.tan()

### Disponibilidad

Flash Player 5. En Flash Player 4, los métodos y las propiedades de la clase Math se emulan utilizando aproximaciones y es posible que no sean tan precisos como las funciones matemáticas no emuladas admitidas por Flash Player 5.

### Sintaxis

`Math.tan(x)`

### Parámetros

*x*    Ángulo medido en radianes.

### Valor devuelto

Un número.

### Descripción

Método; calcula y devuelve la tangente del ángulo especificado. Para calcular los radianes, utilice la información indicada en la introducción de la [Clase Math](#).

## maxscroll

### Disponibilidad

Flash Player 4. Esta función se ha eliminado y se ha sustituido por la propiedad `TextField.maxscroll`.

### Sintaxis

`variable_name.maxscroll`



### Descripción

Propiedad (sólo lectura); propiedad desfasada que indica el número de la primera línea de texto visible en un campo de texto cuando la última línea del campo también está visible. La propiedad `maxscroll` funciona con la propiedad `scroll` para controlar la forma de mostrar información en un campo de texto. Esta propiedad puede recuperarse, pero no modificarse.

### Véase también

[TextField.maxscroll](#), [TextField.scroll](#)

## mbchr

### Disponibilidad

Flash Player 4. Esta función se ha sustituido por el método [String.fromCharCode\(\)](#).

### Sintaxis

`mbchr(number)`

### Parámetros

*number* Número que se convierte en carácter multibyte.

### Valor devuelto

Una cadena.

### Descripción

Función de cadena; convierte un número de código ASCII en un carácter multibyte.

### Véase también

[String.fromCharCode\(\)](#)

## mblength

### Disponibilidad

Flash Player 4. Esta función se ha sustituido por la [Clase String](#).

### Sintaxis

`mblength(string)`

### Parámetros

*string* Una cadena.

### Valor devuelto

Un número.

### Descripción

Función de cadena; devuelve la longitud de la cadena de caracteres multibyte.

## mbord

### Disponibilidad

Flash Player 4. Esta función se eliminó en Flash 5 y se sustituyó por [String.charCodeAt\(\)](#).

### Sintaxis

```
mbord(character)
```

### Parámetros

*character*    Carácter que se convierte en un número multibyte.

### Valor devuelto

Un número.

### Descripción

Función de cadena; convierte el carácter especificado en un número multibyte.

### Véase también

[String.fromCharCode\(\)](#)

## mbsubstring

### Disponibilidad

Flash Player 4. Esta función se eliminó en Flash 5 y se sustituyó por [String.substr\(\)](#).

### Sintaxis

```
mbsubstring(value, index, count)
```

### Parámetros

*value*    Cadena multibyte de la que se debe extraer una nueva cadena multibyte.

*index*    Número del primer carácter que se va a extraer.

*count*    Número de caracteres que se van a incluir en la cadena extraída, sin incluir el carácter de índice.

### Valor devuelto

Una cadena.

### Descripción

Función de cadena; extrae una nueva cadena de caracteres multibyte de una cadena de caracteres multibyte.

### Véase también

[String.substr\(\)](#)

# Clase Microphone

## Disponibilidad

Flash Player 6.

## Descripción

La clase Microphone permite capturar audio desde un micrófono conectado al equipo que esté ejecutando Flash Player.

La clase Microphone se utiliza principalmente con Flash Communication Server, pero puede utilizarse con limitaciones sin el servidor; por ejemplo, puede utilizarse para transmitir sonido del micrófono a través de los altavoces del sistema local.

Para crear un objeto Microphone o hacer referencia al mismo, utilice el método `Microphone.get()`.

## Resumen de métodos para la clase Microphone

Método	Descripción
<code>Microphone.get()</code>	Devuelve el objeto Microphone predeterminado o especificado, o <code>null</code> si el micrófono no está disponible.
<code>Microphone.setGain()</code>	Especifica cuánto debe aumentar el micrófono el volumen de la señal.
<code>Microphone.setRate()</code>	Especifica la velocidad a la que el micrófono debe capturar sonido, expresada en kHz.
<code>Microphone.setSilenceLevel()</code>	Especifica la cantidad de sonido necesaria para activar el micrófono.
<code>Microphone.setUseEchoSuppression()</code>	Especifica si debe utilizarse la función de supresión del eco del códec de audio.

## Resumen de propiedades para la clase Microphone

Propiedad (sólo lectura)	Descripción
<code>Microphone.activityLevel</code>	Cantidad de sonido que detecta el micrófono.
<code>Microphone.gain</code>	Cantidad en que el micrófono aumenta el volumen de la señal antes de transmitirla.
<code>Microphone.index</code>	Índice del micrófono actual.
<code>Microphone.muted</code>	Valor booleano que especifica si el usuario ha permitido o denegado el acceso al micrófono.
<code>Microphone.name</code>	Nombre del dispositivo de captura de sonido actual que devuelve el hardware de captura de sonido.
<code>Microphone.names</code>	Propiedad de clase: matriz de cadenas que reflejan los nombres de los dispositivos de captura de sonido disponibles, incluidas las tarjetas de sonido y los micrófonos.

Propiedad (sólo lectura)	Descripción
<code>Microphone.rate</code>	Velocidad de captura del sonido, expresada en kHz.
<code>Microphone.silenceLevel()</code>	Cantidad de sonido necesaria para activar el micrófono.
<code>Microphone.silenceTimeout()</code>	Número de milisegundos que transcurren entre el momento en que el micrófono deja de detectar sonido y el momento en que se llama a <code>Microphone.onActivity(false)</code> .
<code>Microphone.useEchoSuppression()</code>	Valor booleano que especifica si se está utilizando la supresión del eco.

## Resumen de controladores de eventos para la clase `Microphone`

Controlador de eventos	Descripción
<code>Microphone.onActivity</code>	Se invoca cuando el micrófono empieza a detectar sonido o cuando deja de detectarlo.
<code>Microphone.onStatus</code>	Se invoca cuando el usuario permite o deniega el acceso al micrófono.

## Constructor para la clase `Microphone`

Véase `Microphone.get()`.

## `Microphone.activityLevel`

### Disponibilidad

Flash Player 6.

### Sintaxis

`activeMicrophone.activityLevel`

### Descripción

Propiedad de sólo lectura; valor numérico que especifica la cantidad de sonido que detecta el micrófono. Los valores válidos oscilan entre 0 (no se detecta sonido) y 100 (se detecta un volumen muy alto). El valor de esta propiedad puede ayudarle a determinar un valor apropiado para pasarlo al método `Microphone.setSilenceLevel()`.

Si el micrófono está disponible pero no se está utilizando porque no se ha llamado a `Microphone.get()`, esta propiedad se establece en -1.

### Ejemplo

En el ejemplo siguiente se establece la variable `level` en el nivel de actividad del micrófono actual, `myMic.activityLevel`.

```
var level = myMic.activityLevel;
```

### Véase también

`Microphone.setGain()`

# Microphone.gain

## Disponibilidad

Flash Player 6.

## Sintaxis

*activeMicrophone.gain*

## Descripción

Propiedad de sólo lectura; cantidad en que el micrófono aumenta el volumen de la señal. Los valores válidos oscilan entre 0 y 100, siendo 50 el valor predeterminado.

## Ejemplo

El ejemplo siguiente se asocia a la punta de una barra deslizante. Cuando se carga este clip, Flash comprueba el valor de `myMic.gain` y proporciona un valor predeterminado si el valor es `undefined`. A continuación, se utiliza la posición `_x` para establecer la ganancia del micrófono en el valor de preferencia del usuario.

```
onClipEvent (load) {  
    if (_root.myMic.gain == undefined) {  
        _root.myMic.setGain = 75;  
    }  
  
    this._x = _root.myMic.gain;  
    _root.txt_micgain = this._x;  
  
    left = this._x;  
    right = left+50;  
    top = this._y;  
    bottom = top;  
}  
  
on(press){  
    startDrag(this, false, left, top, right, bottom);  
    this._xscale = 100;  
    this._yscale = 100;  
}  
  
on (release, releaseOutside) {  
    stopDrag();  
    g = (this._x-50)*2;  
    _root.myMic.setGain(g);  
    _root.txt_micgain = g;  
    this._xscale = 100;  
    this._yscale = 100;  
}
```

## Véase también

[Microphone.setGain\(\)](#)

# Microphone.get()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
Microphone.get([index])
```

**Nota:** la sintaxis correcta es `Microphone.get()`. Para asignar el objeto `Microphone` a una variable, utilice una sintaxis de este tipo: `active_mic = Microphone.get()`.

## Parámetros

*index* Número entero opcional con base cero que especifica qué cámara debe obtenerse, según lo que determina la matriz que `Microphone.names` contiene. Para obtener el micrófono predeterminado (generalmente el que se recomienda para la mayoría de las aplicaciones), omita este parámetro.

## Valor devuelto

- Si el parámetro *index* no se especifica, este método devuelve una referencia al micrófono predeterminado o, en caso de no estar disponible, al primer micrófono que sí esté disponible. Si no hay ningún micrófono disponible o instalado, el método devuelve `null`.
- Si se especifica el parámetro *index*, este método devuelve una referencia al micrófono solicitado, o `null`, si dicho micrófono no está disponible.

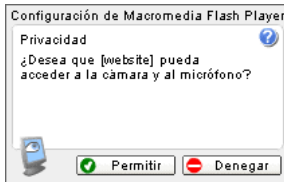
## Descripción

Método; devuelve una referencia a un objeto `Microphone` para capturar audio. Para empezar a capturar audio, debe asociar el objeto `Microphone` al objeto `MovieClip` (véase `MovieClip.attachAudio()`).

A diferencia de los objetos que crea utilizando el constructor `new`, varias llamadas a `Microphone.get()` hacen referencia al mismo micrófono. Por consiguiente, si el script contiene las líneas `mic1 = Microphone.get()` y `mic2 = Microphone.get()`, tanto `mic1` como `mic2` hacen referencia al mismo micrófono (al predeterminado).

Por lo general, no debe pasar ningún valor para el parámetro *index*; simplemente debe utilizar el método `Microphone.get()` para que devuelva una referencia al micrófono predeterminado. Mediante el panel de configuración del micrófono (descrito más adelante en esta sección), el usuario puede especificar el micrófono predeterminado que Flash debe utilizar. Si pasa un valor para el parámetro *index*, es posible que intente hacer referencia a un micrófono distinto del que el usuario desea utilizar. Puede que tenga que utilizar el parámetro *index* en raras ocasiones; por ejemplo, si la aplicación captura audio de dos micrófonos al mismo tiempo.

Cuando un archivo SWF intenta acceder al micrófono que devuelve el método `Microphone.get()` (por ejemplo, cuando se especifica `MovieClip.attachAudio()`), Flash Player muestra un cuadro de diálogo de confidencialidad en el que el usuario puede seleccionar si desea permitir o denegar el acceso al micrófono. Asegúrese de que establece un tamaño mínimo de 215 x 138 píxeles; éste es el tamaño mínimo que Flash necesita para visualizar el cuadro de diálogo.



Cuando el usuario responde a este cuadro de diálogo, el controlador de eventos `Microphone.onStatus` devuelve un objeto de información que indica la respuesta del usuario. Para determinar si el usuario ha denegado o permitido el acceso a la cámara sin procesar este controlador de eventos, véase `Microphone.muted`.

El usuario también puede especificar valores de confidencialidad permanentes para un dominio determinado; para ello, debe hacer clic con el botón derecho del ratón (Windows) o presionar Control y hacer clic (Macintosh) durante la reproducción de un archivo SWF, seleccionar Configuración, abrir el panel de confidencialidad y seleccionar Recordar.



Si `Microphone.get()` devuelve `null`, ello indica que el micrófono está siendo utilizado por otra aplicación o que no se ha instalado ningún micrófono en el sistema. Para determinar si hay micrófonos instalados, utilice `Microphone.names.length`. Para mostrar el panel de configuración del micrófono de Flash Player, en el que el usuario puede seleccionar el micrófono al que `Microphone.get()` debe hacer referencia, utilice `System.showSettings(2)`.



## Ejemplo

En el ejemplo siguiente se permite al usuario especificar el micrófono predeterminado y, a continuación, se captura audio y se reproduce localmente. Para evitar el sonido, es aconsejable utilizar auriculares al probar este código.

```
System.showSettings(2);
myMic = Microphone.get();
_root.attachAudio(myMic);
```

### Véase también

[Microphone.index](#), [Microphone.muted](#), [Microphone.names](#), [Microphone.onStatus](#), [MovieClip.attachAudio\(\)](#)

## Microphone.index

### Disponibilidad

Flash Player 6.

### Sintaxis

*activeMicrophone.index*

### Descripción

Propiedad de sólo lectura; número entero en base cero que especifica el índice del micrófono, como se indica en la matriz devuelta por [Microphone.names](#).

### Véase también

[Microphone.get\(\)](#), [Microphone.names](#)

## Microphone.muted

### Disponibilidad

Flash Player 6.

### Sintaxis

*activeMicrophone.muted*

### Descripción

Propiedad de sólo lectura; valor booleano que especifica si el usuario ha denegado el acceso al micrófono (`true`) o lo ha permitido (`false`). Si se modifica este valor, se invoca a [Microphone.onStatus](#). Para más información, consulte [Microphone.get\(\)](#).

### Ejemplo

En el ejemplo siguiente, cuando el usuario hace clic en el botón, Flash publica y reproduce un flujo en vivo si el micrófono no está silenciado.

```
on (press)
{
    // Si el usuario silencia el micrófono, mostrar un aviso de sin conexión.
    // De lo contrario, publicar y reproducir el flujo en vivo desde el
    micrófono.
    if(myMic.muted) {
        _root.debugWindow+="Micrófono sin conexión." + newline;
    } else {

        // Publicar los datos del micrófono llamando
        // a la función raíz pubLive().
        _root.pubLive();

        // Reproducir lo que se está editando llamando
        // a la función raíz playLive().
        _root.playLive();
    }
}
```



```
}  
}
```

#### Véase también

[Microphone.get\(\)](#), [Microphone.onStatus](#)

## Microphone.name

#### Disponibilidad

Flash Player 6.

#### Sintaxis

*activeMicrophone.name*

#### Descripción

Propiedad de sólo lectura; cadena que especifica el nombre del dispositivo de captura de sonido actual, según lo devuelve el hardware de captura de sonido.

#### Ejemplo

En el ejemplo siguiente se muestra el nombre del micrófono predeterminado en el panel Salida.

```
myMic = Microphone.get();  
trace("El nombre del micrófono es: " + myMic.name);
```

#### Véase también

[Microphone.get\(\)](#), [Microphone.names](#)

## Microphone.names

#### Disponibilidad

Flash Player 6.

#### Sintaxis

*Microphone.names*

**Nota:** la sintaxis correcta es *Microphone.names*. Para asignar el valor devuelto a una variable, utilice una sintaxis del tipo *mic\_array* = *Microphone.names*. Para determinar el nombre del micrófono actual, utilice *activeMicrophone.name*.

#### Descripción

Propiedad de sólo lectura; recupera una matriz de cadenas que reflejan los nombres de todos los dispositivos de captura de sonido disponibles sin mostrar el panel de configuración de confidencialidad de Flash Player. Esta matriz se comporta del mismo modo que cualquier otra matriz de ActionScript, y proporciona implícitamente el índice en base cero de cada uno de los dispositivos de captura de sonido y el número de dispositivos de captura de sonido del sistema (mediante *Microphone.names.length*). Para más información, consulte la entrada [Clase Array](#).

Para llamar a *Microphone.names* se requiere un análisis exhaustivo del hardware y es posible que se necesiten varios segundos para crear la matriz. En la mayoría de los casos, puede utilizar simplemente el micrófono predeterminado.

## Ejemplo

El código siguiente devuelve información sobre la matriz de dispositivos de audio.

```
allMicNames_array = Microphone.names;
_root.debugWindow += "Microphone.names ubicados en estos dispositivos:" +
    newline;
for(i=0; i < allMicNames_array.length; i++){
    debugWindow += "[" + i + "]: " + allMicNames[i] + newline;
}
```

Podría visualizarse, por ejemplo, la siguiente información.

```
Microphone.names localizados en estos dispositivos:
[0]: Crystal SoundFusion(tm)
[1]: Dispositivo de audio USB
```

## Véase también

[Clase Array](#), [Microphone.name](#)

# Microphone.onActivity

## Disponibilidad

Flash Player 6.

## Sintaxis

```
activeMicrophone.onActivity = function(activity) {
    // las sentencias se escriben aquí
}
```

## Parámetros

*activity* Valor booleano que se establece en `true` cuando el micrófono empieza a detectar sonido y, en `false`, cuando detiene la detección.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando el micrófono empieza a detectar sonido o deja de hacerlo. Si desea responder a este controlador de eventos, debe crear una función para procesar su valor de *activity*.

Para especificar la cantidad de sonido necesaria para invocar `Microphone.onActivity(true)` y el tiempo que debe transcurrir sin sonido antes de invocar `Microphone.onActivity(false)`, utilice `Microphone.setSilenceLevel()`.

## Ejemplo

En el ejemplo siguiente se muestra `true` o `false` en el panel Salida cuando el micrófono empieza a detectar sonido o deja de hacerlo.

```
m = Microphone.get();
_root.attachAudio(m);
m.onActivity = function(mode)
{
```

```
        trace(mode);
    };
```

#### Véase también

[Microphone.onActivity](#), [Microphone.setSilenceLevel\(\)](#)

## Microphone.onStatus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
activeMicrophone.onStatus = function(infoObject) {
    // las sentencias se escriben aquí
}
```

### Parámetros

*infoObject*    Parámetro definido de acuerdo con el mensaje de estado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el usuario permite o deniega el acceso al micrófono. Si desea responder a este controlador de eventos, debe crear una función para procesar el objeto de información generado por el micrófono.

Cuando un archivo SWF intenta acceder al micrófono, Flash Player muestra un cuadro de diálogo de confidencialidad en el que el usuario puede seleccionar si desea permitir o denegar el acceso.

- Si el usuario permite el acceso, la propiedad [Microphone.muted](#) se establece en `false` y este controlador de eventos se invoca con un objeto de información cuya propiedad `code` es `"Microphone.Unmuted"` y cuya propiedad `level` es `"Status"`.
- Si el usuario deniega el acceso, la propiedad [Microphone.muted](#) se establece en `true` y este controlador de eventos se invoca con un objeto de información cuya propiedad `code` es `"Microphone.Muted"` y cuya propiedad `level` es `"Status"`.

Para determinar si el usuario ha denegado o permitido el acceso al micrófono sin procesar el controlador de eventos, utilice [Microphone.muted](#).

**Nota:** si el usuario decide permitir o denegar el acceso de forma permanente para todos los archivos SWF de un dominio determinado, este método no se invoca para los archivos SWF de dicho dominio a menos que el usuario cambie más adelante la configuración de la confidencialidad. Para más información, consulte [Microphone.get\(\)](#).

### Ejemplo

Consulte el ejemplo para [Camera.onStatus](#).

### Véase también

[Microphone.get\(\)](#), [Microphone.muted](#)

# Microphone.rate

## Disponibilidad

Flash Player 6.

## Sintaxis

*activeMicrophone.rate*

## Descripción

Propiedad de sólo lectura; velocidad a la que el micrófono captura sonido, expresada en kHz. El valor predeterminado es 8 kHz si el dispositivo de captura de sonido admite este valor. De lo contrario, el valor predeterminado es el siguiente nivel de captura disponible por encima de 8 kHz que admita el dispositivo de captura de sonido, generalmente, 11 kHz.

Para establecer este valor, utilice `Microphone.setRate()`.

## Ejemplo

El ejemplo siguiente guarda la velocidad actual en la variable `original`.

```
original = myMic.rate;
```

## Véase también

`Microphone.setRate()`

# Microphone.setGain()

## Disponibilidad

Flash Player 6.

## Sintaxis

*activeMicrophone.setGain(gain)*

## Parámetros

*gain* Número entero que especifica cuánto debe aumentar el micrófono el volumen de la señal. Los valores válidos oscilan entre 0 y 100, siendo 50 el valor predeterminado; no obstante, el usuario puede cambiar este valor en el panel de configuración del micrófono de Flash Player.

## Valor devuelto

Ninguno.

## Descripción

Método; establece la ganancia del micrófono, es decir, la cantidad por la que el micrófono debería multiplicar la señal antes de transmitirla. El valor 0 indica a Flash que debe multiplicar por 0, en cuyo caso el micrófono no transmite sonido.

Este valor puede considerarse como un control de volumen de un equipo estéreo: 0 indica que no hay volumen y 50 es el volumen normal; los números por debajo de 50 especifican un volumen inferior al normal, mientras que los números por encima de 50 especifican un volumen superior al normal.

### Ejemplo

En el ejemplo siguiente se garantiza que la ganancia del micrófono sea inferior o igual a 55.

```
var myMic = Microphone.get();
if (myMic.gain > 55){
    myMic.setGain(55);
}
```

### Véase también

[Microphone.gain](#), [Microphone.setUseEchoSuppression\(\)](#)

## Microphone.setRate()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
activeMicrophone.setRate(kHz)
```

### Parámetros

*kHz* Velocidad a la que el micrófono debería capturar sonido, expresada en kHz. Los valores aceptables son 5, 8, 11, 22 y 44. El valor predeterminado es 8 kHz si el dispositivo de captura de sonido admite este valor. De lo contrario, el valor predeterminado es el siguiente nivel de captura disponible por encima de 8 kHz que admita el dispositivo de captura de sonido, generalmente, 11 kHz.

### Valor devuelto

Ninguno.

### Descripción

Método; establece la velocidad, expresada en kHz, a la que el micrófono captura sonido.

### Ejemplo

En el ejemplo siguiente se establece la velocidad del micrófono en el valor de preferencia del usuario (que se ha asignado a la variable `userRate`) si es uno de los valores siguientes: 5, 8, 11, 22 ó 44. Si no es ninguno de ellos, el valor se redondea al valor aceptable más cercano admitido por el dispositivo de captura de sonido.

```
myMic.setRate(userRate);
```

### Véase también

[Microphone.rate](#)

## Microphone.setSilenceLevel()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
activeMicrophone.setSilenceLevel(level [, timeout])
```

## Parámetros

*level* Número entero que especifica la cantidad de sonido necesaria para activar el micrófono e invocar `Microphone.onActivity(true)`. Los valores aceptables oscilan entre 0 y 100, siendo 10 el valor predeterminado.

*timeout* Parámetro de número entero opcional que especifica cuántos milisegundos deben transcurrir sin actividad antes de que Flash considere que se ha detenido el sonido e invoque `Microphone.onActivity(false)`. El valor predeterminado es 2000 (2 segundos).

## Valor devuelto

Ninguno.

## Descripción

Método; establece el nivel de entrada mínimo que debe considerarse como sonido y, opcionalmente, la cantidad de tiempo en silencio que indica que ya ha empezado el silencio.

- Para evitar que el micrófono detecte sonido alguno, pase un valor de 100 para *level*; `Microphone.onActivity` nunca se invoca.
- Para determinar la cantidad de sonido que está detectando el micrófono, utilice `Microphone.activityLevel`.

La detección de actividad es la capacidad de detectar cuándo los niveles de audio sugieren que una persona está hablando. Si no hay nadie hablando, puede ahorrarse el ancho de banda porque no es necesario enviar el flujo de audio asociado. Esta información también puede utilizarse para la respuesta visual de modo que los usuarios sepan que ellos (u otros) están en silencio.

Los valores de silencio se corresponden directamente con valores de actividad. El silencio absoluto es una actividad con el valor 0. Un ruido constante y de alto volumen (tan alto como pueda registrarse en función del valor de ganancia actual) corresponde a un valor de actividad 100. Tras ajustar la ganancia correctamente, el valor de actividad es inferior al valor de silencio cuando no se habla; cuando se está hablando, el valor de actividad sobrepasa el valor de silencio.

Este método es parecido en su finalidad a `Camera.setMotionLevel()`; ambos métodos se utilizan para especificar cuándo debe invocarse el controlador de eventos `onActivity`. No obstante, estos métodos tienen un impacto notablemente distinto en los flujos que se publican:

- `Camera.setMotionLevel()` está concebido para detectar movimiento y no tiene ningún efecto sobre el uso del ancho de banda. Aun cuando un flujo de vídeo no detecte movimiento, se enviará vídeo.
- `Microphone.setSilenceLevel()` está concebido para optimizar el ancho de banda. Cuando se considera que el flujo de audio está silenciado, no se envían datos de audio. En su lugar, se envía un único mensaje en el que se indica que se ha iniciado el silencio.

## Ejemplo

En el ejemplo siguiente se modifica el nivel de silencio en función de lo que haya indicado el usuario. El botón tiene el siguiente código asociado:

```
on (press)
{
    this.makeSilenceLevel(this.silenceLevel);
}
```

La función `makeSilenceLevel()` llamada por el botón continúa:

```
function makeSilenceLevel(s)
{
    this.obj.setSilenceLevel(s);
    this.SyncMode();
    this.silenceLevel= s;
}
```

Para más información, consulte el ejemplo para `Camera.setMotionLevel()`.

#### Véase también

`Microphone.activityLevel`, `Microphone.onActivity`, `Microphone.setGain()`,  
`Microphone.silenceLevel()`, `Microphone.silenceTimeout()`

## Microphone.setUseEchoSuppression()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
activeMicrophone.setUseEchoSuppression(suppress)
```

#### Parámetros

*suppress* Valor booleano que indica si debe utilizarse la supresión del eco (`true`) o no (`false`).

#### Valor devuelto

Ninguno.

#### Descripción

Método; especifica si debe utilizarse la función de supresión del eco del códec de audio. El valor predeterminado es `false`, a menos que el usuario haya seleccionado Reducir eco en el panel de configuración del micrófono de Flash Player.

La supresión del eco supone un esfuerzo por reducir los efectos de la respuesta de audio que tiene lugar cuando el micrófono capta el sonido que sale del altavoz en el mismo equipo. Esta función es distinta de la de cancelación del eco, que elimina por completo la respuesta.

Por lo general, es aconsejable utilizar la función de supresión del eco cuando el sonido que se captura se reproduce a través de los altavoces (en lugar de utilizar auriculares) del mismo equipo. Si el archivo SWF permite a los usuarios especificar el dispositivo de salida del sonido, tal vez desee llamar a `Microphone.setUseEchoSuppression(true)` si indican que están utilizando altavoces y también desean utilizar un micrófono.

Los usuarios también pueden ajustar estos valores en el panel de configuración del micrófono de Flash Player.

#### Ejemplo

En el ejemplo siguiente se suprime el eco.

```
my_mic.setUseEchoSuppression(true);
```

#### Véase también

`Microphone.setGain()`, `Microphone.useEchoSuppression()`

## Microphone.silenceLevel()

### Disponibilidad

Flash Player 6.

### Sintaxis

*activeMicrophone.silenceLevel*

### Descripción

Propiedad de sólo lectura; número entero que especifica la cantidad de sonido necesaria para activar el micrófono e invocar `Microphone.onActivity(true)`. El valor predeterminado es 10.

### Ejemplo

Consulte el ejemplo para `Microphone.silenceTimeout()`.

### Véase también

`Microphone.gain`, `Microphone.setSilenceLevel()`

## Microphone.silenceTimeout()

### Disponibilidad

Flash Player 6.

### Sintaxis

*activeMicrophone.silenceTimeout*

### Descripción

Propiedad de sólo lectura; valor numérico que representa el número de milisegundos que transcurren entre el momento en que el micrófono deja de detectar sonido y el momento en que se invoca `Microphone.onActivity(false)`. El valor predeterminado es 2000 (2 segundos).

Para establecer este valor, utilice `Microphone.setSilenceLevel()`.

### Ejemplo

En el ejemplo siguiente se establece el valor del tiempo de espera en el doble de su valor actual.

```
myMic.setSilenceLevel(myMic.silenceLevel, myMic.silenceTimeOut * 2);
```

### Véase también

`Microphone.setSilenceLevel()`

## Microphone.useEchoSuppression()

### Disponibilidad

Flash Player 6.

### Sintaxis

*activeMicrophone.useEchoSuppression*



## Descripción

Propiedad de sólo lectura; valor booleano `true` si se activa la supresión del eco, o `false` en el caso contrario. El valor predeterminado es `false`, a menos que el usuario haya seleccionado Reducir eco en el panel de configuración del micrófono de Flash Player.

## Ejemplo

En el ejemplo siguiente se comprueba la supresión del eco y se activa si está desactivada.

```
_root.myMic.onActivity = function(active) {  
    if (active == true) {  
        if (_root.myMic.useEchoSuppression == false) {  
            _root.myMic.setUseEchoSuppression(true);  
        }  
    }  
}
```

## Véase también

`Microphone.setUseEchoSuppression()`

# MMExecute()

## Disponibilidad

Flash Player 7.

## Sintaxis

`MMExecute("Flash JavaScript API command;")`

## Parámetros

*Flash JavaScript API command*    Cualquier comando que puede utilizar en un archivo Flash JavaScript (JSFL).

## Valor devuelto

El resultado, si lo hubiere, enviado por la sentencia de JavaScript.

## Descripción

Función; permite enviar comandos de la interfaz API JavaScript de Flash desde ActionScript.

La interfaz API JavaScript de Flash (JSAPI) proporcionan varios objetos, métodos y propiedades para duplicar o emular los comandos que un usuario puede introducir en el entorno de edición. Con la interfaz JSAPI, puede escribir scripts que amplían Flash de varias formas: puede añadir comandos a menús, manipular objetos en el escenario, repetir secuencias de comandos, etc.

En general, un usuario ejecuta un script de JSAPI seleccionando Comandos > Ejecutar comando. Sin embargo, puede utilizar esta función en un script ActionScript para llamar a un comando JSAPI directamente. Si utiliza `MMExecute()` en un script en el Fotograma 1 del archivo, el comando se ejecuta cuando se carga el archivo SWF.

Para más información sobre la JSAPI, consulte [www.macromedia.com/go/jsapi\\_info\\_en](http://www.macromedia.com/go/jsapi_info_en).

## Ejemplo

El comando siguiente devuelve una matriz de objetos de la biblioteca:

```
var libe:Array = MMExecute("fl.getDocumentDOM().library.items;");  
trace(libe.length + " elementos de la biblioteca");
```

## Clase Mouse

### Disponibilidad

Flash Player 5.

### Descripción

La clase Mouse es una clase de nivel superior a cuyos métodos y propiedades se puede acceder sin utilizar un constructor. Los métodos de la clase Mouse pueden utilizarse para ocultar y mostrar el puntero del ratón (cursor) en el archivo SWF. El puntero del ratón está visible de forma predeterminada, pero puede ocultarlo e implementar un puntero personalizado creado con un clip de película (véase [“Creación de un puntero de ratón personalizado” en la página 96](#)).

## Resumen de métodos para la clase Mouse

Método	Descripción
<code>Mouse.addListener()</code>	Registra un objeto para que reciba las notificaciones <code>onMouseDown</code> , <code>onMouseMove</code> y <code>onMouseUp</code> .
<code>Mouse.hide()</code>	Oculto el puntero del ratón en el archivo SWF.
<code>Mouse.removeListener()</code>	Elimina un objeto registrado con <code>addListener()</code> .
<code>Mouse.show()</code>	Muestra el puntero del ratón en el archivo SWF.

## Resumen de detectores para la clase Mouse

Método	Descripción
<code>Mouse.onMouseDown</code>	Recibe notificación cuando se presiona el botón del ratón.
<code>Mouse.onMouseMove</code>	Recibe notificación cuando se mueve el botón del ratón.
<code>Mouse.onMouseUp</code>	Recibe notificación cuando se suelta el botón del ratón.
<code>Mouse.onMouseWheel</code>	Recibe notificación cuando el usuario hace girar la rueda del ratón.

## Mouse.addListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Mouse.addListener (newListener)
```

### Parámetros

*newListener* Un objeto.

### Valor devuelto

Ninguno.

### Descripción

Método; registra un objeto para que reciba notificaciones de los detectores `onMouseDown`, `onMouseMove` y `onMouseUp`.

El parámetro *nuevoDetector* debe contener un objeto con métodos definidos para los detectores `onMouseDown`, `onMouseMove` y `onMouseUp`.

Cuando se presiona, mueve o suelta el botón del ratón, independientemente de la selección de entrada, se invoca el método `onMouseDown`, `onMouseMove` o `onMouseUp` de todos los objetos de detección que estén registrados con este método. Varios objetos pueden detectar notificaciones del ratón. Si el detector *newListener* ya está registrado, no se produce ningún cambio.

### Véase también

[Mouse.onMouseDown](#), [Mouse.onMouseMove](#), [Mouse.onMouseUp](#)

## Mouse.hide()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Mouse.hide()
```

### Parámetros

Ninguno.

### Valor devuelto

Un valor booleano: `true` si el puntero está visible, y `false` si no lo está.

### Descripción

Método; oculta el puntero en un archivo SWF. De forma predeterminada, el puntero está visible.

### Ejemplo

El código siguiente, asociado a un clip de película en la línea de tiempo principal, oculta el cursor estándar y establece las posiciones de *x* e *y* de la instancia del clip de película `customPointer_mc` en las posiciones de ratón *x* e *y* de la línea de tiempo principal.

```
onClipEvent(enterFrame) {  
    Mouse.hide();  
    customPointer_mc._x = _root._xmouse;  
    customPointer_mc._y = _root._ymouse;  
}
```

### Véase también

[Mouse.show\(\)](#), [MovieClip.\\_xmouse](#), [MovieClip.\\_ymouse](#)

## Mouse.onMouseDown

### Disponibilidad

Flash Player 6.

### Sintaxis

*someListener*.onMouseDown

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Detector; recibe notificación cuando se presiona el ratón. Para utilizar el detector `onMouseDown`, debe crear un objeto detector. A continuación, puede definir una función para `onMouseDown` y utilizar el método `addListener()` para registrar el detector en el objeto `Mouse`, como en el código siguiente:

```
someListener = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

### Véase también

[Mouse.addListener\(\)](#)

## Mouse.onMouseMove

### Disponibilidad

Flash Player 6.

### Sintaxis

*someListener*.onMouseMove

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

## Descripción

Detector; recibe notificación cuando se mueve el ratón. Para utilizar el detector `onMouseMove`, debe crear un objeto detector. A continuación, puede definir una función para `onMouseMove` y utilizar el método `addListener()` para registrar el detector en el objeto `Mouse`, como en el código siguiente:

```
someListener = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

## Véase también

[Mouse.addListener\(\)](#)

# Mouse.onMouseUp

## Disponibilidad

Flash Player 6.

## Sintaxis

```
someListener.onMouseUp
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Detector; recibe notificación cuando se suelta el ratón. Para utilizar el detector `onMouseUp`, debe crear un objeto detector. A continuación, puede definir una función para `onMouseUp` y utilizar `addListener()` para registrar el detector en el objeto `Mouse`, como en el código siguiente:

```
someListener = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

## Véase también

[Mouse.addListener\(\)](#)

# Mouse.onMouseWheel

## Disponibilidad

Flash Player 7 (sólo Windows).

## Sintaxis

```
someListener.onMouseWheel = function ( [ delta , scrollTarget ] ) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*delta* Número opcional que indica cuántas líneas se deberían desplazar por cada vuelta que el usuario hace girar la rueda del ratón. Un valor *delta* positivo indica un desplazamiento hacia arriba; un valor negativo indica un desplazamiento hacia abajo. Los valores típicos son de 1 a 3, mientras que un desplazamiento más rápido puede producir valores mayores.

Si no desea especificar un valor para *delta* pero sí un valor para *scrollTarget*, pase `null` para *delta*.

*scrollTarget* La instancia de clip de película situada más arriba bajo el ratón cuando se ha movido la rueda del ratón.

## Valor devuelto

Ninguno.

## Descripción

Detector; recibe notificación cuando el usuario hace girar la rueda del ratón. Para utilizar el detector `onMouseWheel`, debe crear un objeto detector. A continuación, puede definir una función para `onMouseWheel` y utilizar `addListener()` para registrar el detector en el objeto `Mouse`:

**Nota:** los detectores de eventos de rueda del ratón sólo están disponibles en las versiones de Flash Player para Windows .

## Ejemplo

En el ejemplo siguiente se muestra cómo crear un objeto detector que responda a los eventos de rueda del ratón. En este ejemplo, la coordenada *x* de un objeto de clip de película denominado `clip_mc` (no se muestra) cambia cada vez que el usuario hace girar la rueda del ratón.

```
mouseListener = new Object();  
mouseListener.onMouseWheel = function(delta) {  
    clip_mc._x += delta;  
}  
Mouse.addListener(mouseListener);
```

## Véase también

`Mouse.addListener()`, `TextField.mouseWheelEnabled`

## Mouse.removeListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Mouse.removeListener ( listener )
```

### Parámetros

*listener*    Un objeto.

### Valor devuelto

Si el objeto *listener* se ha eliminado correctamente, el método devuelve el valor `true`; si el *listener* no se ha eliminado correctamente (por ejemplo, si el *listener* no se encontraba en la lista de detectores del objeto `Mouse`), el método devuelve el valor `false`.

### Descripción

Método; elimina un objeto que se ha registrado anteriormente con `addListener()`.

## Mouse.show()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Mouse.show()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; muestra el puntero del ratón en un archivo SWF. De forma predeterminada, el puntero está visible.

### Véase también

[Mouse.show\(\)](#), [MovieClip.\\_xmouse](#), [MovieClip.\\_ymouse](#)

## Clase MovieClip

### Disponibilidad

Flash Player 3.

### Descripción

Los métodos de la clase `MovieClip` proporcionan la misma funcionalidad que las acciones que utilizan clips de película. También existen métodos adicionales que no tienen acciones equivalentes en la caja de herramientas Acciones del panel Acciones.

No es necesario utilizar un método constructor para poder llamar a los métodos de la clase `MovieClip`; en lugar de eso, haga referencia a las instancias de clip de película por nombre, mediante la sintaxis siguiente:

```
my_mc.play();  
my_mc.gotoAndPlay(3);
```

## Resumen de métodos para la clase `MovieClip`

Método	Descripción
<code>MovieClip.attachAudio()</code>	Captura y reproduce audio local desde el hardware de micrófono.
<code>MovieClip.attachMovie()</code>	Asocia un archivo SWF de la biblioteca.
<code>MovieClip.createEmptyMovieClip()</code>	Crea un clip de película vacío.
<code>MovieClip.createTextField()</code>	Crea un campo de texto vacío.
<code>MovieClip.duplicateMovieClip()</code>	Duplica el clip de película especificado.
<code>MovieClip.getBounds()</code>	Devuelve el máximo y el mínimo de las coordenadas x e y de un archivo SWF en un espacio de coordenadas especificado.
<code>MovieClip.getBytesLoaded()</code>	Devuelve el número de bytes cargados para el clip de película especificado.
<code>MovieClip.getBytesTotal()</code>	Devuelve el tamaño del clip de película en bytes.
<code>MovieClip.getDepth()</code>	Devuelve la profundidad de un clip de película.
<code>MovieClip.getInstanceAtDepth()</code>	Especifica si un clip de película ya ha ocupado una profundidad determinada.
<code>MovieClip.getNextHighestDepth()</code>	Especifica un valor de profundidad que se puede pasar a otros métodos para asegurarse de que Flash muestra el clip de película delante de todos los demás objetos del clip de película actual.
<code>MovieClip.getSWFVersion()</code>	Devuelve un número entero que indica la versión de Flash Player para la que se ha publicado el clip de película.
<code>MovieClip.getTextSnapshot()</code>	Devuelve un objeto <code>TextSnapshot</code> que contiene el texto en los campos de texto estático del clip de película especificado.
<code>MovieClip.getURL()</code>	Recupera un documento de una URL.
<code>MovieClip.globalToLocal()</code>	Convierte el objeto point de las coordenadas del escenario a las coordenadas locales del clip de película especificado.
<code>MovieClip.gotoAndPlay()</code>	Envía la cabeza lectora a un fotograma específico en el clip de película y reproduce el archivo SWF.
<code>MovieClip.gotoAndStop()</code>	Envía la cabeza lectora a un fotograma específico en el clip de película y detiene el archivo SWF.
<code>MovieClip.hitTest()</code>	Devuelve <code>true</code> si el recuadro de delimitación del clip de película especificado se cruza con el recuadro de delimitación del clip de película de destino.
<code>MovieClip.loadMovie()</code>	Carga el archivo SWF especificado en el clip de película.



Método	Descripción
<code>MovieClip.loadVariables()</code>	Carga variables de una URL u otra ubicación en el clip de película.
<code>MovieClip.localToGlobal()</code>	Convierte un objeto point de las coordenadas locales del clip de película a las coordenadas del escenario global.
<code>MovieClip.nextFrame()</code>	Envía la cabeza lectora al fotograma siguiente del clip de película.
<code>MovieClip.play()</code>	Reproduce el clip de película especificado.
<code>MovieClip.prevFrame()</code>	Envía la cabeza lectora al fotograma anterior del clip de película.
<code>MovieClip.removeMovieClip()</code>	Elimina el clip de película de la línea de tiempo si se ha creado con <code>duplicateMovieClip()</code> , <code>MovieClip.duplicateMovieClip()</code> o <code>MovieClip.attachMovie()</code> .
<code>MovieClip.setMask()</code>	Especifica un clip de película como una máscara para otro clip de película.
<code>MovieClip.startDrag()</code>	Especifica un clip de película como arrastrable y comienza a arrastrarlo.
<code>MovieClip.stop()</code>	Detiene la reproducción del archivo SWF.
<code>MovieClip.stopDrag()</code>	Detiene el arrastrado de cualquier clip de película que se esté arrastrando.
<code>MovieClip.swapDepths()</code>	Intercambia el nivel de profundidad de dos archivos SWF.
<code>MovieClip.unloadMovie()</code>	Elimina un archivo SWF que se había cargado con <code>loadMovie()</code> .

## Resumen de métodos de dibujo para la clase `MovieClip`

Método	Descripción
<code>MovieClip.beginFill()</code>	Empieza a dibujar un relleno en el escenario.
<code>MovieClip.beginGradientFill()</code>	Empieza a dibujar un relleno en degradado en el escenario.
<code>MovieClip.clear()</code>	Elimina todos los comandos de dibujo asociados con una instancia de clip de película.
<code>MovieClip.curveTo()</code>	Dibuja una curva con el último estilo de línea.
<code>MovieClip.endFill()</code>	Finaliza el relleno especificado mediante <code>beginFill()</code> o <code>beginGradientFill()</code> .
<code>MovieClip.lineStyle()</code>	Define el trazo de las líneas creadas con los métodos <code>lineTo()</code> y <code>curveTo()</code> .
<code>MovieClip.lineTo()</code>	Dibuja una línea con el estilo de línea actual.
<code>MovieClip.moveTo()</code>	Mueve la posición actual del dibujo a las coordenadas especificadas.

## Resumen de propiedades para la clase MovieClip

Propiedad	Descripción
<code>MovieClip._alpha</code>	Valor de transparencia de una instancia de clip de película.
<code>MovieClip._currentframe</code>	Número de fotograma en el que se encuentra la cabeza lectora.
<code>MovieClip._droptarget</code>	Ruta absoluta en notación de sintaxis con barras de la instancia de clip de película en la que se soltó un clip de película que se puede arrastrar.
<code>MovieClip.enabled</code>	Indica si el clip de película de botón está activado.
<code>MovieClip.focusEnabled</code>	Permite que un clip de película puede seleccionarse.
<code>MovieClip._focusrect</code>	Indica si un clip de película seleccionado tiene un rectángulo amarillo a su alrededor.
<code>MovieClip._framesloaded</code>	Número de fotogramas que se han cargado de un archivo SWF de flujo.
<code>MovieClip._height</code>	Altura de una instancia de clip de película en píxeles.
<code>MovieClip.hitArea</code>	Designa otro clip de película que sirve como el área activa para un clip de película de botón.
<code>MovieClip._highquality</code>	Establece la calidad de representación de un archivo SWF.
<code>MovieClip.menu</code>	Asocia un objeto ContextMenu con un clip de película.
<code>MovieClip._name</code>	Nombre de instancia de una instancia de clip de película.
<code>MovieClip._parent</code>	Referencia al clip de película que incluye el clip de película.
<code>MovieClip._rotation</code>	Grado de rotación de una instancia de clip de película.
<code>MovieClip._soundbuftime</code>	Número de segundos que transcurrirán antes de que empiece el sonido.
<code>MovieClip.tabChildren</code>	Indica si los elementos secundarios de un clip de película se incluyen en el orden de tabulación automático.
<code>MovieClip.tabEnabled</code>	Indica si un clip de película se incluye en el orden de tabulación.
<code>MovieClip.tabIndex</code>	Indica el orden de tabulación de un objeto.
<code>MovieClip._target</code>	Ruta de destino de una instancia de clip de película.
<code>MovieClip._totalframes</code>	Número total de fotogramas de una instancia de clip de película.
<code>MovieClip.trackAsMenu</code>	Indica si otros botones pueden recibir eventos al soltar el ratón.
<code>MovieClip._url</code>	La URL del archivo SWF de la que se ha descargado un clip de película.
<code>MovieClip.useHandCursor</code>	Determina si la mano aparece cuando un usuario se sitúa sobre un clip de película de botón.

Propiedad	Descripción
<code>MovieClip._visible</code>	Valor booleano que determina si una instancia de clip de película está oculta o visible.
<code>MovieClip._width</code>	Anchura de una instancia de clip de película en píxeles.
<code>MovieClip._x</code>	Coordenada x de una instancia de clip de película.
<code>MovieClip._xmouse</code>	Coordenada x del puntero del ratón en una instancia de clip de película.
<code>MovieClip._xscale</code>	Valor que especifica el porcentaje de una escala horizontal de un clip de película.
<code>MovieClip._y</code>	Coordenada y de una instancia de clip de película.
<code>MovieClip._ymouse</code>	Coordenada y del puntero del ratón en una instancia de clip de película.
<code>MovieClip._yscale</code>	Valor que especifica el porcentaje de escala vertical de un clip de película.

## Resumen de controladores de eventos para la clase `MovieClip`

Controlador de eventos	Descripción
<code>MovieClip.onData</code>	Se invoca cuando todos los datos se han cargado en un clip de película.
<code>MovieClip.onDragOut</code>	Se invoca mientras el puntero se encuentra fuera del botón; el botón del ratón se presiona dentro y, a continuación, se desplaza fuera del área del botón.
<code>MovieClip.onDragOver</code>	Se invoca mientras el puntero se encuentra sobre el botón; se ha presionado el botón del ratón, se ha desplazado fuera del botón y, a continuación, se ha vuelto a desplazar sobre el botón.
<code>MovieClip.onEnterFrame</code>	Se invoca de forma continua a la velocidad de los fotogramas del archivo SWF. Las acciones asociadas con el evento de clip <code>enterFrame</code> se procesan antes que cualquiera de las acciones de fotogramas asociadas a los fotogramas afectados.
<code>MovieClip.onKeyDown</code>	Se invoca cuando se presiona una tecla. Utilice los métodos <code>Key.getCode()</code> y <code>Key.getAscii()</code> para recuperar información sobre la última tecla que se ha presionado.
<code>MovieClip.onKeyUp</code>	Se invoca cuando se suelta una tecla.
<code>MovieClip.onKillFocus</code>	Se invoca cuando se deja de seleccionar un botón.
<code>MovieClip.onLoad</code>	Se invoca cuando se crea una instancia del clip de película y aparece en la línea de tiempo.
<code>MovieClip.onMouseDown</code>	Se invoca cuando se presiona el botón izquierdo del ratón.
<code>MovieClip.onMouseMove</code>	Se invoca cada vez que se mueve el ratón.
<code>MovieClip.onMouseUp</code>	Se invoca cuando se suelta el botón izquierdo del ratón.

Controlador de eventos	Descripción
<code>MovieClip.onPress</code>	Se invoca cuando se presiona el ratón mientras el puntero está sobre un botón.
<code>MovieClip.onRelease</code>	Se invoca cuando se suelta el ratón mientras el puntero está sobre un botón.
<code>MovieClip.onReleaseOutside</code>	Se invoca cuando se suelta el ratón mientras el puntero está fuera del botón después de presionar el botón mientras el puntero está dentro del botón.
<code>MovieClip.onRollOut</code>	Se invoca cuando el puntero se desplaza fuera del área de un botón.
<code>MovieClip.onRollOver</code>	Se invoca cuando el puntero del ratón se desplaza sobre un botón.
<code>MovieClip.onSetFocus</code>	Se invoca cuando un botón se selecciona en el momento de la entrada y se suelta una tecla.
<code>MovieClip.onUnload</code>	Se invoca en el primer fotograma después de eliminar el clip de película de la línea de tiempo. Las acciones asociadas con el evento de clip de película Unload se procesan antes de que se asocien acciones al fotograma afectado.

## MovieClip.\_alpha

### Disponibilidad

Flash Player 4.

### Sintaxis

`my_mc._alpha`

### Descripción

Propiedad; valor de transparencia alfa del clip de película especificado por `my_mc`. Los valores válidos van de 0 (completamente transparente) a 100 (completamente opaco). El valor predeterminado es 100. Los objetos de clip de película con el valor de `_alpha` establecido en 0 están activos, aun cuando no sean visibles. Por ejemplo, puede seguir haciendo clic en un botón de un clip de película cuya propiedad `_alpha` esté establecida en 0.

### Ejemplo

El código siguiente establece la propiedad `_alpha` de un clip de película llamado `star_mc` en un 30% al hacer clic en el botón:

```
on(release) {
    star_mc._alpha = 30;
}
```

### Véase también

`Button._alpha`, `TextField._alpha`

# MovieClip.attachAudio()

## Disponibilidad

Flash Player 6; en Flash Player 7 se ha añadido la posibilidad de asociar audio desde archivos Flash Video (FLV).

## Sintaxis

```
my_mc.attachAudio(source)
```

## Parámetros

*source* Objeto que contiene el audio que debe reproducirse. Los valores válidos son un objeto Microphone, un objeto NetStream que esté reproduciendo un archivo FLV y `false` (se detiene la reproducción del audio).

## Valor devuelto

Ninguno.

## Descripción

Método; especifica la fuente de audio que debe reproducirse. Para detener la reproducción de la fuente de audio, pase el valor `false` para *source*.

## Ejemplo

El código siguiente adjunta un micrófono a un clip de película.

```
my_mic = Microphone.get();  
this.attachAudio(my_mic);
```

En el ejemplo siguiente se muestra cómo puede utilizar un objeto Sound para controlar el sonido asociado con un archivo FLV.

```
// Clip es el nombre de instancia del clip de película  
// que contiene el objeto de vídeo "my_video".  
_root.Clip.my_video.attachVideo(_root.myNetStream);  
_root.Clip.attachAudio(_root.myNetStream);  
var snd = new Sound("_root.Clip");  
//Para ajustar el audio:  
_root.snd.setVolume(100);
```

## Véase también

[Clase Microphone](#), [NetStream.play\(\)](#), [Clase Sound](#), [Video.attachVideo\(\)](#)

# MovieClip.attachMovie()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_mc.attachMovie(idName, newName, depth [, initObject])
```

## Parámetros

*idNombre* El nombre de vínculo de un símbolo de clip de película de la biblioteca para asociarlo a un clip de película del escenario. Éste es el nombre introducido en el campo Identificador en el cuadro de diálogo Propiedades de vinculación.

*newname* Nombre de instancia exclusivo para el clip de película que se asocia al clip de película.

*depth* Número entero que especifica el nivel de profundidad en el que se encuentra el archivo SWF.

*initObject* (Admitido para Flash Player 6 y versiones posteriores) Objeto que contiene las propiedades con las que debe llenarse el nuevo clip de película asociado. Este parámetro permite que los clips de película creados de forma dinámica puedan recibir parámetros de clip. Si *initObject* no es un objeto, se pasará por alto. Todas las propiedades de *initObject* se copian en la nueva instancia. Las propiedades especificadas con *initObject* están disponibles para la función constructora. Este parámetro es opcional.

## Valor devuelto

Una referencia a la nueva instancia creada.

## Descripción

Método; selecciona un símbolo de la biblioteca y lo asocia al archivo SWF del escenario especificado por *my\_mc*. Utilice `removeMovieClip()` o `unloadMovie()` para eliminar un archivo SWF asociado con `attachMovie()`.

## Ejemplo

En el ejemplo siguiente se asocia el símbolo del identificador de vínculo “circle” a la instancia de clip de película, que se encuentra en el escenario del archivo SWF.

```
on(release) {  
    thing.attachMovie( "circle", "circle1", 2 );  
}
```

## Véase también

`MovieClip.removeMovieClip()`, `MovieClip.unloadMovie()`, `Object.registerClass()`, `removeMovieClip()`

# MovieClip.beginFill()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.beginFill([rgb[, alpha]])
```

## Parámetro

*rgb* Valor de color hexadecimal (por ejemplo, rojo es 0xFF0000, azul es 0x0000FF, etc.). Si este valor no se proporciona o no está definido, no se crea un relleno.

*alpha* Número entero entre 0 y 100 que especifica el valor alfa del relleno. Si no se proporciona este valor, se utiliza el valor 100 (sólido). Si el valor es inferior a 0, Flash utiliza 0; si el valor es superior a 100, Flash utiliza 100.

#### Valor devuelto

Ninguno.

#### Descripción

Método; indica el comienzo de un nuevo trazado de dibujo. Si hay un trazado abierto (es decir, si la posición actual del dibujo no es la misma que la posición anterior especificada en un método `moveTo()` y éste tiene un relleno asociado, el trazado se cierra con una línea y, a continuación, se rellena. Este comportamiento es parecido a lo que ocurre al llamar a `endFill()`. Si no hay ningún relleno asociado al trazado, debe llamarse a `endFill()` para poder aplicar el relleno.

#### Véase también

`MovieClip.beginGradientFill()`, `MovieClip.endFill()`

## MovieClip.beginGradientFill()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_mc.beginGradientFill(fillType, colors, alphas, ratios, matrix)
```

#### Parámetro

*fillType* La cadena "linear" o la cadena "radial".

*colors* Matriz de valores de color hexadecimales RGB que debe utilizarse en el degradado (por ejemplo, rojo es 0xFF0000, azul es 0x0000FF, etc.).

*alphas* Matriz de valores alfa para los colores correspondientes de la matriz *colors*; los valores válidos son de 0 a 100. Si el valor es inferior a 0, Flash utiliza 0; si el valor es superior a 100, Flash utiliza 100.

*ratios* Matriz de relaciones de distribución de colores; los valores válidos son de 0 a 255. Este valor define el porcentaje de la anchura a la que el color se muestrea al cien por cien.

*matrix* Matriz de transformación que es un objeto con uno de los dos siguientes conjuntos de propiedades.

- *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i* pueden utilizarse para describir una matriz de 3 x 3 del modo siguiente:  
a b c  
d e f  
g h i

En el ejemplo siguiente se utiliza el método `beginGradientFill()` con un parámetro *matrix* que es un objeto con estas propiedades.

```
_root.createEmptyMovieClip( "grad", 1 );  
with ( _root.grad )  
{
```

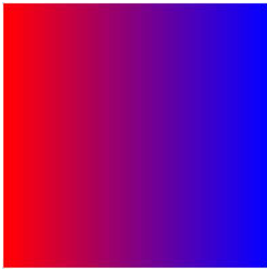
```

colors = [ 0xFF0000, 0x0000FF ];
alphas = [ 100, 100 ];
ratios = [ 0, 0xFF ];
matrix = { a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1 };
beginGradientFill( "linear", colors, alphas, ratios, matrix );
moveto(100.100);
lineto(100.300);
lineto(300.300);
lineto(300.100);
lineto(100.100);
endFill();

}

```

Si no existe una propiedad *matrixType*, todos los demás parámetros son necesarios; la función falla si falta alguno de estos parámetros. Esta matriz cambia la escala, transforma, gira y sesga el degradado de la unidad definido en (-1,-1) y (1,1).



- *matrixType*, *x*, *y*, *w*, *h*, *r*.

Las propiedades indican lo siguiente: *matrixType* es la cadena "box", *x* es la posición horizontal relativa al punto de registro del clip principal para la esquina superior izquierda del degradado, *y* es la posición vertical relativa al punto de registro del clip principal para la esquina superior izquierda del degradado, *w* es la anchura del degradado, *h* es la altura del degradado y *r* es la rotación en radianes del degradado.

En el ejemplo siguiente se utiliza el método `beginGradientFill()` con un parámetro *matrix* que es un objeto con estas propiedades.

```

_root.createEmptyMovieClip( "grad", 1 );
    with ( _root.grad )

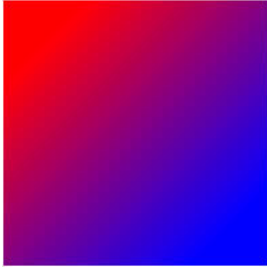
    {

        colors = [ 0xFF0000, 0x0000FF ];
        alphas = [ 100, 100 ];
        ratios = [ 0, 0xFF ];
        matrix = { matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
180)*Math.PI };
        beginGradientFill( "linear", colors, alphas, ratios, matrix );
        moveto(100.100);
        lineto(100.300);
        lineto(300.300);
        lineto(300.100);
        lineto(100.100);
        endFill();
    }

```



Si existe una propiedad *matrixType*, debe ser igual a "box" y todos los demás parámetros son obligatorios. La función falla si no se cumple alguna de estas condiciones.



#### Valor devuelto

Ninguno.

#### Descripción

Método; indica el comienzo de un nuevo trazado de dibujo. Si el primer parámetro tiene el valor *undefined*, o si no se pasa ningún parámetro, el trazado no tiene relleno. Si hay un trazado abierto (es decir, si la posición actual del dibujo no es la misma que la posición anterior especificada en un método *moveTo()*) y éste tiene un relleno asociado, el trazado se cierra con una línea y, a continuación, se rellena. Este comportamiento es parecido al que se produce al llamar a *endFill()*.

Este método falla si se produce alguna de las condiciones siguientes:

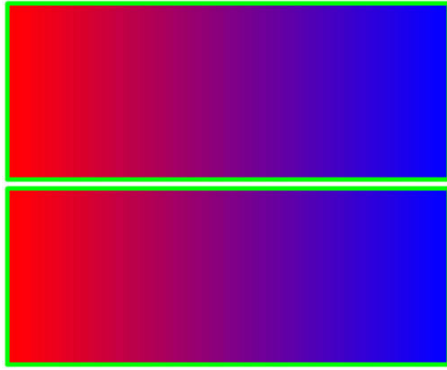
- El número de elementos de los parámetros *colors*, *alphas* y *ratios* no coincide.
- El parámetro *fillType* no es "linear" o "radial".
- Alguno de los campos del objeto para el parámetro *matrix* no es válido o falta.

#### Ejemplo

El código siguiente utiliza ambos métodos para dibujar dos rectángulos apilados con un relleno en degradado rojo-azul y un trazo verde sólido de 5 puntos.

```
_root.createEmptyMovieClip("goober",1);
with ( _root.goober )
{
    colors = [ 0xFF0000, 0x0000FF ];
    alphas = [ 100, 100 ];
    ratios = [ 0, 0xFF ];
    lineStyle( 5, 0x00ff00 );
    matrix = { a:500,b:0,c:0,d:0,e:200,f:0,g:350,h:200,i:1 };
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
    moveto(100.100);
    lineto(100.300);
    lineto(600.300);
    lineto(600.100);
    lineto(100.100);
    endFill();
    matrix = { matrixType:"box", x:100, y:310, w:500, h:200, r:(0/180)*Math.PI };
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
    moveto(100.310);
    lineto(100.510);
}
```

```
lineto(600.510);  
lineto(600.310);  
lineto(100.310);  
endFill();  
}
```



#### Véase también

[MovieClip.beginFill\(\)](#), [MovieClip.endFill\(\)](#), [MovieClip.lineStyle\(\)](#),  
[MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

## MovieClip.clear()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_mc.clear()
```

#### Parámetros

Ninguno.

#### Valor devuelto

Ninguno.

#### Descripción

Método; elimina todos los gráficos creados en tiempo de ejecución utilizando los métodos de dibujo de clip de película, incluidos los estilos de línea especificados con [MovieClip.lineStyle\(\)](#). Las formas y las líneas que se dibujan manualmente durante el proceso de edición con la ayuda de las herramientas de dibujo de Flash no se ven afectadas.

#### Véase también

[MovieClip.lineStyle\(\)](#)

## MovieClip.createEmptyMovieClip()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.createEmptyMovieClip(instanceName, depth)
```

### Parámetros

*instanceName* Cadena que identifica el nombre de instancia del clip de película nuevo.

*depth* Número entero que especifica la profundidad del clip de película nuevo.

### Valor devuelto

Una referencia al clip de película recién creado.

### Descripción

Método; crea un clip de película vacío como un elemento secundario de un clip de película existente. Este método se comporta de forma parecida al método `attachMovie()`, pero no es necesario proporcionar un nombre de vínculo externo para el nuevo clip de película. El punto de registro de un clip de película vacío recién creado se encuentra en la esquina superior izquierda. Este método falla si falta alguno de estos parámetros.

### Véase también

[MovieClip.attachMovie\(\)](#)

## MovieClip.createTextField()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.createTextField(instanceName, depth, x, y, width, height)
```

### Parámetros

*instanceName* Cadena que identifica el nombre de instancia del campo de texto nuevo.

*depth* Número entero positivo que especifica la profundidad del campo de texto nuevo.

*x* Número entero que especifica la coordenada *x* del campo de texto nuevo.

*y* Número entero que especifica la coordenada *y* del campo de texto nuevo.

*width* Número entero positivo que especifica la anchura del campo de texto nuevo.

*height* Número entero positivo que especifica la altura del campo de texto nuevo.

### Valor devuelto

Ninguno.

## Descripción

Método; crea un campo de texto nuevo y vacío que tiene un elemento secundario del clip de película especificado por *my\_mc*. Puede utilizar `createTextField()` para crear campos de texto mientras se reproduce un archivo SWF. El campo de texto se sitúa en (*x*, *y*) con las dimensiones *width* por *height*. Los parámetros *x* e *y* se refieren al clip de película contenedor; estos parámetros corresponden a las propiedades `_x` e `_y` del campo de texto. Los parámetros *width* y *height* corresponden a las propiedades `_width` y `_height` del campo de texto.

Las propiedades predeterminadas de un campo de texto son las siguientes:

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
variable = null
maxChars = null
```

Un campo de texto creado con `createTextField()` recibe el siguiente objeto `TextFormat` predeterminado:

```
font = "Times New Roman"
size = 12
textColor = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
bullet = false
tabStops = [] (empty array)
```

## Ejemplo

En el ejemplo siguiente se crea un campo de texto cuya anchura es 300, la altura es 100, la coordenada *x* es 100, la coordenada *y* es 100, sin borde, texto de color rojo y subrayado.

```
_root.createTextField("mytext",1,100,100,300,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = false;

myformat = new TextFormat();
myformat.color = 0xff0000;
myformat.bullet = false;
myformat.underline = true;

mytext.text = "este es mi primer texto de objeto de campo de prueba";
mytext.setTextFormat(myformat);
```

## Véase también

[Clase TextFormat](#)

# MovieClip.\_currentframe

## Disponibilidad

Flash Player 4.

## Sintaxis

*my\_mc.\_currentframe*

## Descripción

Propiedad (sólo lectura); devuelve el número del fotograma en el que se encuentra la cabeza lectora en la línea de tiempo especificada por *my\_mc*.

## Ejemplo

En el ejemplo siguiente se utiliza la propiedad `_currentframe` para que la cabeza lectora del clip de película `actionClip_mc` avance cinco fotogramas respecto a su ubicación actual.

```
actionClip_mc.gotoAndStop(_currentframe + 5);
```

# MovieClip.curveTo()

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_mc.curveTo(controlX, controlY, anchorX, anchorY)*

## Parámetros

*controlX* Número entero que especifica una posición horizontal relativa al punto de registro del clip de película principal del punto de control.

*controlY* Número entero que especifica una posición vertical relativa al punto de registro del clip de película principal del punto de control.

*anchorX* Número entero que especifica una posición horizontal relativa al punto de registro del clip de película principal del siguiente punto de anclaje.

*anchorY* Número entero que especifica una posición vertical relativa al punto de registro del clip de película principal del siguiente punto de anclaje.

## Valor devuelto

Ninguno.

## Descripción

Métodos; dibuja una curva con el estilo de línea actual desde la posición actual del dibujo hasta (*anchorX*, *anchorY*) y utiliza el punto de control especificado por (*controlX*, *controlY*). La posición actual del dibujo se establece en (*anchorX*, *anchorY*). Si el clip de película que está dibujando contiene un contenido creado con las herramientas de dibujo de Flash, las llamadas a `curveTo()` se dibujan por debajo de este contenido. Si llama a `curveTo()` antes de que se llame a `moveTo()`, la posición actual del dibujo se establece de forma predeterminada en (0, 0). Si falta alguno de estos parámetros, el método falla y la posición del dibujo no se cambia.

## Ejemplo

En el ejemplo siguiente se dibuja un círculo con una línea de color azul sólido muy fina y un relleno de color rojo sólido.

```
_root.createEmptyMovieClip( "circle", 1 );
with ( _root.circle )
{
    lineStyle( 0, 0x0000FF, 100 );
    beginFill( 0xFF0000 );
    moveTo( 500, 500 );
    curveTo( 600, 500, 600, 400 );
    curveTo( 600, 300, 500, 300 );
    curveTo( 400, 300, 400, 400 );
    curveTo( 400, 500, 500, 500 );
    endFill();
}
```

## Véase también

[MovieClip.beginFill\(\)](#), [MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.endFill\(\)](#),  
[MovieClip.lineStyle\(\)](#), [MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

# MovieClip.\_droptarget

## Disponibilidad

Flash Player 4.

## Sintaxis

*my\_mc.\_droptarget*

## Descripción

Propiedad (sólo lectura); devuelve la ruta absoluta en notación de sintaxis con barras de la instancia de clip de película en la que se soltó *my\_mc*. La propiedad `_droptarget` siempre devuelve una ruta que comienza con una barra (/). Para comparar la propiedad `_droptarget` de una instancia con una referencia, utilice la función `eval()` para convertir el valor devuelto en sintaxis con barras a una referencia en sintaxis con puntos.

**Nota:** debe realizar esta conversión si está utilizando ActionScript 2.0, que no admite la sintaxis con barras.

## Ejemplo

En el ejemplo siguiente se calcula el valor de la propiedad `_droptarget` de la instancia de clip de película `garbage` y se utiliza `eval()` para convertir la sintaxis con barras en una referencia de sintaxis con puntos. A continuación, la referencia `garbage` se compara con la referencia a la instancia de clip de película `trash`. Si las dos referencias son equivalentes, la visibilidad de `garbage` se establece en `false`. Si no son equivalentes, la instancia `garbage` se restablece a su posición original.

```
if (eval(garbage._droptarget) == _root.trash) {
    garbage._visible = false;
} else {
    garbage._x = x_pos;
    garbage._y = y_pos;
}
```

Las variables `x_pos` e `y_pos` se establecen en el fotograma 1 del archivo SWF con el script siguiente:

```
x_pos = garbage._x;  
y_pos = garbage._y;
```

**Véase también**

`startDrag()`

## MovieClip.duplicateMovieClip()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_mc.duplicateMovieClip(newname, depth [,initObject])
```

**Parámetros**

*newname* Identificador único para el clip de película duplicado.

*depth* Número único que especifica la profundidad a la que debe colocarse el archivo SWF especificado.

*initObject* (Compatible con Flash Player 6 y versiones posteriores) Objeto que contiene las propiedades con las que se debe rellenar el clip de película duplicado. Este parámetro permite que los clips de película creados de forma dinámica puedan recibir parámetros de clip. Si *initObject* no es un objeto, se pasará por alto. Todas las propiedades de *initObject* se copian en la nueva instancia. Las propiedades especificadas con *initObject* están disponibles para la función constructora. Este parámetro es opcional.

**Valor devuelto**

Una referencia al clip de película duplicado.

**Descripción**

Método; crea una instancia del clip de película especificado mientras se reproduce el archivo SWF. Los clips de película duplicados siempre empiezan a reproducirse en el fotograma 1, sin tener en cuenta en qué fotograma está el clip de película original cuando se llama al método `duplicateMovieClip()`. Las variables del clip de película principal no se copian en el clip de película duplicado. Los clips de película que se han creado con `duplicateMovieClip()` no se duplican si se llama al método `duplicateMovieClip()` en el clip de película principal correspondiente. Si se elimina el clip de película principal, también se elimina el clip de película duplicado.

**Véase también**

`duplicateMovieClip()`, `MovieClip.removeMovieClip()`

# MovieClip.enabled

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.enabled
```

## Descripción

Propiedad; valor booleano que indica si un clip de película de botón está activado. El valor predeterminado de `enabled` es `true`. Si `enabled` se establece en `false`, los métodos callback y los controladores de eventos *on action* del clip de película de botón ya no se invocan, y los fotogramas Sobre, Abajo y Arriba se desactivan. La propiedad `enabled` no afecta a la línea de tiempo del clip de película de botón; si se está reproduciendo un clip de película, éste continúa reproduciéndose. El clip de película sigue recibiendo eventos de clip de película (por ejemplo, `mouseDown`, `mouseUp`, `keyDown` y `keyUp`).

La propiedad `enabled` sólo controla las propiedades de tipo botón de un clip de película de botón. Puede cambiar la propiedad `enabled` en cualquier momento; el clip de película de botón modificado se activa o desactiva inmediatamente. La propiedad `enabled` puede leerse fuera de un objeto prototipo. Si `enabled` se establece en `false`, el objeto no se incluye en el orden de tabulación automático.

# MovieClip.endFill()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.endFill()
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Método; aplica un relleno a las líneas y curvas agregadas desde la última llamada al método `beginFill()` o `beginGradientFill()`. Flash utiliza el relleno que se especificó en la llamada anterior a `beginFill()` o `beginGradientFill()`. Si la posición actual del dibujo no es la misma que la posición anterior especificada en el método `moveTo()` y se define un relleno, el trazado se cierra con una línea y se rellena.



# MovieClip.focusEnabled

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.focusEnabled
```

## Descripción

Propiedad; si el valor es `undefined` o `false`, un clip de película no puede seleccionarse en el momento de la entrada a menos que sea un clip de película de botón. Si el valor de la propiedad `focusEnabled` es `true`, un clip de película puede seleccionarse en el momento de la entrada aunque no sea un clip de película de botón.

# MovieClip.\_focusrect

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc._focusrect
```

## Descripción

Propiedad; un valor booleano que especifica si un clip de película está rodeado por un rectángulo amarillo cuando se selecciona mediante el teclado. Esta propiedad puede suplantar la propiedad `_focusrect` global.

# MovieClip.\_framesloaded

## Disponibilidad

Flash Player 4.

## Sintaxis

```
my_mc._framesloaded
```

## Descripción

Propiedad (sólo lectura); número de fotogramas que se han cargado de un archivo SWF. Esta propiedad es útil para determinar si el contenido de un fotograma específico y todos los fotogramas anteriores a él se han cargado y están disponibles localmente en el navegador. Esta propiedad es útil para controlar la descarga de archivos SWF grandes. Por ejemplo, puede que desee mostrar un mensaje a los usuarios indicando que el archivo SWF se está cargando hasta que se acabe de cargar un fotograma especificado del archivo SWF.

## Ejemplo

En el ejemplo siguiente se utiliza la propiedad `_framesloaded` para iniciar un archivo SWF una vez cargados todos los fotogramas. Si no se han cargado todos los fotogramas, la propiedad `_xscale` de la instancia de clip de película `loader` aumenta proporcionalmente para crear una barra de progreso.

```
if (_framesloaded >= _totalframes) {
    gotoAndPlay ("Scene 1", "start");
} else {
    _root.loader._xscale = (_framesloaded/_totalframes)*100;
}
```

## Véase también

[Clase MovieClipLoader](#)

# MovieClip.getBounds()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_mc.getBounds(targetCoordinateSpace)
```

## Parámetros

*targetCoordinateSpace* Ruta de destino de la línea de tiempo cuyo sistema de coordenadas desea utilizar como punto de referencia.

## Valor devuelto

Un objeto con las propiedades `xMin`, `xMax`, `yMin` e `yMax`.

## Descripción

Método; devuelve las propiedades que son los valores máximo y mínimo de las coordenadas *x* e *y* de la instancia especificada por *my\_mc* para el parámetro *targetCoordinateSpace*.

**Nota:** utilice [MovieClip.localToGlobal\(\)](#) y [MovieClip.globalToLocal\(\)](#) para convertir las coordenadas locales del clip de película en coordenadas de escenario, o las coordenadas de escenario en coordenadas locales.

## Ejemplo

En el ejemplo siguiente, el objeto que devuelve `getBounds()` se asigna al identificador `clipBounds`. Puede acceder a los valores de cada propiedad y utilizarlos en un script. En este script, se coloca otra instancia de clip de película, `clip2`, junto a `clip`.

```
clipBounds = clip.getBounds(_root);
clip2._x = clipBounds.xMax;
```

## Véase también

[MovieClip.globalToLocal\(\)](#), [MovieClip.localToGlobal\(\)](#)

## MovieClip.getBytesLoaded()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.getBytesLoaded()
```

### Parámetros

Ninguno.

### Valor devuelto

Entero que indica el número de bytes cargados.

### Descripción

Método; devuelve el número de bytes que ya se han cargado para el clip de película especificado por *my\_mc*. Puede comparar este valor con el valor devuelto por [MovieClip.getBytesTotal\(\)](#) para determinar el porcentaje de carga del clip de película.

### Véase también

[MovieClip.getBytesTotal\(\)](#)

## MovieClip.getBytesTotal()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.getBytesTotal()
```

### Parámetros

Ninguno.

### Valor devuelto

Número entero que indica el tamaño total, expresado en bytes, de *my\_mc*.

### Descripción

Método; devuelve el tamaño, expresado en bytes, del clip de película especificado por *my\_mc*. Para clips de película que son externos (el archivo SWF raíz o un clip de película que se está cargando en un destino o en un nivel), el valor devuelto es el tamaño del archivo SWF.

### Véase también

[MovieClip.getBytesLoaded\(\)](#)

## MovieClip.getDepth()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.getDepth()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve la profundidad de una instancia de clip de película. Para más información, consulte [“Gestión de las profundidades de los clips de película” en la página 132](#).

### Véase también

[MovieClip.getInstanceAtDepth\(\)](#), [MovieClip.getNextHighestDepth\(\)](#),  
[MovieClip.swapDepths\(\)](#)

## MovieClip.getInstanceAtDepth()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_mc.getInstanceAtDepth(depth)
```

### Parámetros

*depth* Número entero que especifica el nivel de profundidad que debe consultarse.

### Valor devuelto

Una cadena que representa el nombre del clip de película ubicado en la profundidad especificada, o undefined si no hay ningún clip de película en dicha profundidad.

### Descripción

Método; permite determinar si una profundidad determinada está ocupada por un clip de película. Puede utilizar este método antes de utilizar [MovieClip.attachMovie\(\)](#), [MovieClip.duplicateMovieClip\(\)](#) o [MovieClip.createEmptyMovieClip\(\)](#) para determinar si el parámetro de profundidad que desea pasar a cualquiera de estos métodos ya contiene un clip de película. Para más información, consulte [“Gestión de las profundidades de los clips de película” en la página 132](#).

### Véase también

[MovieClip.getDepth\(\)](#), [MovieClip.getNextHighestDepth\(\)](#), [MovieClip.swapDepths\(\)](#)

# MovieClip.getNextHighestDepth()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_mc.getNextHighestDepth()
```

## Parámetros

Ninguno.

## Valor devuelto

Número entero que refleja el siguiente índice de profundidad disponible que se representaría por encima de todos los demás objetos que se encuentran en el mismo nivel y la misma capa en

*my\_mc*.

## Descripción

Método; permite determinar el valor de profundidad que puede pasarse a [MovieClip.attachMovie\(\)](#), [MovieClip.duplicateMovieClip\(\)](#) o [MovieClip.createEmptyMovieClip\(\)](#) para garantizar que Flash represente el clip de película delante de todos los demás objetos del clip de película actual que se encuentren en el mismo nivel y la misma capa. El valor devuelto es 0 o un número mayor (es decir, no se devuelven números negativos).

Para más información, consulte “[Gestión de las profundidades de los clips de película](#)” en [la página 132](#).

## Véase también

[MovieClip.getDepth\(\)](#), [MovieClip.getInstanceAtDepth\(\)](#), [MovieClip.swapDepths\(\)](#)

# MovieClip.getSWFVersion()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_mc.getSWFVersion()
```

## Parámetros

Ninguno.

## Valor devuelto

Número entero que especifica la versión de Flash Player que se deseaba utilizar cuando se publicó el archivo SWF cargado en *my\_mc*.

## Descripción

Método; devuelve un número entero que indica la versión de Flash Player para la que se ha publicado *my\_mc*. Si *my\_mc* es un archivo JPEG, o si se produce un error y Flash no puede determinar la versión SWF de *my\_mc*, se devuelve -1.

# MovieClip.getTextSnapshot()

## Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

## Sintaxis

```
my_mc.getTextSnapshot()
```

## Parámetros

Ninguno.

## Valor devuelto

Objeto TextSnapshot que contiene el texto estático de `my_mc`, o una cadena vacía si `my_mc` no contiene texto estático.

## Descripción

Método; devuelve un objeto TextSnapshot que contiene el texto en todos los campos de texto estático del clip de película especificado; el texto de los clips de película secundarios no está incluido.

Flash concatena el texto y lo coloca en el objeto TextSnapshot en un orden que refleja el orden de índice de tabulación de los campos de texto estático del clip de película. Los campos de texto que no tienen valores de índice de tabulación se colocan en un orden aleatorio en el objeto y preceden a cualquier texto de los campos que cuentan con valores de índice de tabulación. No hay saltos de línea u otro tipo de formato que indique los límites entre campos.

**Nota:** no puede especificar un valor de índice de tabulación para texto estático en Flash. Sin embargo, otros productos pueden hacerlo; por ejemplo, Macromedia FlashPaper.

El contenido del objeto TextSnapshot no es dinámico; es decir, si el clip de película se desplaza a un fotograma diferente, o se altera de alguna forma (por ejemplo, si se añaden o eliminan objetos del clip), el objeto TextSnapshot podría no representar el texto actual del clip de película. Para garantizar que el contenido del objeto es el actual, vuelva a emitir este comando cuanto sea necesario.

## Véase también

[Objeto TextSnapshot](#)

# MovieClip.getURL()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_mc.getURL(URL [,window, variables])
```

## Parámetros

*URL* URL de la que se obtiene el documento.

*window* Parámetro opcional que especifica el nombre, el fotograma o la expresión que indica la ventana o el fotograma HTML en el que se carga el documento. También puede utilizar uno de los nombres de destino reservados que se indican a continuación: `_self` especifica el fotograma actual en la ventana actual, `_blank` especifica una ventana nueva, `_parent` especifica el fotograma principal del fotograma actual, y `_top` especifica el fotograma de nivel superior de la ventana actual.

*variables* Parámetro opcional que especifica un método para enviar variables asociadas con el archivo SWF que se va a cargar. Si no hay variables, no incluya este parámetro; en caso contrario, especifique si deben cargarse con el método GET o POST. GET adjunta las variables al final de la URL y se utiliza para un número reducido de variables. POST envía las variables en un encabezado HTTP aparte y se usa para cadenas largas de variables.

#### Valor devuelto

Ninguno.

#### Descripción

Método; carga un documento de la URL especificada en la ventana especificada. El método `getURL` también puede utilizarse para pasar variables a otra aplicación definida en la URL mediante un método GET o POST.

#### Véase también

[getURL\(\)](#)

## MovieClip.globalToLocal()

#### Disponibilidad

Flash Player 5.

#### Sintaxis

```
my_mc.globalToLocal(point)
```

#### Parámetros

*point* Nombre o identificador de un objeto creado con la [Clase Object](#) genérica. El objeto especifica las coordenadas *x* e *y* como propiedades.

#### Valor devuelto

Ninguno.

#### Descripción

Método; convierte el objeto *point* de las coordenadas (globales) del escenario a las coordenadas (locales) del clip de película.

#### Ejemplo

En el ejemplo siguiente se convierten las coordenadas globales *x* e *y* del objeto *point* a las coordenadas locales del clip de película.

```
onClipEvent(mouseMove){  
    point = new object();  
    point.x = _root._xmouse;  
    point.y = _root._ymouse;  
}
```

```
globalToLocal(point);
trace(_root._xmouse + " " + _root._ymouse);
trace(point.x + " " + point.y);
updateAfterEvent();
}
```

#### Véase también

[MovieClip.getBounds\(\)](#), [MovieClip.localToGlobal\(\)](#)

## MovieClip.gotoAndPlay()

#### Disponibilidad

Flash Player 5.

#### Sintaxis

```
my_mc.gotoAndPlay( frame)
```

#### Parámetros

*frame* Un número que representa el número de fotograma, o una cadena que represente la etiqueta del fotograma, al que se envía la cabeza lectora.

#### Valor devuelto

Ninguno.

#### Descripción

Método; comienza a reproducir el archivo SWF en el fotograma especificado. Si desea especificar una escena y un fotograma, utilice [gotoAndPlay\(\)](#).

## MovieClip.gotoAndStop()

#### Disponibilidad

Flash Player 5.

#### Sintaxis

```
my_mc.gotoAndStop( frame)
```

#### Parámetros

*frame* Número de fotograma al que se envía la cabeza lectora.

#### Valor devuelto

Ninguno.

#### Descripción

Método; coloca la cabeza lectora en el fotograma especificado de este clip de película y la detiene en ese punto.

#### Véase también

[gotoAndStop\(\)](#)



## MovieClip.\_height

### Disponibilidad

Flash Player 4.

### Sintaxis

*my\_mc.\_height*

### Descripción

Propiedad; altura del clip de película, expresada en píxeles.

### Ejemplo

El código de ejemplo siguiente establece la altura y la anchura de un clip de película cuando el usuario hace clic en un botón del ratón.

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

## MovieClip.\_highquality

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_mc.\_highquality*

### Descripción

Propiedad (global); especifica el nivel de suavizado aplicado al archivo SWF actual. Especifique 2 (calidad óptima) para aplicar alta calidad con el suavizado de mapa de bits siempre activado. Especifique 1 (alta calidad) para aplicar suavizado; esto suavizará los mapas de bits si el archivo SWF no contiene animación. Especifique 0 (baja calidad) para evitar el suavizado. Esta propiedad prevalece sobre la propiedad global [\\_highquality](#).

### Ejemplo

```
my_mc._highquality = 2;
```

### Véase también

[\\_quality](#)

## MovieClip.hitArea

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_mc.hitArea*

### Valor devuelto

Una referencia a un clip de película.

### Descripción

Propiedad; designa otro clip de película que sirve como área activa para un clip de película de botón. Si la propiedad `hitArea` no existe o tiene el valor `null` o `undefined`, se utilizará el propio clip de película de botón como área activa. El valor de la propiedad `hitArea` puede ser una referencia a un objeto de clip de película.

Puede cambiar la propiedad `hitArea` en cualquier momento; el clip de película de botón modificado asume de inmediato el comportamiento de la nueva área activa. No es necesario que el clip de película designado como área activa sea visible; se comprueba la presencia de clics en su forma gráfica, aunque no sea visible. La propiedad `hitArea` puede leerse fuera de un objeto prototipo.

## MovieClip.hitTest()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.hitTest(x, y, shapeFlag)
my_mc.hitTest(target)
```

### Parámetros

*x*    Coordenada *x* del área activa del escenario.

*y*    Coordenada *y* del área activa del escenario.

Las coordenadas *x* e *y* se definen en el espacio de coordenadas global.

*target*    Ruta de destino del área activa que puede cruzarse o solaparse con la instancia especificada por *my\_mc*. El parámetro *target* normalmente representa un botón o un campo de introducción de texto.

*shapeFlag*    Valor booleano que especifica si se tiene en cuenta toda la forma de la instancia especificada (*true*) o solamente el recuadro de delimitación (*false*). Este parámetro sólo puede especificarse si el área activa se ha identificado con los parámetros de coordenadas *x* e *y*.

### Valor devuelto

El valor booleano *true* si *my\_mc* se solapa con el área activa especificada; de lo contrario, el valor *false*.

### Descripción

Método; obtiene la instancia especificada por *my\_mc* para ver si se solapa o cruza con el área activa identificada por el parámetro *destino* o los parámetros de coordenadas *x* e *y*.

Sintaxis 1: compara las coordenadas *x* e *y* con la forma o el recuadro de delimitación de la instancia especificada, según el valor de *shapeFlag*. Si *shapeFlag* está establecido en *true*, sólo se tiene en cuenta el área que ocupa actualmente la instancia en el escenario; si *x* e *y* se solapan en algún punto, se devuelve el valor *true*. Esto es muy útil para determinar si el clip de película se encuentra dentro del área activa especificada.

Sintaxis 2: obtiene los recuadros de delimitación de *target* y de la instancia especificada y devuelve el valor `true` si se solapan o se cruzan en algún momento.

### Ejemplo

En el ejemplo siguiente se utiliza `hitTest()` con las propiedades `_xmouse` y `_ymouse` para determinar si el puntero del ratón se encuentra sobre el recuadro de delimitación del destino:

```
if (hitTest( _root._xmouse, _root._ymouse, false));
```

En el ejemplo siguiente se utiliza `hitTest()` para determinar si un clip de película `ball` se solapa o se cruza con el clip de película `square`:

```
if(_root.ball.hitTest(_root.square)){  
    trace("ball se cruza con square");  
}
```

### Véase también

[MovieClip.getBounds\(\)](#), [MovieClip.globalToLocal\(\)](#), [MovieClip.localToGlobal\(\)](#)

## MovieClip.lineStyle()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.lineStyle([thickness[, rgb[, alpha]])
```

### Parámetros

*thickness* Número entero que indica el grosor de la línea en puntos; los valores válidos oscilan entre 0 y 255. Si no se especifica ningún número, o si el parámetro es `undefined`, no se dibuja ninguna línea. Si se especifica un valor menor que 0, Flash utiliza 0. El valor 0 indica un grosor muy fino; el grosor máximo es 255. Si se especifica un valor superior a 255, el intérprete de Flash utiliza 255.

*rgb* Valor de color hexadecimal de la línea (por ejemplo, rojo es `0xFF0000`, azul es `0x0000FF`, etc.). Si no se indica ningún valor, Flash utiliza el valor `0x000000` (negro).

*alpha* Número entero que indica el valor alfa del color de la línea; los valores válidos oscilan entre 0 y 100. Si no se indica ningún valor, Flash utiliza 100 (sólido). Si el valor es inferior a 0, Flash utiliza 0; si el valor es superior a 100, Flash utiliza 100.

### Valor devuelto

Ninguno.

### Descripción

Método; especifica un estilo de línea que Flash utiliza para las siguientes llamadas a los métodos `lineTo()` y `curveTo()` hasta que se llame a `lineStyle()` con parámetros distintos. Puede llamar al método `lineStyle()` mientras dibuja un trazado para especificar diferentes estilos para distintos segmentos de línea de un trazado.

**Nota:** las llamadas a `clear` restablecen el valor de `lineStyle()` en `undefined`.

## Ejemplo

El código siguiente dibuja un triángulo con una línea de color magenta sólido de 5 puntos sin relleno.

```
_root.createEmptyMovieClip ("triangle", 1);
with ( _root.triangle )
{
    lineStyle (5, 0xff00ff, 100);
    moveTo (200, 200);
    lineTo( 300,300 );
    lineTo (100, 300);
    lineTo (200, 200);
}
```

## Véase también

[MovieClip.beginFill\(\)](#), [MovieClip.beginGradientFill\(\)](#), [MovieClip.clear\(\)](#),  
[MovieClip.curveTo\(\)](#), [MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

# MovieClip.lineTo()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.lineTo(x, y)
```

## Parámetros

*x* Número entero que indica la posición horizontal relativa al punto de registro del clip de película principal.

*y* Número entero que indica la posición vertical relativa al punto de registro del clip de película principal.

## Valor devuelto

Ninguno.

## Descripción

Método; dibuja una línea con el estilo de línea actual desde la posición actual del dibujo hasta (*x*, *y*); la posición actual del dibujo se establece en (*x*, *y*). Si el clip de película que está dibujando tiene un contenido creado con las herramientas de dibujo de Flash, las llamadas a `lineTo()` se dibujan debajo del contenido. Si llama al método `lineTo()` antes de llamar al método `moveTo()`, la posición del dibujo se establece de forma predeterminada en (0, 0). Si falta alguno de estos parámetros, el método falla y la posición del dibujo no se cambia.

## Ejemplo

En el ejemplo siguiente se dibuja un triángulo sin líneas y con un relleno azul parcialmente transparente.

```
_root.createEmptyMovieClip ("triangle", 1);
with (_root.triangle){
    beginFill (0x0000FF, 50);
    lineStyle (5, 0xFF00FF, 100);
```

```

        moveTo (200, 200);
        lineTo (300, 300);
        lineTo (100, 300);
        lineTo (200, 200);
        endFill();
    }

```

### Véase también

[MovieClip.beginFill\(\)](#), [MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.endFill\(\)](#), [MovieClip.lineStyle\(\)](#), [MovieClip.moveTo\(\)](#)

## MovieClip.loadMovie()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.loadMovie("url" [,variables])
```

### Parámetros

*url* URL absoluta o relativa del archivo SWF o JPEG que se debe cargar. Una ruta relativa debe ser relativa respecto al archivo SWF del nivel 0. Los URL absolutos deben incluir la referencia al protocolo, como `http://` o `file:///`.

*variables* Parámetro opcional que especifica un método HTTP para enviar o cargar variables. El parámetro debe ser la cadena `GET` o `POST`. Si no hay ninguna variable para enviar, no incluya este parámetro. El método `GET` adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método `POST` envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

### Valor devuelto

Ninguno.

### Descripción

Método; carga archivos SWF o JPEG en un clip de película de Flash Player mientras se reproduce el archivo SWF original.

**Sugerencia:** si desea supervisar el progreso de la operación de descarga de un archivo, utilice [MovieClipLoader.loadClip\(\)](#) en lugar de esta función.

Sin el método `loadMovie()`, Flash Player muestra un solo archivo SWF y después se cierra. El método `loadMovie()` permite ver varios archivos SWF al mismo tiempo y pasar de uno a otro sin cargar otro documento HTML.

Los archivos SWF y las imágenes que se cargan en un clip de película heredan las propiedades de posición, rotación y escala del clip de película. Puede utilizar la ruta de destino del clip de película para acceder al archivo SWF cargado.

Utilice el método `unloadMovie()` para eliminar archivos SWF o imágenes cargadas con el método `loadMovie()`. Utilice el método `loadVariables()` para mantener el archivo SWF activo y actualizar las variables con valores nuevos.

## Véase también

```
loadMovie(), loadMovieNum(), MovieClip.loadVariables(), MovieClip.unloadMovie(),  
unloadMovie(), unloadMovieNum()
```

# MovieClip.loadVariables()

## Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

## Sintaxis

```
my_mc.loadVariables("url", variables)
```

## Parámetros

*url* URL absoluta o relativa del archivo externo que contiene las variables que se van a cargar. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*variables* Parámetro opcional que especifica un método HTTP para enviar variables. El parámetro debe ser la cadena GET o POST. Si no hay ninguna variable para enviar, no incluya este parámetro. El método GET adjunta las variables al final de la URL y se utiliza para un número pequeño de variables. El método POST envía las variables en un encabezado HTTP distinto y se usa para cadenas largas de variables.

## Valor devuelto

Ninguno.

## Descripción

Método; lee los datos de un archivo externo y establece los valores de las variables en *my\_mc*. El archivo externo puede ser un archivo de texto generado por un script CGI, Active Server Page (ASP) o un script PHP y puede contener cualquier número de variables.

Este método también puede utilizarse para actualizar las variables del clip de película activa con nuevos valores.

Este método requiere que el texto de la URL tenga el formato MIME estándar: *application/x-www-form-urlencoded* (formato de script CGI).

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de *www.someDomain.com* puede cargar variables desde un archivo SWF de *store.someDomain.com* porque ambos archivos se encuentran en el mismo superdominio que *someDomain.com*.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de *www.someDomain.com* sólo puede cargar variables desde archivos SWF que también estén en *www.someDomain.com*. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

## Véase también

[loadMovie\(\)](#), [loadVariables\(\)](#), [loadVariablesNum\(\)](#), [MovieClip.unloadMovie\(\)](#)

# MovieClip.localToGlobal()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_mc.localToGlobal(point)
```

## Parámetros

*point* Nombre o identificador de un objeto creado con la [Clase Object](#) que especifica las coordenadas *x* e *y* como propiedades.

## Valor devuelto

Ninguno.

## Descripción

Método; convierte el objeto *point* de las coordenadas (locales) del clip de película a las coordenadas (globales) del escenario.

## Ejemplo

En el ejemplo siguiente se convierten las coordenadas *x* e *y* del objeto *point* de las coordenadas (locales) del clip de película a las coordenadas (globales) del escenario. Las coordenadas locales *x* e *y* se especifican con las propiedades `_xmouse` e `_ymouse` para recuperar las coordenadas *x* e *y* de la posición del puntero del ratón.

```
onClipEvent(mouseMove){
    point = new object();
    point.x = _xmouse;
    point.y = _ymouse;
    _root.out3 = point.x + " === " + point.y;
    _root.out = _root._xmouse + " === " + _root._ymouse;
    localToGlobal(point);
    _root.out2 = point.x + " === " + point.y;
    updateAfterEvent();
}
```

## Véase también

[MovieClip.globalToLocal\(\)](#)

# MovieClip.\_lockroot

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_mc._lockroot
```

## Descripción

Propiedad; especifica a qué hace referencia `_root` cuando se carga un archivo SWF en un clip de película. La propiedad `_lockroot` tiene el valor `undefined` de forma predeterminada. Puede establecer esta propiedad en el archivo SWF que se está cargando o en el controlador que está cargando el clip de película.

Por ejemplo, suponga que tiene un documento denominado `Games.fla` que permite a un usuario seleccionar el juego con el que desea jugar y carga dicho juego (por ejemplo, `Chess.swf`) en el clip de película `game_mc`. Debe asegurarse de que, si se utiliza `_root` en el archivo `Chess.swf`, todavía hace referencia a `_root` en el archivo `Chess.swf` tras cargarlo en `Games.swf`. Si tiene acceso a `Chess.fla` y lo publica en Flash Player 7 o una versión posterior, puede añadirle esta sentencia:

```
this._lockroot = true;
```

Si no tiene acceso a `Chess.fla` (por ejemplo, si está cargando el archivo `Chess.swf` del sitio de otro usuario), puede establecer su propiedad `_lockroot` cuando lo cargue, tal como se muestra a continuación. En este caso, el archivo `Chess.swf` puede publicarse en cualquier versión de Flash Player, siempre y cuando `Games.swf` se haya publicado para Flash Player 7 o una versión posterior.

```
onClipEvent (load)
{
    this._lockroot = true;
}
game_mc.loadMovie ("Chess.swf");
```

Si no ha utilizado la sentencia `this._lockroot = true` en ningún archivo SWF, `_root` en `Chess.swf` hará referencia a `_root` en `Games.swf` después de que `Chess.swf` se cargue en `Games.swf`.

## Véase también

[\\_root](#), [MovieClip.attachMovie\(\)](#), [MovieClip.loadMovie\(\)](#)

# MovieClip.menu

## Disponibilidad

Flash Player 7.

## Utilización

```
my_mc.menu = contextMenu
```

## Parámetros

*contextMenu*    Objeto ContextMenu.

## Descripción

Propiedad; asocia el objeto ContextMenu especificado con el clip de película *my\_mc*. La clase ContextMenu permite modificar el menú contextual que aparece al hacer clic con el botón derecho del ratón (Windows) o al mantener presionada la tecla Control y hacer clic (Macintosh) en Flash Player.



## Ejemplo

En el ejemplo siguiente se asigna el objeto `ContextMenu` denominado `menu_cm` al clip de película `content_mc`. El objeto `ContextMenu` contiene un elemento de menú personalizado etiquetado “Imprimir...” con un controlador callback asociado denominado `doPrint()`.

```
var menu_cm = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Imprimir...", doPrint));
function doPrint(menu, obj) {
    // Aquí aparece el código "Imprimir"
}
content_mc.menu = menu_cm;
```

## Véase también

[Button.menu](#), [Clase ContextMenu](#), [Clase ContextMenuItem](#), [TextField.menu](#)

# MovieClip.moveTo()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.moveTo(x, y)
```

## Parámetros

*x* Número entero que indica la posición horizontal relativa al punto de registro del clip de película principal.

*y* Número entero que indica la posición vertical relativa al punto de registro del clip de película principal.

## Valor devuelto

Ninguno.

## Descripción

Método; mueve la posición actual del dibujo a (*x*, *y*). Si falta alguno de estos parámetros, el método falla y la posición del dibujo no se cambia.

## Ejemplo

En este ejemplo se dibuja un triángulo con líneas de color magenta sólido de 5 puntos sin relleno. La primera línea crea un clip de película vacío con el que se dibuja. En la sentencia `with`, se define un tipo de línea y, a continuación, el método `moveTo()` indica la posición inicial de dibujo.

```
_root.createEmptyMovieClip ("triangle", 1);
with ( _root.triangle )
{
    lineStyle (5, 0xff00ff, 100);
    moveTo (200, 200);
    lineTo( 300,300 );
    lineTo (100, 300);
    lineTo (200, 200);
}
```

### Véase también

[MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.lineStyle\(\)](#), [MovieClip.lineTo\(\)](#)

## MovieClip.\_name

### Disponibilidad

Flash Player 4.

### Sintaxis

*my\_mc*.\_name

### Descripción

Propiedad; nombre de instancia del clip de película especificado por *my\_mc*.

## MovieClip.nextFrame()

### Disponibilidad

Flash Player 5.

### Sintaxis

*my\_mc*.nextFrame()

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; envía la cabeza lectora al fotograma siguiente y la detiene.

### Véase también

[nextFrame\(\)](#)

## MovieClip.onData

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onData = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando un clip de película recibe datos de una llamada `loadVariables()` o `loadMovie()`. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

Este controlador se puede utilizar con clips de película que tengan un símbolo en la biblioteca que esté asociada con una clase. Si desea que se invoque un controlador de eventos cuando un clip de película determinado recibe datos, debe utilizar `onClipEvent(data)` en lugar de este controlador. Este último controlador se invoca cuando cualquier clip de película recibe datos.

## Ejemplo

En el ejemplo siguiente se ilustra el uso correcto de `MovieClip.onData()` y `onClipEvent(data)`.

```
// symbol_mc es un símbolo de clip de película de la biblioteca.
// Está vinculado con la clase MovieClip.
// Para cada instancia de symbol_mc se activa la función siguiente
// cuando recibe datos.
symbol_mc.onData = function() {
    trace("El clip de película ha recibido datos");
}

// dynamic_mc es un clip de película que se carga con MovieClip.loadMovie().
// Este código intenta llamar a una función cuando el clip se carga,
// pero no funcionará, ya que el archivo SWF cargado no es un símbolo
// de la biblioteca asociada con la clase MovieClip.
function output()
{
    trace("No se llamará nunca.");
}
dynamic_mc.onData = output;
dynamic_mc.loadMovie("replacement.swf");

// La función siguiente se invoca para cualquier clip de película que
// reciba datos, tanto si está en la biblioteca como si no.
// Por lo tanto, esta función se invoca cuando se crea una instancia de
// symbol_mc
// y también cuando se carga replacement.swf.
onClipEvent( data ) {
    trace("El clip de película ha recibido datos");
}
```

## Véase también

[onClipEvent\(\)](#)

# MovieClip.onDragOut

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onDragOut = function() {
    // las sentencias se escriben aquí
}
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Controlador de eventos; se invoca cuando se presiona el botón del ratón y el puntero se desliza fuera del objeto. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

**Ejemplo**

En el ejemplo siguiente se define una función para el método `onDragOut` que envía una acción `trace()` al panel Salida.

```
my_mc.onDragOut = function () {  
    trace ("se ha llamado al método onDragOut");  
};
```

**Véase también**

[MovieClip.onDragOver](#)

## MovieClip.onDragOver

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_mc.onDragOver = function() {  
    // las sentencias se escriben aquí  
}
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Controlador de eventos; se invoca cuando se desliza el puntero fuera del clip de película y después se vuelve a situar encima de éste. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

**Ejemplo**

En el ejemplo siguiente se define una función para el método `onDragOver` que envía una acción `trace()` al panel Salida.

```
my_mc.onDragOver = function () {  
    trace ("se ha llamado al método onDragOver");  
};
```

**Véase también**

[MovieClip.onDragOut](#)

## MovieClip.onEnterFrame

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onEnterFrame = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca de forma continua a la velocidad de los fotogramas del archivo SWF. Las acciones asociadas con el evento de clip `enterFrame` se procesan antes que cualquiera de las acciones de fotogramas asociadas a los fotogramas afectados.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onEnterFrame` que envía una acción `trace()` al panel Salida.

```
my_mc.onEnterFrame = function () {  
    trace ("se ha llamado al método onEnterFrame");  
};
```

## MovieClip.onKeyDown

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onKeyDown = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando un clip de película se selecciona en el momento de la entrada y se presiona una tecla. El controlador de eventos `onKeyDown` se invoca sin parámetros. Puede utilizar los métodos `Key.getAscii()` y `Key.getCode()` para determinar qué tecla se ha presionado. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

El controlador de eventos `onKeyDown` sólo funciona si el clip de película se ha establecido y activado en el momento de la entrada. En primer lugar, debe establecerse la propiedad `focusEnabled` en `true` para el clip de película. A continuación, debe seleccionarse el clip. Esto puede hacerse utilizando `Selection.setFocus()` o configurando la tecla de tabulación para desplazarse hasta el clip.

Si utiliza `Selection.setFocus()`, debe pasar la ruta para el clip de película a `Selection.setFocus()`. Es muy fácil para otros elementos recuperar la selección después de mover el ratón.

## Ejemplo

En el ejemplo siguiente se define una función para el método `onKeyDown()` que envía una acción `trace()` al panel Salida.

```
my_mc.onKeyDown = function () {  
    trace ("se ha llamado al método onKeyDown");  
};
```

En el ejemplo siguiente se establece la selección de entrada.

```
MovieClip.focusEnabled = true;  
Selection.setFocus(MovieClip);
```

## Véase también

[MovieClip.onKeyUp](#)

# MovieClip.onKeyUp

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onKeyUp = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se suelta una tecla. El controlador de eventos `onKeyUp` se invoca sin parámetros. Puede utilizar los métodos `Key.getAscii()` y `Key.getCode()` para determinar qué tecla se ha presionado. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

El controlador de eventos `onKeyUp` sólo funciona si se ha establecido y activado el clip de película en el momento de la entrada. En primer lugar, debe establecerse la propiedad `focusEnabled` en `true` para el clip de película. A continuación, debe seleccionarse el clip. Esto puede hacerse utilizando `Selection.setFocus()` o configurando la tecla de tabulación para desplazarse hasta el clip.

Si utiliza `Selection.setFocus()`, debe pasar la ruta para el clip de película a `Selection.setFocus()`. Es muy fácil para otros elementos recuperar la selección después de mover el ratón.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onKeyUp` que envía una acción `trace()` al panel Salida.

```
my_mc.onKeyUp = function () {  
    trace ("se ha llamado al método onKeyUp");  
};
```

En el ejemplo siguiente se establece la selección de entrada:

```
MovieClip.focusEnabled = true;  
Selection.setFocus(MovieClip);
```

## MovieClip.onKillFocus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onKillFocus = function (newFocus) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*newFocus*    Objeto que se selecciona mediante el teclado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un clip de película ya no se selecciona mediante el teclado. El método `onKillFocus` recibe un parámetro, *newFocus*, que es un objeto que representa el nuevo objeto seleccionado. Si no hay ningún objeto seleccionado, *newFocus* contiene el valor `null`.

## MovieClip.onLoad

### Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onLoad = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se crea la instancia del clip de película y aparece en la línea de tiempo. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

Este controlador se puede utilizar con clips de película que tengan un símbolo en la biblioteca que esté asociada con una clase. Si desea que se invoque un controlador de eventos cuando se carga un clip de película determinado (por ejemplo, cuando se utiliza `MovieClip.loadMovie()` para cargar un archivo SWF de forma dinámica), debe utilizar `onClipEvent(load)` en lugar de este controlador. Este último controlador se invoca cuando se carga cualquier clip de película.

## Ejemplo

En el ejemplo siguiente se ilustra el uso correcto de `MovieClip.onLoad()` y `onClipEvent(load)`.

```
// symbol_mc es un símbolo de clip de película de la biblioteca.  
// Está vinculado con la clase MovieClip.  
// Para cada instancia de symbol_mc se activa la función siguiente  
// a medida que se crean y aparecen en la línea de tiempo.  
symbol_mc.onLoad = function() {  
    trace("El clip de película se ha cargado");  
}  
  
// dynamic_mc es un clip de película que se carga con MovieClip.loadMovie().  
// Este código intenta llamar a una función cuando el clip se carga,  
// pero no funcionará, ya que el archivo SWF cargado no es un símbolo  
// de la biblioteca asociada con la clase MovieClip.  
function output()  
{  
    trace("No se llamará nunca.");  
}  
dynamic_mc.onLoad = output;  
dynamic_mc.loadMovie("replacement.swf");  
  
// La función siguiente se invoca para cualquier clip de película que  
// aparezca en la línea de tiempo, tanto si está en la biblioteca como si no.  
// Por lo tanto, esta función se invoca cuando se crea una instancia de  
symbol_mc  
// y también cuando se carga replacement.swf.  
onClipEvent( load ) {  
    trace("El clip de película se ha cargado");  
}
```

## Véase también

[onClipEvent\(\)](#)



# MovieClip.onMouseDown

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onMouseDown = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se presiona el botón del ratón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

## Ejemplo

En el ejemplo siguiente se define una función para el método `onMouseDown` que envía una acción `trace()` al panel Salida.

```
my_mc.onMouseDown = function () {  
    trace ("se ha llamado al método onMouseDown");  
}
```

# MovieClip.onMouseMove

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onMouseMove = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se mueve el ratón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onMouseMove` que envía una acción `trace()` al panel Salida.

```
my_mc.onMouseMove = function () {  
    trace ("se ha llamado al método onMouseMove");  
};
```

## MovieClip.onMouseUp

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onMouseUp = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se suelta el botón del ratón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onMouseUp` que envía una acción `trace()` al panel Salida.

```
my_mc.onMouseUp = function () {  
    trace ("se ha llamado al método onMouseUp");  
};
```

## MovieClip.onPress

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onPress = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando el usuario hace clic con el ratón mientras el puntero se encuentra sobre un clip de película. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onPress` que envía una acción `trace()` al panel Salida.

```
my_mc.onPress = function () {  
    trace ("se ha llamado al método onPress");  
};
```

## MovieClip.onRelease

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onRelease = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando se suelta un clip de película de botón. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

### Ejemplo

En el ejemplo siguiente se define una función para el método `onPress` que envía una acción `trace()` al panel Salida.

```
my_mc.onRelease = function () {  
    trace ("se ha llamado al método onRelease");  
};
```

## MovieClip.onReleaseOutside

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.onReleaseOutside = function() {  
    // las sentencias se escriben aquí  
}
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Controlador de eventos; se invoca cuando se suelta el botón del ratón mientras el puntero está fuera del clip de película después de presionar el botón del ratón dentro del clip de película.

Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

**Ejemplo**

En el ejemplo siguiente se define una función para el método `onReleaseOutside` que envía una acción `trace()` al panel Salida.

```
my_mc.onReleaseOutside = function () {  
    trace ("se ha llamado al método onReleaseOutside");  
};
```

## MovieClip.onRollOut

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_mc.onRollOut = function() {  
    // las sentencias se escriben aquí  
}
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Controlador de eventos; se invoca cuando el puntero se desplaza fuera del área de un clip de película. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

**Ejemplo**

En el ejemplo siguiente se define una función para el método `onRollOut` que envía una acción `trace()` al panel Salida.

```
my_mc.onRollOut = function () {  
    trace ("se ha llamado al método onRollOut");  
};
```

# MovieClip.onRollOver

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onRollOver = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando el puntero se desplaza sobre el área de un clip de película. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

## Ejemplo

En el ejemplo siguiente se define una función para el método `onRollOver` que envía un método `trace()` al panel Salida.

```
my_mc.onRollOver = function () {  
    trace ("se ha llamado al método onRollOver");  
};
```

# MovieClip.onSetFocus

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onSetFocus = function(oldFocus){  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*oldFocus*    Objeto que dejará de estar seleccionado.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se selecciona un clip de película con el teclado. El parámetro *oldFocus* es el objeto que deja de estar seleccionado. Por ejemplo, si el usuario presiona la tecla Tabulador para cambiar la selección de entrada de un clip de película a un campo de texto, *oldFocus* contiene la instancia del clip de película.

Si no hay ningún objeto seleccionado anteriormente, *oldFocus* contiene un valor `null`.

# MovieClip.onUnload

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.onUnload = function() {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca en el primer fotograma después de eliminar el clip de película de la línea de tiempo. Flash procesa las acciones asociadas con el controlador de eventos `onUnload` antes de asociar una acción al fotograma afectado. Debe definir una función que se ejecute cuando se invoque el controlador de eventos.

## Ejemplo

En el ejemplo siguiente se define una función para el método `MovieClip.onUnload` que envía una acción `trace()` al panel Salida.

```
my_mc.onUnload = function () {  
    trace ("se ha llamado al método onUnload");  
};
```

# MovieClip.\_parent

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_mc._parent.property  
_parent.property
```

## Descripción

Propiedad; referencia al clip de película u objeto que contiene el objeto o clip de película actual. El objeto actual es el objeto que contiene el código de ActionScript que hace referencia a `_parent`. Utilice la propiedad `_parent` para especificar una ruta relativa a los clips de película u objetos que están por encima del clip de película u objeto actual.

Puede utilizar `_parent` para subir varios niveles en la lista de visualización, como se muestra a continuación:

```
_parent._parent._alpha = 20;
```

## Véase también

`Button._parent`, `_root`, `targetPath`, `TextField._parent`

## MovieClip.play()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.play()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; mueve la cabeza lectora de la línea de tiempo del clip de película.

### Véase también

[play\(\)](#)

## MovieClip.prevFrame()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.prevFrame()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; envía la cabeza lectora al fotograma anterior y la detiene.

### Véase también

[prevFrame\(\)](#)

## MovieClip.removeMovieClip()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.removeMovieClip()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; elimina una instancia de clip de película creada con `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()` o `MovieClip.attachMovie()`.

## MovieClip.\_rotation

### Disponibilidad

Flash Player 4.

### Sintaxis

```
my_mc._rotation
```

### Descripción

Propiedad; la rotación del clip de película, en grados, de su orientación original. Los valores entre 0 y 180 corresponden a la rotación en el sentido de las agujas del reloj, mientras que los valores entre 0 y -180 corresponden a la rotación en el sentido contrario a las agujas del reloj. Los valores que quedan fuera de este rango se suman o se restan de 360 para obtener un valor dentro del rango. Por ejemplo, la sentencia `my_mc._rotation = 450` es la misma que `my_mc._rotation = 90`.

### Véase también

`Button._rotation`, `TextField._rotation`

## MovieClip.setMask()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.setMask(mask_mc)
```

### Parámetros

*my\_mc* Nombre de instancia del clip de película que se va a enmascarar.

*mask\_mc* Nombre de instancia del clip de película que se va a enmascarar.

### Valor devuelto

Ninguno.

### Descripción

Método; convierte el clip de película del parámetro `mask_mc` en una máscara que revela el clip de película especificado por el parámetro `my_mc`.



Este método permite que los clips de película de varios fotogramas con contenido complejo y de varias capas actúen como máscaras. Puede activar o desactivar las máscaras en tiempo de ejecución. Sin embargo, no puede utilizar la misma máscara para varias máscaras (lo cual es posible con el uso de capas de máscara). Si dispone de fuentes de dispositivo en un clip de película con máscara, dichas fuentes se dibujarán pero no se enmascararán. No es posible establecer que un clip de película sea su propia máscara, por ejemplo, `my_mc.setMask(my_mc)`.

Si crea una capa de máscara que contenga un clip de película y, a continuación, le aplica el método `setMask()`, la llamada a `setMask()` tiene prioridad y, por lo tanto, es irreversible. Por ejemplo, puede tener un clip de película en una capa de máscara denominada `UIMask` que enmascara otra capa que contiene un clip de película denominado `UIMaskee`. Si, durante la reproducción del archivo SWF, llama a `UIMask.setMask(UIMaskee)`, a partir de ese momento, `UIMask` quedará enmascarado por `UIMaskee`.

Para cancelar una máscara creada con ActionScript, pase el valor `null` al método `setMask()`. El código siguiente cancela la máscara sin que afecte a la capa de máscara de la línea de tiempo.

```
UIMask.setMask(null);
```

### Ejemplo

El código siguiente utiliza el clip de película `circleMask_mc` para enmascarar el clip de película `theMaskee_mc`.

```
theMaskee_mc.setMask(circleMask_mc);
```

## MovieClip.\_soundbuftime

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc._soundbuftime
```

### Descripción

Propiedad (global); entero que especifica el número de segundos que un sonido se almacena previamente en una memoria intermedia antes de que empiece a reproducirse.

## MovieClip.startDrag()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.startDrag([lock, [left, top, right, bottom]])
```

### Parámetros

*lock* Valor booleano que especifica si el clip de película arrastrable está bloqueado en el centro de la posición del ratón (`true`) o en el punto en el que el usuario hizo clic por primera vez en el clip de película (`false`). Este parámetro es opcional.

*left, top, right, bottom* Valores relativos a las coordenadas del elemento principal del clip de película que especifican un rectángulo de limitación para el clip de película. Estos parámetros son opcionales.

**Valor devuelto**

Ninguno.

**Descripción**

Método; permite al usuario arrastrar el clip de película especificado. El clip de película sigue siendo arrastrable hasta que se detiene de forma explícita mediante una llamada a `MovieClip.stopDrag()` o hasta que otro clip de película se convierte en arrastrable. Sólo puede arrastrarse un clip de película cada vez.

**Véase también**

`MovieClip._droptarget`, `startDrag()`, `MovieClip.stopDrag()`

## MovieClip.stop()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_mc.stop()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Método; detiene el clip de película que se está reproduciendo actualmente.

**Véase también**

`stop()`

## MovieClip.stopDrag()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_mc.stopDrag()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

### Descripción

Método; pone fin a un método `MovieClip.startDrag()`. Un clip de película que se convirtió en arrastrable con ese método continúa siendo arrastrable hasta que se añade un método `stopDrag()` o hasta que otro clip de película se convierte en arrastrable. Sólo puede arrastrarse un clip de película cada vez.

### Véase también

`MovieClip._droptarget`, `MovieClip.startDrag()`, `stopDrag()`

## MovieClip.swapDepths()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.swapDepths(depth)  
my_mc.swapDepths(target)
```

### Parámetros

*depth* Número que especifica el nivel de profundidad donde se va a colocar *my\_mc*.

*target* Cadena que especifica la instancia de clip de película cuya profundidad se intercambia con la instancia especificada por *my\_mc*. Ambas instancias deben tener el mismo clip de película principal.

### Valor devuelto

Ninguno.

### Descripción

Método; intercambia el apilamiento, o el orden *z* (nivel de profundidad), de la instancia especificada (*my\_mc*) con el clip de película especificado por el parámetro *target* o con el clip de película que actualmente ocupa el nivel de profundidad especificado en el parámetro *depth*. Ambos clips de película deben tener el mismo clip de película principal. Al intercambiar el nivel de profundidad de los clips de película se mueve un clip de película frente o detrás del otro. Si el clip de película se está interpolando cuando se llama a este método, la interpolación se detiene. Para más información, consulte “[Gestión de las profundidades de los clips de película](#)” en [la página 132](#).

### Véase también

`_level`, `MovieClip.getDepth()`, `MovieClip.getInstanceAtDepth()`,  
`MovieClip.getNextHighestDepth()`

## MovieClip.tabChildren

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.tabChildren
```

## Descripción

Propiedad; el valor predeterminado es `undefined`. Si `tabChildren` tiene el valor `undefined` o `true`, los elementos secundarios de un clip de película se incluyen en el orden de tabulación automático. Si el valor de `tabChildren` es `false`, los elementos secundarios de un clip de película no se incluyen en el orden de tabulación automático.

## Ejemplo

Un objeto de interfaz de cuadro de lista creado como clip de película contiene varios elementos. El usuario puede hacer clic en cada elemento para seleccionarlo, de modo que cada elemento es un botón. Sin embargo, únicamente el cuadro de lista puede ser una tabulación. Los elementos que contiene el cuadro de lista deben excluirse del orden de tabulación. Para ello, la propiedad `tabChildren` del cuadro de lista debe estar establecida en `false`.

La propiedad `tabChildren` no tiene ningún efecto si se utiliza la propiedad `tabIndex`; la propiedad `tabChildren` sólo afecta al orden de tabulación automático.

## Véase también

[Button.tabIndex](#), [MovieClip.tabEnabled](#), [MovieClip.tabIndex](#), [TextField.tabIndex](#)

# MovieClip.tabEnabled

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.tabEnabled
```

## Descripción

Propiedad; especifica si `my_mc` se incluye en el orden de tabulación automático. El valor predeterminado es `undefined`.

Si `tabEnabled` es `undefined`, el objeto sólo se incluye en el orden de tabulación automático si define al menos un controlador de botón, como por ejemplo [MovieClip.onRelease](#). Si `tabEnabled` es `true`, el objeto se incluye en el orden de tabulación automático. Si la propiedad `tabIndex` también se establece en un valor, el objeto se incluye en el orden de tabulación personalizado.

Si `tabEnabled` es `false`, el objeto no se incluye en el orden de tabulación personalizado o automático, aunque se establezca la propiedad `tabIndex`. Sin embargo, si [MovieClip.tabChildren](#) es `true`, los elementos secundarios del clip de película pueden seguir incluyéndose en el orden de tabulación automático, aun cuando el valor de `tabEnabled` sea `false`.

## Véase también

[Button.tabEnabled](#), [MovieClip.tabChildren](#), [MovieClip.tabIndex](#), [TextField.tabEnabled](#)

# MovieClip.tabIndex

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_mc.tabIndex*

## Descripción

Propiedad; permite personalizar el orden de tabulación de los objetos de una película. El valor predeterminado de la propiedad `tabIndex` es `undefined`. Puede establecer `tabIndex` en un botón, un clip de película o una instancia de campo de texto.

Si un objeto de un archivo SWF contiene una propiedad `tabIndex` el orden de tabulación automático se desactiva y el orden de tabulación se calcula a partir de las propiedades `tabIndex` de los objetos del archivo SWF. El orden de tabulación personalizado sólo incluye objetos que tienen propiedades `tabIndex`.

La propiedad `tabIndex` debe ser un entero positivo. Los objetos se ordenan de acuerdo con sus propiedades `tabIndex`, en orden ascendente. Un objeto cuyo valor `tabIndex` es 1 precede a un objeto cuyo valor `tabIndex` es 2. El orden de tabulación personalizado no tiene en cuenta las relaciones jerárquicas de los objetos de un archivo SWF. Todos los objetos del archivo SWF con las propiedades `tabIndex` se colocan según el orden de tabulación. No debe utilizarse el mismo valor de `tabIndex` para varios objetos.

## Véase también

[Button.tabIndex](#), [TextField.tabIndex](#)

# MovieClip.\_target

## Disponibilidad

Flash Player 4.

## Sintaxis

*my\_mc.\_target*

## Descripción

Propiedad (sólo lectura); devuelve la ruta de destino de la instancia de clip de película especificada por *my\_mc*.

# MovieClip.\_totalframes

## Disponibilidad

Flash Player 4.

## Sintaxis

*my\_mc.\_totalframes*

## Descripción

Propiedad (sólo lectura); devuelve el número total de fotogramas de la instancia de clip de película especificada en el parámetro *MovieClip*.

## MovieClip.trackAsMenu

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_mc.trackAsMenu
```

### Descripción

Propiedad; propiedad booleana que indica si los botones o los clips de película pueden recibir o no eventos al soltar el botón del ratón. Permite crear menús. Puede establecer la propiedad `trackAsMenu` en cualquiera de los objetos de clip de película o botón. Si la propiedad `trackAsMenu` no existe, el comportamiento predeterminado es `false`.

Puede cambiar la propiedad `trackAsMenu` en cualquier momento; el clip de película de botón modificado asume de inmediato el nuevo comportamiento.

### Véase también

[Button.trackAsMenu](#)

## MovieClip.unloadMovie()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_mc.unloadMovie()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; elimina el contenido de una instancia de clip de película. Las propiedades de la instancia y los controladores del clip permanecen.

Para eliminar la instancia, incluidas las propiedades y los controladores de clip, utilice [MovieClip.removeMovieClip\(\)](#).

### Véase también

[MovieClip.attachMovie\(\)](#), [MovieClip.loadMovie\(\)](#), [unloadMovie\(\)](#), [unloadMovieNum\(\)](#)

## MovieClip.\_url

### Disponibilidad

Flash Player 4.

## Sintaxis

```
my_mc._url
```

## Descripción

Propiedad (sólo lectura); recupera la URL del archivo SWF desde el que se descargó el clip de película.

# MovieClip.useHandCursor

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_mc.useHandCursor
```

## Descripción

Propiedad; valor booleano que indica si aparece el cursor con forma de mano cuando se desplaza el ratón sobre un clip de película de botón. El valor predeterminado de `useHandCursor` es `true`. Si `useHandCursor` se establece en `true`, el puntero en forma de mano que se utiliza para los botones aparece cuando el ratón se desplaza por encima de un clip de película de botón. En cambio, si `useHandCursor` es `false`, se utiliza el cursor con forma de flecha.

Puede cambiar la propiedad `useHandCursor` en cualquier momento; el clip de película de botón modificado asume de inmediato el comportamiento del nuevo cursor. La propiedad `useHandCursor` puede leerse de un objeto prototipo.

# MovieClip.\_visible

## Disponibilidad

Flash Player 4.

## Sintaxis

```
my_mc._visible
```

## Descripción

Propiedad; valor booleano que indica si el clip de película especificado por `my_mc` es visible. Los clips de película que no son visibles (propiedad `_visible` establecida en `false`) están desactivados. Por ejemplo, no es posible hacer clic en un botón de un clip de película con el valor de `_visible` establecido en `false`.

## Véase también

[Button.\\_visible](#), [TextField.\\_visible](#)

# MovieClip.\_width

## Disponibilidad

Flash Player 4 como propiedad de sólo lectura.

### Sintaxis

`my_mc._width`

### Descripción

Propiedad; anchura del clip de película, expresada en píxeles.

### Ejemplo

En el ejemplo siguiente se establecen las propiedades de altura y anchura de un clip de película cuando el usuario hace clic en el botón del ratón.

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

### Véase también

[MovieClip.\\_height](#)

## MovieClip.\_x

### Disponibilidad

Flash Player 3.

### Sintaxis

`my_mc._x`

### Descripción

Propiedad; número entero que establece la coordenada *x* de un clip de película en relación con las coordenadas locales del clip de película principal. Si un clip de película está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el clip de película se encuentra dentro de otro clip de película que tiene transformaciones, el clip de película está en el sistema de coordenadas local del clip de película que lo contiene. Por consiguiente, para un clip de película que se ha girado 90 grados en sentido contrario a las agujas del reloj, los elementos secundarios del clip de película heredan un sistema de coordenadas que se ha girado 90 grados en el mismo sentido. Las coordenadas del clip de película hacen referencia a la posición del punto de registro.

### Véase también

[MovieClip.\\_xscale](#), [MovieClip.\\_y](#), [MovieClip.\\_yscale](#)

## MovieClip.\_xmouse

### Disponibilidad

Flash Player 5.

### Sintaxis

`my_mc._xmouse`

### Descripción

Propiedad (sólo lectura); devuelve la coordenada *x* de la posición del ratón.



Véase también

[Clase Mouse](#), [MovieClip.\\_ymouse](#)

## MovieClip.\_xscale

### Disponibilidad

Flash Player 4.

### Sintaxis

*my\_mc.\_xscale*

### Descripción

Propiedad; determina la escala horizontal (*porcentaje*) del clip de película que se aplica desde el punto de registro del clip de película. El punto de registro predeterminado es (0,0).

Cambiar la escala del sistema de coordenadas local afecta a la configuración de las propiedades `_x` e `_y`, que se definen en píxeles. Por ejemplo, si se cambia la escala del clip de película principal al 50%, al establecer la propiedad `_x` se mueve un objeto en el clip de película la mitad del número de píxeles que se movería si la película estuviera al 100%.

Véase también

[MovieClip.\\_x](#), [MovieClip.\\_y](#), [MovieClip.\\_yscale](#)

## MovieClip.\_y

### Disponibilidad

Flash Player 3.

### Sintaxis

*my\_mc.\_y*

### Descripción

Propiedad; establece la coordenada *y* de un clip de película en relación con las coordenadas locales del clip de película principal. Si un clip de película está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el clip de película se encuentra dentro de otro clip de película que tiene transformaciones, el clip de película está en el sistema de coordenadas local del clip de película que lo contiene. Por consiguiente, para un clip de película que se ha girado 90 grados en sentido contrario a las agujas del reloj, los elementos secundarios del clip de película heredan un sistema de coordenadas que se ha girado 90 grados en el mismo sentido. Las coordenadas del clip de película hacen referencia a la posición del punto de registro.

Véase también

[MovieClip.\\_x](#), [MovieClip.\\_xscale](#), [MovieClip.\\_yscale](#)

## MovieClip.\_ymouse

### Disponibilidad

Flash Player 5.

### Sintaxis

*my\_mc.\_ymouse*

### Descripción

Propiedad (sólo lectura); indica la coordenada *y* de la posición del ratón.

### Véase también

[Clase Mouse](#), [MovieClip.\\_xmouse](#)

## MovieClip.\_yscale

### Disponibilidad

Flash Player 4.

### Sintaxis

*my\_mc.\_yscale*

### Descripción

Propiedad; establece la escala vertical (*porcentaje*) del clip de película que se aplica desde el punto de registro del clip de película. El punto de registro predeterminado es (0,0).

Cambiar la escala del sistema de coordenadas local afecta a la configuración de las propiedades *\_x* e *\_y*, que se definen en píxeles. Por ejemplo, si se cambia la escala del clip de película principal al 50%, al establecer la propiedad *\_x* se mueve un objeto en el clip de película la mitad del número de píxeles que se movería si la película estuviera al 100%.

### Véase también

[MovieClip.\\_x](#), [MovieClip.\\_xscale](#), [MovieClip.\\_y](#)

## Clase MovieClipLoader

### Disponibilidad

Flash Player 7.

### Descripción

Esta clase permite implementar callbacks del detector que proporcionan información de estado mientras se cargan (descargan) los archivos SWF o JPEG en los clips de película. Para utilizar las funciones `MovieClipLoader`, utilice [MovieClipLoader.loadClip\(\)](#) en lugar de `loadMovie()` o `MovieClip.loadMovie()` para cargar archivos SWF.

Después de emitir el comando `MovieClipLoader.loadClip()`, se producen los eventos siguientes en el orden de la lista:

- Cuando los primeros bytes del archivo descargado se han escrito en el disco, se invoca el detector [MovieClipLoader.onLoadStart\(\)](#).

- Si ha implementado el detector `MovieClipLoader.onLoadProgress()`, se invocará durante el proceso de carga.

**Nota:** puede llamar a `MovieClipLoader.getProgress()` en cualquier momento durante el proceso de carga.

- Cuando la totalidad del archivo descargado se ha escrito en el disco, se invoca el detector `MovieClipLoader.onLoadComplete()`.
- Una vez que se hayan ejecutado las acciones del primer fotograma del archivo descargado, se invoca el detector `MovieClipLoader.onLoadInit()`.

Una vez que se haya invocado `MovieClipLoader.onLoadInit()`, puede establecer las propiedades, utilizar los métodos e interactuar de otras formas con la película que ha cargado.

Si se produce un error que impide la descarga completa del archivo, se invoca el detector `MovieClipLoader.onLoadError()`.

## Resumen de métodos para la clase `MovieClipLoader`

Método	Descripción
<code>MovieClipLoader.addListener()</code>	Registra un objeto para que reciba una notificación cuando se invoque un controlador de eventos <code>MovieClipLoader</code> .
<code>MovieClipLoader.getProgress()</code>	Devuelve el número de bytes cargados y el número total de bytes de un archivo que se esté cargando mediante <code>MovieClipLoader.loadClip()</code> .
<code>MovieClipLoader.loadClip()</code>	Carga archivos SWF o JPEG en un clip de película de Flash Player mientras se reproduce la película original.
<code>MovieClipLoader.removeListener()</code>	Elimina un objeto registrado mediante <code>MovieClipLoader.addListener()</code> .
<code>MovieClipLoader.unloadClip()</code>	Elimina un clip de película que se ha cargado mediante <code>MovieClipLoader.loadClip()</code> .

## Resumen de detectores para la clase `MovieClipLoader`

Detector	Descripción
<code>MovieClipLoader.onLoadComplete()</code>	Se invoca cuando un archivo cargado mediante <code>MovieClipLoader.loadClip()</code> ha terminado de descargarse.
<code>MovieClipLoader.onLoadError()</code>	Se invoca cuando un archivo cargado mediante <code>MovieClipLoader.loadClip()</code> no puede cargarse.
<code>MovieClipLoader.onLoadInit()</code>	Se invoca cuando las acciones del primer fotograma del clip cargado se han ejecutado.
<code>MovieClipLoader.onLoadProgress()</code>	Se invoca cada vez que el contenido cargado se escribe en el disco durante el proceso de carga.
<code>MovieClipLoader.onLoadStart()</code>	Se invoca cuando una llamada a <code>MovieClipLoader.loadClip()</code> ha iniciado correctamente la descarga de un archivo.

## Constructor para la clase MovieClipLoader

### Disponibilidad

Flash Player 7.

### Sintaxis

```
new MovieClipLoader()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto MovieClipLoader que puede utilizar para implementar una serie de detectores que respondan a los eventos mientras se estén descargando archivos SWF o JPEG.

### Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

### Véase también

[MovieClipLoader.addListener\(\)](#)

## MovieClipLoader.addListener()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_mcl.addListener(listenerObject)
```

### Parámetros

*listenerObject* Objeto que detecta las notificaciones callback de los controladores de eventos MovieClipLoader.

### Valor devuelto

Ninguno.

### Descripción

Método; registra un objeto para que reciba una notificación cuando se invoque un controlador de eventos MovieClipLoader.

### Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

### Véase también

[MovieClipLoader.onLoadComplete\(\)](#), [MovieClipLoader.onLoadError\(\)](#),  
[MovieClipLoader.onLoadInit\(\)](#), [MovieClipLoader.onLoadProgress\(\)](#),  
[MovieClipLoader.onLoadStart\(\)](#), [MovieClipLoader.removeListener\(\)](#)

## MovieClipLoader.getProgress()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_mcl.getProgress(target_mc)
```

### Parámetros

*target\_mc* Archivo SWF o JPEG que se carga utilizando [MovieClipLoader.loadClip\(\)](#).

### Valor devuelto

Objeto que tiene dos propiedades de entero: `bytesLoaded` y `bytesTotal`.

### Descripción

Método; devuelve el número de bytes cargados y el número de bytes totales del archivo que se está cargando mediante [MovieClipLoader.loadClip\(\)](#); para las películas comprimidas, refleja el número de bytes comprimidos. Este método permite solicitar explícitamente esta información, en lugar de (o además de) escribir una función de detector [MovieClipLoader.onLoadProgress\(\)](#).

### Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

### Véase también

[MovieClipLoader.onLoadProgress\(\)](#)

## MovieClipLoader.loadClip()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_mcl.loadMovie("url", target )
```

### Parámetros

*url* URL absoluta o relativa del archivo SWF o JPEG que se debe cargar. Una ruta relativa debe ser relativa respecto al archivo SWF del nivel 0. Los URL absolutos deben incluir la referencia al protocolo, como `http://` o `file:///`. Los nombres de archivo no pueden incluir especificaciones de unidad de disco.

*target* Ruta de destino de un clip de película o número entero que especifica el nivel de Flash Player en el que se cargará la película. El clip de película de destino se sustituirá por la imagen o película cargada.

## Valor devuelto

Ninguno.

## Descripción

Método; carga archivos SWF o JPEG en un clip de película de Flash Player mientras se reproduce la película original. El método `loadMovie()` permite ver varias películas al mismo tiempo y pasar de una película a otra sin cargar otro documento HTML.

La utilización de este método en lugar de `loadMovie()` o `MovieClip.loadMovie()` ofrece varias ventajas:

- El controlador `MovieClipLoader.onLoadStart()` se invoca cuando se inicia la carga.
- El controlador `MovieClipLoader.onLoadError()` se invoca si no se puede cargar el clip.
- El controlador `MovieClipLoader.onLoadProgress()` se invoca a medida que evoluciona el proceso.
- El controlador `MovieClipLoader.onLoadInit()` se invoca después de que se hayan ejecutado las acciones del primer fotograma del clip, de forma que puede empezar a manipular el clip cargado.

Una película o imagen cargada en un clip de película hereda las propiedades de posición, rotación y escala del clip de película. Puede utilizar la ruta de destino del clip de película para acceder a la película cargada.

Puede utilizar este método para cargar uno o más archivos en un clip de película o en un nivel. La instancia del clip de película destino que se pasa como parámetro a los objetos detectores `MovieClipLoader`. En su lugar, puede crear un objeto `MovieClipLoader` diferente para cada archivo que cargue.

Utilice `MovieClipLoader.unloadClip()` para eliminar películas o imágenes cargadas mediante este método o para cancelar una operación de carga que esté en curso.

## Ejemplo

En el ejemplo siguiente se muestra el uso de muchos de los métodos y detectores `MovieClipLoader`.

```
// primer conjunto de detectores
var my_mcl = new MovieClipLoader();
myListener = new Object();
myListener.onLoadStart = function (target_mc)
{
    myTrace ("*****First my_mcl instance*****");
    myTrace ("La carga ha empezado en el clip de película " + target_mc);
    var loadProgress = my_mcl.getProgress(target_mc);
    myTrace(loadProgress.bytesLoaded + " = bytes cargado al iniciar");
    myTrace(loadProgress.bytesTotal + " = bytes totales al iniciar");
}
myListener.onLoadProgress = function (target_mc, loadedBytes, totalBytes)
{
    myTrace ("*****First my_mcl instance Progress*****");
    myTrace ("Se repite la llamada a onLoadProgress() en el clip de película " +
        target_mc);
    myTrace(loadedBytes + " = bytes cargados durante el curso de callback " );
    myTrace(totalBytes + " = bytes totales durante el curso de callback \n");
}
myListener.onLoadComplete = function (target_mc)
```

```

{
myTrace ("*****First my_mcl instance*****");
myTrace ("Ha finalizado la carga en el clip de película = " + target_mc);
var loadProgress = my_mcl.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes cargados al finalizar" );
myTrace(loadProgress.bytesTotal + " = bytes totales al finalizar=");
}
myListener.onLoadInit = function (target_mc)
{
myTrace ("*****First my_mcl instance*****");
myTrace ("Clip de película = " + target_mc + " se ha inicializado ");
// ahora puede llevar a cabo la configuración necesaria, por ejemplo:
target_mc._width = 100;
target_mc._width = 100;
}
myListener.onLoadError = function (target_mc, errorCode)
{
myTrace ("*****First my_mcl instance*****");
myTrace ("ERROR CODE = " + errorCode);
myTrace ("Error en la carga del clip de película = " + target_mc + "\n");
}
my_mcl.addListener(myListener);
//Ahora cargue los archivos en sus destinos.
// cargas en los clips de película - cadenas utilizadas como destino
my_mcl.loadClip("http://www.somedomain.somewhere.com/
    someFile.swf", "_root.myMC");
my_mcl.loadClip("http://www.somedomain.somewhere.com/someOtherFile.swf",
    "_level0.myMC2");
//error en la carga
my_mcl.loadClip("http://www.somedomain.somewhere.com/someFile.jpg",
    _root.myMC5);

// carga en clips de película - instancias de clip de película utilizadas como
destino.
my_mcl.loadClip("http://www.somedomain.somewhere.com/someOtherFile.jpg",
    _level0.myMC3);

// carga en _level1
my_mcl.loadClip("file:///C:/media/images/somePicture.jpg", 1);

//Segundo conjunto de detectores
var another_mcl = new MovieClipLoader();
myListener2 = new Object();
myListener2.onLoadStart = function (target_mc)
{
myTrace("*****Second my_mcl instance*****");
myTrace ("La carga ha empezado en el clip de la película 22 . = " + target_mc);
var loadProgress = my_mcl.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes cargados al iniciar" );
myTrace(loadProgress.bytesTotal + " = bytes totales al iniciar");
}
myListener2.onLoadComplete = function (target_mc)
{
myTrace ("*****Second my_mcl instance*****");
myTrace ("Ha finalizado la carga en el clip de película = " + target_mc);
var loadProgress = my_mcl.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes cargados al finalizar");
myTrace(loadProgress.bytesTotal + " = bytes totales al finalizar" );
}
myListener2.onLoadError = function (target_mc, errorCode)
{

```

```

myTrace ("*****Second my_mcl instance*****");
myTrace ("ERROR CODE = " + errorCode);
myTrace ("Error en la carga del clip de película = " + target_mc + "\n");
}
another_mcl.addListener(myListener2);
//Ahora cargue los archivos en sus destinos (utilizando la segunda instancia de
MovieClipLoader)
another_mcl.loadClip("http://www.somedomain.somewhere.com/yetAnotherFile.jpg",
_root.myMC4);
// Emita las sentencias siguientes después de completar la descarga,
// y después de llamar a my_mcl.onLoadInit.
// my_mcl.removeListener(myListener)
// my_mcl.removeListener(myListener2)

```

#### Véase también

[MovieClipLoader.unloadClip\(\)](#)

## MovieClipLoader.onLoadComplete()

#### Disponibilidad

Flash Player 7.

#### Sintaxis

```

listenerObject.onLoadComplete() = function(target_mc) {
    // las sentencias se escriben aquí
}

```

#### Parámetros

*listenerObject* Objeto detector que se ha añadido mediante [MovieClipLoader.addListener\(\)](#).

*target\_mc* Clip de película que ha cargado un método [MovieClipLoader.loadClip\(\)](#).

#### Valor devuelto

Ninguno.

#### Descripción

Detector; se invoca cuando un archivo cargado mediante [MovieClipLoader.loadClip\(\)](#) ha terminado de cargarse.

#### Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

#### Véase también

[MovieClipLoader.addListener\(\)](#), [MovieClipLoader.onLoadStart\(\)](#), [MovieClipLoader.onLoadError\(\)](#)



# MovieClipLoader.onLoadError()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
listenerObject.onLoadError() = function(target_mc, errorCode) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*listenerObject* Objeto detector que se ha añadido mediante [MovieClipLoader.addListener\(\)](#).

*target\_mc* Clip de película que ha cargado un método [MovieClipLoader.loadClip\(\)](#).

*errorCode* Cadena que explica la razón del error.

## Valor devuelto

Una de dos cadenas: “URLNotFound” o “LoadNeverCompleted”.

## Descripción

Detector; se invoca cuando un archivo cargado mediante [MovieClipLoader.loadClip\(\)](#) no puede cargarse.

La cadena “URLNotFound” se devuelve si no se ha llamado al detector [MovieClipLoader.onLoadStart\(\)](#) ni [MovieClipLoader.onLoadComplete\(\)](#). Por ejemplo, si un servidor no funciona o no se encuentra el archivo, no se llama a estos detectores.

La cadena “LoadNeverCompleted” se devuelve si se ha llamado a [MovieClipLoader.onLoadStart\(\)](#) pero no a [MovieClipLoader.onLoadComplete\(\)](#). Por ejemplo, si se ha llamado a [MovieClipLoader.onLoadStart\(\)](#) pero se interrumpe la descarga a causa de la sobrecarga o el bloqueo del servidor, o a otra razón similar, no se llamará a [MovieClipLoader.onLoadComplete\(\)](#).

## Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

# MovieClipLoader.onLoadInit()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
listenerObject.onLoadInit() = function(target_mc) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*listenerObject* Objeto detector que se ha añadido mediante [MovieClipLoader.addListener\(\)](#).

*target\_mc* Clip de película que ha cargado un método [MovieClipLoader.loadClip\(\)](#).

### Valor devuelto

Ninguno.

### Descripción

Detector; se invoca cuando las acciones del primer fotograma del clip cargado se han ejecutado. Una vez que se haya invocado este detector, puede establecer las propiedades, utilizar los métodos e interactuar de otras formas con la película que ha cargado.

### Ejemplo

Véase `MovieClipLoader.loadClip()`.

### Véase también

`MovieClipLoader.onLoadStart()`

## MovieClipLoader.onLoadProgress()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
listenerObject.onLoadProgress() =  
    function(target_mc [, loadedBytes [, totalBytes ] ] ) {  
        // las sentencias se escriben aquí  
    }
```

### Parámetros

*listenerObject* Objeto detector que se ha añadido mediante `MovieClipLoader.addListener()`.

*target\_mc* Clip de película que ha cargado un método `MovieClipLoader.loadClip()`.

*loadedBytes* Número de bytes que se había cargado cuando se invocó al detector.

*totalBytes* Número total de bytes del archivo que se está cargando.

### Valor devuelto

Ninguno.

### Descripción

Detector; se invoca cada vez que el contenido cargado se escribe en el disco durante el proceso de carga (es decir, entre `MovieClipLoader.onLoadStart()` y `MovieClipLoader.onLoadComplete()`). Puede utilizar este método para mostrar información sobre el progreso de la descarga, mediante los parámetros `loadedBytes` y `totalBytes`.

### Ejemplo

Véase `MovieClipLoader.loadClip()`.

### Véase también

`MovieClipLoader.getProgress()`

# MovieClipLoader.onLoadStart()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
listenerObject.onLoadStart() = function(target_mc) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*listenerObject* Objeto detector que se ha añadido mediante [MovieClipLoader.addListener\(\)](#).

*target\_mc* Clip de película que ha cargado un método [MovieClipLoader.loadClip\(\)](#).

## Valor devuelto

Ninguno.

## Descripción

Detector; se invoca cuando una llamada a [MovieClipLoader.loadClip\(\)](#) ha iniciado correctamente la descarga de un archivo.

## Ejemplo

Véase [MovieClipLoader.loadClip\(\)](#).

## Véase también

[MovieClipLoader.onLoadError\(\)](#), [MovieClipLoader.onLoadInit\(\)](#),  
[MovieClipLoader.onLoadComplete\(\)](#)

# MovieClipLoader.removeListener()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_mcl.removeListener(listenerObject)
```

## Parámetros

*listenerObject* Objeto detector que se ha añadido mediante [MovieClipLoader.addListener\(\)](#).

## Valor devuelto

Ninguno.

## Descripción

Método; elimina un objeto que se ha utilizado para recibir notificación cuando se ha invocado el controlador de eventos [MovieClipLoader](#).

### Ejemplo

Véase `MovieClipLoader.loadClip()`.

## MovieClipLoader.unloadClip()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_mcl.unloadClip(target)
```

### Parámetros

*target* Cadena o número entero que se pasa a la correspondiente llamada a `my_mcl.loadClip()`.

### Valor devuelto

Ninguno.

### Descripción

Método; elimina un clip de película que se ha cargado mediante `MovieClipLoader.loadClip()`. Si emite este comando mientras se está cargando una película, se invoca `MovieClipLoader.onLoadError()`.

### Véase también

`MovieClipLoader.loadClip()`

## NaN

### Disponibilidad

Flash Player 5.

### Sintaxis

NaN

### Descripción

Variable; variable predefinida con el valor IEEE-754 para NaN (No es un número, Not a Number). Para determinar si un número es NaN, utilice `isNaN()`.

### Véase también

`isNaN()`, `Number.NaN`

## ne (no igual; específico para cadenas)

### Disponibilidad

Flash Player 4. Este operador se ha sustituido por el operador `!=` (*desigualdad*).

### Sintaxis

```
expression1 ne expression2
```

### Parámetros

*expression1*, *expression2*    Números, cadenas o variables.

### Valor devuelto

Valor booleano.

### Descripción

Operador (de comparación); compara *expression1* con *expression2* y devuelve `true` si *expression1* no es igual a *expression2*; en caso contrario, devuelve `false`.

### Véase también

`!=` (desigualdad)

## Clase NetConnection

### Disponibilidad

Flash Player 7.

**Nota:** esta clase también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Descripción

La clase `NetConnection` proporciona los medios para reproducir archivos FLV de flujo desde una unidad local o una dirección HTTP. Para más información sobre la reproducción de vídeo, consulte [“Reproducción dinámica de archivos FLV externos” en la página 205](#).

## Resumen de métodos de la clase NetConnection

Método	Descripción
<code>NetConnection.connect()</code>	Abre una conexión local mediante la que puede reproducir archivos de vídeo (FLV) desde una dirección HTTP o desde el sistema de archivos local.

## Constructor para la clase NetConnection

### Disponibilidad

Flash Player 7.

**Nota:** esta clase también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
new NetConnection()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

## Descripción

Constructor; crea un objeto `NetConnection` que puede utilizar junto con un objeto `NetStream` para reproducir archivos de flujo de vídeo (FLV) locales. Después de crear el objeto `NetConnection`, utilice `NetConnection.connect()` para realizar la conexión real.

La reproducción de archivos FLV externos ofrece varias ventajas frente a la incorporación de vídeo en un documento de Flash, como por ejemplo mejor rendimiento y administración de la memoria y velocidades de fotogramas de vídeo y Flash independientes. Para más información, consulte [“Reproducción dinámica de archivos FLV externos” en la página 205](#).

## Véase también

[Clase `NetStream`](#), `Video.attachVideo()`

## `NetConnection.connect()`

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
my_nc.connect(null);
```

### Parámetros

Ninguno (debe pasar `null`).

### Valor devuelto

Ninguno.

### Descripción

Constructor; abre una conexión local mediante la que puede reproducir archivos de vídeo (FLV) desde una dirección HTTP o desde el sistema de archivos local.

## Véase también

[Clase `NetStream`](#)

## Clase `NetStream`

### Disponibilidad

Flash Player 7.

**Nota:** esta clase también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Descripción

La clase `NetStream` proporciona métodos y propiedades para reproducir archivos de Flash Video (FLV) del sistema de archivos local o de una dirección HTTP. Utilice un objeto `NetStream` para transmitir vídeo mediante un objeto `NetConnection`. La reproducción de archivos FLV externos ofrece varias ventajas frente a la incorporación de vídeo en un documento de Flash, como por ejemplo mejor rendimiento y administración de la memoria y velocidades de fotogramas de vídeo y Flash independientes. Esta clase proporciona una serie de métodos y propiedades que puede utilizar para llevar el seguimiento del progreso de la carga y la reproducción de un archivo, así como para proporcionar al usuario el control de la reproducción (detener, pausa, etc.).

Para más información sobre la reproducción de vídeo, consulte [“Reproducción dinámica de archivos FLV externos” en la página 205](#).

## Resumen de métodos para la clase `NetStream`

Los métodos y propiedades siguientes de las clases `NetConnection` y `NetStream` se utilizan para controlar la reproducción de FLV.

Método	Propósito
<code>NetStream.close()</code>	Cierra el flujo pero no se borra el objeto de vídeo.
<code>NetStream.pause()</code>	Pausa o reanuda la reproducción de un flujo.
<code>NetStream.play()</code>	Inicia la reproducción de un archivo de vídeo externo (FLV).
<code>NetStream.seek()</code>	Busca una posición específica en el archivo FLV.
<code>NetStream.setBufferTime()</code>	Especifica cuántos datos debe haber en el búfer antes de empezar a visualizar el flujo.

## Resumen de propiedades para la clase `NetStream`

Propiedad	Descripción
<code>NetStream.bufferLength</code>	Número de segundos de los datos que hay en el búfer.
<code>NetStream.bufferTime</code>	Sólo lectura; número de segundos que <code>NetStream.setBufferTime()</code> asigna al búfer.
<code>NetStream.bytesLoaded</code>	Sólo lectura; número de bytes de datos que se han cargado en el reproductor.
<code>NetStream.bytesTotal</code>	Sólo lectura; tamaño total en bytes del archivo que se está cargando en el reproductor.
<code>NetStream.currentFps</code>	Número de fotogramas por segundo que se está visualizando.
<code>NetStream.time</code>	Sólo lectura; posición de la cabeza lectora, en segundos.

## Resumen de controladores de eventos para la clase `NetStream`

Controlador de eventos	Descripción
<code>NetStream.onStatus</code>	Se invoca cada vez que se genera un cambio de estado o un error para el objeto <code>NetStream</code> .

## Constructor para la clase NetStream

### Disponibilidad

Flash Player 7.

**Nota:** esta clase también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
new NetStream(my_nc)
```

### Parámetros

*my\_nc*    Objeto NetConnection.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un flujo que puede utilizarse para reproducir archivos FLV a través del objeto NetConnection especificado.

### Ejemplo

En el código siguiente, en primer lugar se construye un nuevo objeto NetConnection, *my\_nc*, y se utiliza para construir un nuevo objeto NetStream denominado *videoStream\_ns*.

```
my_nc = new NetConnection();
my_nc.connect(null);
videoStream_ns = new NetStream(my_nc);
```

### Véase también

[Clase NetConnection](#), [Clase NetStream](#), [Video.attachVideo\(\)](#)

## NetStream.bufferLength

### Disponibilidad

Flash Player 7.

**Nota:** esta propiedad también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
my_ns.bufferLength
```

### Descripción

Propiedad de sólo lectura; número de segundos de los datos que hay en el búfer. Puede utilizar esta propiedad junto con [NetStream.bufferTime](#) para estimar cuánto le falta al buffer para llenarse; por ejemplo, para visualizar la respuesta a un usuario que está esperando que se carguen datos en el búfer.

### Véase también

[NetStream.bytesLoaded](#)



# NetStream.bufferTime

## Disponibilidad

Flash Player 7.

**Nota:** esta propiedad también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

*myStream.bufferTime*

## Descripción

Propiedad de sólo lectura; número de segundos que [NetStream.setBufferTime\(\)](#) asigna al búfer. El valor predeterminado es .1 (una décima de segundo). Para determinar el número de segundos que se encuentran actualmente en el buffer, utilice [NetStream.bufferLength](#).

## Véase también

[NetStream.time](#)

# NetStream.bytesLoaded

## Disponibilidad

Flash Player 7.

## Sintaxis

*my\_ns.bytesLoaded*

## Descripción

Propiedad de sólo lectura; número de bytes de datos que se han cargado en el reproductor. Puede utilizar este método junto con [NetStream.bytesTotal](#) para estimar cuánto le falta al buffer para llenarse; por ejemplo, para visualizar la respuesta a un usuario que está esperando que se carguen datos en el búfer.

## Véase también

[NetStream.bufferLength](#)

# NetStream.bytesTotal

## Disponibilidad

Flash Player 7.

## Sintaxis

*my\_ns.bytesLoaded*

## Descripción

Propiedad de sólo lectura; tamaño total en bytes del archivo que se está cargando en el reproductor.

## Véase también

[NetStream.bytesLoaded](#), [NetStream.bufferTime](#)

# NetStream.close()

## Disponibilidad

Flash Player 7.

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

```
my_ns.close()
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Método; detiene la reproducción de todos los datos del flujo, establece la propiedad `NetStream.time` en 0 y deja el flujo disponible para otro uso. Este comando también elimina la copia local de un archivo FLV que se ha descargado con HTTP.

## Ejemplo

La función `onDisconnect()` siguiente cierra una conexión y elimina la copia temporal de `someFile.flv` que se ha almacenado en el disco local.

```
my_nc = new NetConnection();
my_nc.connect(null);
my_ns = new NetStream(my_nc);
my_ns.play("http://www.someDomain.com/videos/someFile.flv");

function onDisconnect() {
    my_ns.close();
}
```

## Véase también

`NetStream.pause()`, `NetStream.play()`

# NetStream.currentFps

## Disponibilidad

Flash Player 7.

**Nota:** esta propiedad también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

```
my_ns.currentFps
```

## Descripción

Propiedad de sólo lectura; número de fotogramas por segundo que se está visualizando. Si está exportando archivos FLV para que se reproduzcan en diversos sistemas, puede comprobar este valor durante las pruebas para que le ayude a determinar la cantidad de compresión que debe aplicar al exportar el archivo.

# NetStream.onStatus

## Disponibilidad

Flash Player 7.

**Nota:** este controlador también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

```
my_ns.onStatus = function(infoObject) {  
    // Aquí aparece el código  
}
```

## Parámetros

*infoObject* Parámetro definido de acuerdo con el mensaje de estado o error. Para más información sobre este parámetro, consulte la descripción que aparece a continuación.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cada vez que se informa de un error o cambio de estado al objeto NetStream. Si desea responder a este controlador de eventos, debe crear una función para procesar el objeto de información.

Cada objeto de información tiene una propiedad `code` con una cadena que describe el resultado del controlador `onStatus`, y una propiedad `level` que contiene una cadena que puede ser "Status", o "Error".

Además de este controlador `onStatus`, Flash también proporciona una función de superclase denominada `System.onStatus`. Si se invoca `onStatus` para un objeto determinado y no existe ninguna función asignada para responderle, Flash procesará una función asignada a `System.onStatus` si existe.

Los siguientes eventos envían una notificación si ocurren ciertas actividades de NetStream.

Propiedad Code	Propiedad Level	Significado
NetStream.Buffer.Empty	Status	No se reciben datos con la rapidez suficiente como para que el búfer se llene. El flujo de datos se interrumpe hasta que el búfer vuelve a estar lleno; en ese momento se envía un mensaje <code>NetStream.Buffer.Full</code> y el flujo se reanuda.
NetStream.Buffer.Full	Status	El búfer está lleno y el flujo va a iniciarse.
NetStream.Play.Start	Status	Ha comenzado la reproducción.
NetStream.Play.Stop	Status	Se ha detenido la reproducción.
NetStream.Play.StreamNotFound	Error	No se ha podido encontrar el archivo FLV que se ha pasado al método <code>play()</code> .

### Ejemplo

El siguiente ejemplo escribe datos acerca del flujo en un archivo de registro.

```
my_ns.onStatus = function(info)
{
    _root.log_stream += "Estado del flujo.\n";
    _root.log_stream += "Evento: " + info.code + "\n";
    _root.log_stream += "Tipo: " + info.level + "\n";
}
```

### Véase también

[System.onStatus](#)

## NetStream.pause()

### Disponibilidad

Flash Player 7.

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
my_ns.pause( [ pauseResume ] )
```

### Parámetros

*pauseResume* Opcional: valor booleano que especifica si se interrumpe la reproducción (`true`) o se reanuda (`false`). Si se omite este parámetro, `NetStream.pause()` actúa como conmutador: la primera vez que se le llama en un flujo determinado, interrumpe la reproducción; la siguiente vez, la reanuda.

### Valor devuelto

Ninguno.

## Descripción

Método; coloca en pausa o reanuda la reproducción de un flujo.

la primera vez que se llama a este método (sin enviar un parámetro), coloca en pausa la reproducción; la siguiente vez, la reanuda. Puede asociar este método a un botón que el usuario pueda presionar para colocar en pausa o reanudar la reproducción.

## Ejemplo

El ejemplo siguiente muestra algunas aplicaciones de este método.

```
my_ns.pause(); // coloca en pausa la reproducción la primera vez que se emite
my_ns.pause(); // reanuda la reproducción
my_ns.pause(false); // sin efecto, continúa la reproducción
my_ns.pause(); // coloca en pausa la reproducción
```

## Véase también

[NetStream.close\(\)](#), [NetStream.play\(\)](#)

# NetStream.play()

## Disponibilidad

Flash Player 7.

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

```
my_ns.play("fileName");
```

## Parámetros

*fileName* Nombre de un archivo FLV que se desea reproducir, entre comillas. Tanto el formato `http://` como `file://` son válidos; la ubicación `file://` es siempre relativa a la ubicación del archivo SWF.

## Valor devuelto

Ninguno.

## Descripción

Método; inicia la reproducción de un archivo de vídeo externo (FLV). Para ver datos de vídeo, debe llamar a un método [Video.attachVideo\(\)](#); el sonido que acompaña al flujo del video o los archivos FLV que sólo contienen sonido se reproducen automáticamente.

Si desea controlar el sonido asociado a un archivo FLV `file`, puede usar `MovieClip.attachAudio()` para dirigir el sonido a un clip de película; a continuación, puede crear un objeto `Sound` para controlar algunos aspectos del sonido. Para más información, consulte [MovieClip.attachAudio\(\)](#).

Si no se puede encontrar el archivo FLV, se invoca el controlador de eventos [NetStream.onStatus](#). Si quiere detener un flujo que está en ejecución, use [NetStream.close\(\)](#).

Puede reproducir archivos FLV locales almacenados en el mismo directorio que el archivo SWF o en un subdirectorio; no puede hacerlo para directorios de nivel superior. Por ejemplo, si el archivo SWF se encuentra en un directorio llamado /training, y quiere reproducir un video ubicado en el directorio /training/videos, usaría la siguiente sintaxis:

```
my_ns.play("file://videos/videoName.flv");
```

Para reproducir un video almacenado en el directorio /training, usaría la siguiente sintaxis:

```
my_ns.play("file://videoName.flv");
```

### Ejemplo

El siguiente ejemplo muestra algunas maneras de utilizar el comando `NetStream.play()`.

```
// Reproduce un archivo que se encuentra en el equipo del usuario
// El directorio joe_user es un subdirectorio del directorio
// en el que está almacenado el archivo SWF
my_ns.play("file://joe_user/flash/videos/lectureJune26.flv");

// Reproduce un archivo en un servidor
my_ns.play("http://someServer.someDomain.com/flash/video/orientation.flv");
```

### Véase también

```
MovieClip.attachAudio(), NetStream.close(), NetStream.pause(),
Video.attachVideo()
```

## NetStream.seek()

### Disponibilidad

Flash Player 7.

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

### Sintaxis

```
my_ns.seek(numberOfSeconds)
```

### Parámetros

*numberOfSeconds* El valor de tiempo aproximado, en segundos, al que desplazarse en un archivo FLV. La cabeza lectora se desplaza hasta el fotograma clave más cercano a *numberOfSeconds*.

- Para volver al comienzo del flujo, pase un valor de 0 para *numberOfSeconds*.
- Para buscar un punto situado más allá del inicio del flujo, pase el número de segundos que desea avanzar. Por ejemplo, para situar la cabeza lectora a 15 segundos del principio, utilice `myStream.seek(15)`.
- Para buscar un punto con relación a la posición actual: pase `mystream.time + n` o `mystream.time - n` para buscar un punto situado a *n* segundos hacia delante o hacia atrás, respectivamente, desde la posición actual. Por ejemplo, para rebobinar 20 segundos desde la posición actual, utilice `my_ns.seek(my_ns.time - 20)`.

### Valor devuelto

Ninguno.

## Descripción

Método; busca el fotograma clave más cercano al número de segundos especificado desde el principio del flujo. La reproducción del flujo se reanuda cuando se alcanza el punto especificado del flujo.

## Ejemplo

El siguiente ejemplo muestra algunas maneras de utilizar el comando `NetStream.seek()`.

```
// Volver al principio del flujo
my_ns.seek(0);

// Mover hasta un punto situado a 30 segundos del principio del flujo
my_ns.seek(30);

// Mover a un punto que esté situado tres minutos hacia atrás con relación al
// actual.
my_ns.seek(my_ns.time - 180);
```

## Véase también

[NetStream.play\(\)](#), [NetStream.time](#)

# NetStream.setBufferTime()

## Disponibilidad

Flash Player 7.

**Nota:** este método también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

```
my_ns.setBufferTime(numberOfSeconds)
```

## Parámetros

*numberOfSeconds* El número de segundos de datos que se almacenan en el búfer antes de que Flash empiece a mostrar datos. El valor predeterminado es .1 (una décima de segundo).

## Descripción

Método; especifica cuánto tiempo se almacenan mensajes en el búfer antes de empezar a mostrar el flujo. Por ejemplo, si desea asegurarse de que los primeros 15 segundos del flujo se muestren sin interrupción, establezca *numberOfSeconds* en 15; Flash no empezará a reproducir el flujo hasta que se hayan almacenado en el búfer 15 segundos de datos.

## Véase también

[NetStream.bufferTime](#)

# NetStream.time

## Disponibilidad

Flash Player 7.

**Nota:** esta propiedad también se admite en Flash Player 6 cuando se utiliza con Flash Communication Server. Para más información, consulte la documentación de Flash Communication Server.

## Sintaxis

`my_ns.time`

## Descripción

Propiedad de sólo lectura; la posición de la cabeza lectora, en segundos.

## Véase también

[NetStream.bufferLength](#), [NetStream.bytesLoaded](#)

# new

## Disponibilidad

Flash Player 5.

## Sintaxis

`new constructor()`

## Parámetros

*constructor* Función seguida de cualquier parámetro opcional entre paréntesis. La función es habitualmente el nombre del tipo de objeto (por ejemplo, Array, Number u Object) que se va a construir.

## Valor devuelto

Ninguno.

## Descripción

Operador; crea un nuevo objeto, inicialmente anónimo, y llama a la función identificada por el parámetro *constructor*. El operador `new` pasa a la función los parámetros opcionales entre paréntesis, así como el objeto recién creado, al que se hace referencia con la palabra clave `this`. De este modo, la función constructora puede utilizar la palabra `this` para establecer las variables del objeto.

## Ejemplo

El ejemplo siguiente crea la función `Book()` y, a continuación, utiliza el operador `new` para crear los objetos `book1` y `book2`.

```
function Book(name, price){
    this.name = name;
    this.price = price;
}

book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```



### Ejemplo

En el ejemplo siguiente se utiliza el operador `new` para crear un objeto `Array` con 18 elementos:

```
golfCourse_array = new Array(18);
```

### Véase también

[\[\] \(acceso a matriz\)](#), [{} \(inicializador de objeto\)](#)

## newline

### Disponibilidad

Flash Player 4.

### Sintaxis

```
newline
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constante; inserta un carácter de retorno de carro (`\n`) que genera una línea en blanco en la salida de texto generada por el código. Utilice `newline` para crear espacio para la información que se recupera mediante una función o una acción del código.

### Ejemplo

En el ejemplo siguiente se muestra cómo `newline` muestra la salida de la acción `trace()` en varias líneas.

```
var myName:String = "Lisa", myAge:Number = 30;
trace(myName + myAge);
trace(myName + newline + myAge);
```

## nextFrame()

### Disponibilidad

Flash 2.

### Sintaxis

```
nextFrame()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; envía la cabeza lectora al fotograma siguiente y la detiene.

### Ejemplo

En este ejemplo, cuando el usuario hace clic en el botón, la cabeza lectora pasa al siguiente fotograma y se detiene.

```
on(release) {  
    nextFrame();  
}
```

## nextScene()

### Disponibilidad

Flash 2.

### Sintaxis

```
nextScene()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; envía la cabeza lectora al fotograma 1 de la escena siguiente y la detiene.

### Ejemplo

En este ejemplo, cuando un usuario suelta el botón, la cabeza lectora se envía al fotograma 1 de la escena siguiente.

```
on(release) {  
    nextScene();  
}
```

### Véase también

[prevScene\(\)](#)

## not

### Disponibilidad

Flash Player 4. Este operador se ha eliminado y se ha sustituido por el operador **!** (NOT lógico).

### Sintaxis

```
not expression
```

### Parámetros

*expression*    Cualquier variable o expresión que puede convertirse en un valor booleano.

### Descripción

Operador; realiza una operación NOT lógica en Flash Player 4.

### Véase también

[! \(NOT lógico\)](#)

## null

### Disponibilidad

Flash Player 5.

### Sintaxis

```
null
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constante; valor especial que puede asignarse a variables o puede ser devuelto por una función si no se han proporcionado datos. Puede utilizar `null` para representar valores que faltan o que no tienen un tipo de datos definido.

### Ejemplo

En un contexto numérico, `null` da como resultado 0. Se pueden realizar comprobaciones de igualdad con `null`. En esta sentencia, un nodo de árbol binario no tiene nivel secundario a la izquierda, de modo que el campo de su nivel secundario a la izquierda podría establecerse como `null`.

```
if (tree.left == null) {  
    tree.left = new TreeNode();  
}
```

## Clase Number

### Disponibilidad

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).

### Descripción

La clase `Number` es un simple objeto envoltorio para el tipo de datos primitivo `Number`. Puede manipular los valores numéricos primitivos utilizando los métodos y propiedades asociados con la clase `Number`. Esta clase es idéntica a la clase `Number` de JavaScript.

Debe utilizar un constructor al llamar a los métodos de un objeto `Number`, pero no es necesario utilizar el constructor cuando llame a las propiedades de un objeto `Number`. Los ejemplos siguientes especifican la sintaxis para llamar a los métodos y propiedades del objeto `Number`.

En el ejemplo siguiente se llama al método `toString()` de un objeto `Number`, que devuelve la cadena “1234”.

```
myNumber = new Number(1234);  
myNumber.toString();
```

Este ejemplo llama a la propiedad `MIN_VALUE` (también denominada constante) de un objeto `Number`:

```
smallest = Number.MIN_VALUE
```

## Resumen de métodos para la clase `Number`

Método	Descripción
<code>Number.toString()</code>	Devuelve la representación de cadena de un objeto <code>Number</code> .
<code>Number.valueOf()</code>	Devuelve el valor primitivo de un objeto <code>Number</code> .

## Resumen de propiedades para la clase `Number`

Propiedad	Descripción
<code>Number.MAX_VALUE</code>	Constante que representa el número mayor que se puede representar (IEEE-754 de doble precisión). Este número es aproximadamente 1,79E+308.
<code>Number.MIN_VALUE</code>	Constante que representa el número menor que se puede representar (IEEE-754 de doble precisión). Este número es aproximadamente 5e-324.
<code>Number.NaN</code>	Constante que representa el valor de Not a Number ( <code>NaN</code> ).
<code>Number.NEGATIVE_INFINITY</code>	Constante que representa el valor de infinito negativo.
<code>Number.POSITIVE_INFINITY</code>	Constante que representa el valor de infinito positivo. Este valor es el mismo que la variable global <code>Infinity</code> .

## Constructor para la clase `Number`

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Number(value)
```

### Parámetros

*value* Valor numérico del objeto `Number` que se está creando o un valor que se va a convertir en un número.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto Number. Debe utilizar el constructor Number cuando utilice `Number.toString()` y `Number.valueOf()`. No se usa un constructor cuando se utilizan las propiedades de un objeto Number. El constructor `new Number` se utiliza principalmente como marcador de posición. Un objeto Number no es lo mismo que la función `Number()` que convierte un parámetro en un valor primitivo.

### Ejemplo

El código siguiente construye nuevos objetos Number.

```
n1 = new Number(3.4);  
n2 = new Number(-10);
```

### Véase también

[Number\(\)](#)

## Number.MAX\_VALUE

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Number.MAX_VALUE
```

### Descripción

Propiedad; el número mayor que se puede representar (IEEE-754 de doble precisión). Este número es aproximadamente 1,79E+308.

## Number.MIN\_VALUE

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Number.MIN_VALUE
```

### Descripción

Propiedad; el número menor que se puede representar (IEEE-754 de doble precisión). Este número es aproximadamente 5e-324.

## Number.NaN

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Number.NaN
```

### Descripción

Propiedad; el valor IEEE-754 que representa Not A Number (NaN).

### Véase también

`isNaN()`, `NaN`

## Number.NEGATIVE\_INFINITY

### Disponibilidad

Flash Player 5.

### Sintaxis

`Number.NEGATIVE_INFINITY`

### Descripción

Propiedad; especifica el valor IEEE-754 que representa infinito negativo. El valor de esta propiedad es el mismo que el de la constante `-Infinity`.

El infinito negativo es un valor numérico especial que se devuelve cuando una operación o función matemática devuelve un valor negativo mayor de lo que se puede representar.

## Number.POSITIVE\_INFINITY

### Disponibilidad

Flash Player 5.

### Sintaxis

`Number.POSITIVE_INFINITY`

### Descripción

Propiedad; especifica el valor IEEE-754 que representa infinito positivo. El valor de esta propiedad es el mismo que el de la constante `Infinity`.

El infinito positivo es un valor numérico especial que se devuelve cuando una operación o función matemática devuelve un valor mayor de lo que se puede representar.

## Number.toString()

### Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

### Sintaxis

`myNumber.toString(radix)`

### Parámetros

*radix* Especifica la base numérica (de 2 a 36) que se utiliza en la conversión de número a cadena. Si no especifica el parámetro *radix*, el valor predeterminado es 10.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve la representación de la cadena del objeto Number especificado (*myNumber*).

Si *myNumber* es undefined, el valor devuelto es el siguiente:

- En archivos publicados para Flash Player 6 o anterior, el resultado es 0.
- En archivos publicados para Flash Player 7 o posterior, el resultado es NaN.

### Ejemplo

El ejemplo siguiente utiliza 2 y 8 para el parámetro *radix* y devuelve una cadena que contiene la correspondiente representación del número 9.

```
myNumber = new Number (9);  
trace(myNumber.toString(2)); / 1001  
trace(myNumber.toString(8)); / 11
```

### Véase también

[NaN](#)

## Number.valueOf()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myNumber.valueOf()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número.

### Descripción

Método; devuelve el tipo de valor primitivo del objeto Number especificado.

## Number()

### Disponibilidad

Flash Player 4; comportamiento modificado en Flash Player 7.

### Sintaxis

```
Number(expression)
```

### Parámetros

*expression* Expresión que debe convertirse en un número.

### Valor devuelto

Un número NaN.

## Descripción

Función; convierte el parámetro *expression* en un número y devuelve un valor de acuerdo con lo siguiente:

- Si *expression* es un número, el valor devuelto es *expression*.
- Si *expression* es un valor booleano, el valor devuelto es 1 si *expression* es true, o bien 0 si *expression* es false.
- Si *expression* es una cadena, la función intenta analizar *expression* como un número decimal con un exponente final opcional, es decir, 1,57505e-3.
- Si *expression* es undefined, el valor devuelto es el siguiente:
  - En archivos publicados para Flash Player 6 o anterior, el resultado es 0.
  - En archivos publicados para Flash Player 7 o posterior, el resultado es NaN.

Esta función se utiliza para convertir los archivos de Flash 4 que contienen operadores desestimados que se importan al entorno de edición de Flash 5 o versiones posteriores. Para más información, consulte [& \(operador AND en modo bit\)](#).

## Véase también

[NaN](#), [Clase Number](#)

# Clase Object

## Disponibilidad

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).

## Descripción

La clase Object se encuentra en la raíz de la jerarquía de clases de ActionScript. Esta clase contiene un pequeño subconjunto de las funciones proporcionadas por la clase Object de JavaScript.

## Resumen de métodos para la clase Object

Método	Descripción
<a href="#">Object.addProperty()</a>	Crea una propiedad de captador/definidor de un objeto.
<a href="#">Object.registerClass()</a>	Asocia un símbolo de clip de película con una clase de objeto de ActionScript.
<a href="#">Object.toString()</a>	Convierte el objeto especificado en una cadena y la devuelve.
<a href="#">Object.unwatch()</a>	Elimina el punto de observación que ha creado <a href="#">Object.watch()</a> .
<a href="#">Object.valueOf()</a>	Devuelve el valor primitivo de un objeto.
<a href="#">Object.watch()</a>	Registra el controlador de eventos que se invoca cuando cambia una propiedad específica de un objeto ActionScript.



## Resumen de propiedades para la clase Object

Propiedad	Descripción
<code>Object.__proto__</code>	Referencia a la propiedad <code>prototype</code> de la función constructor del objeto.

### Constructor para la clase Object

#### Disponibilidad

Flash Player 5.

#### Sintaxis

```
new Object([value])
```

#### Parámetros

*value* Número, valor booleano o cadena que se va a convertir en un objeto. Este parámetro es opcional. Si no especifica el parámetro *value*, el constructor crea un nuevo objeto sin propiedades definidas.

#### Valor devuelto

Ninguno.

#### Descripción

Constructor; crea un nuevo objeto Object.

## Object.addProperty()

#### Disponibilidad

Flash Player 6. En archivos de clase externos, puede utilizar `get` o `set` en lugar de este método.

#### Sintaxis

```
myObject.addProperty(prop, getFunc, setFunc)
```

#### Parámetros

*prop* Nombre de la propiedad de objeto que se debe crear.

*getFunc* Función que se invoca para recuperar el valor de la propiedad; este parámetro es un objeto de función.

*setFunc* Función que se invoca para definir el valor de la propiedad; este parámetro es un objeto de función. Si pasa el valor `null` en este parámetro, la propiedad es de sólo lectura.

#### Valor devuelto

Devuelve el valor `true` si la propiedad se crea correctamente; en caso contrario, devuelve `false`.

#### Descripción

Método; crea una propiedad de captador/definidor. Cuando Flash lee una propiedad de captador/definidor, invoca la función `get` y el valor devuelto por la función se convierte en un valor de *prop*. Cuando Flash escribe una propiedad de captador/definidor, invoca la función `set` y le pasa el nuevo valor como un parámetro. Si existe una propiedad con ese nombre concreto, la nueva propiedad lo sobrescribe.

La función “get” es una función que no tiene parámetros. El valor devuelto puede ser de cualquier tipo. El tipo de valor puede cambiar según la invocación. El valor devuelto se trata como el valor actual de la propiedad.

La función “set” es una función que acepta un parámetro: el nuevo valor de la propiedad. Por ejemplo, si la propiedad `x` se asigna mediante la sentencia `x = 1`, la función `set` recibirá el parámetro `1` de tipo número. El valor devuelto por la función “set” se pasa por alto.

Puede agregar propiedades de captador/definidor a los objetos prototipo. Si agrega una propiedad de captador/definidor a un objeto prototipo, todas las instancias del objeto que heredan el objeto prototipo heredarán la propiedad de captador/definidor. Esto hace posible agregar una propiedad de captador/definidor a una ubicación, el objeto prototipo, y aplicarla a todas las instancias de una clase (como si se agregaran métodos a objetos prototipo). Si se invoca una función `get/set` para una propiedad de captador/definidor de un objeto prototipo heredado, la referencia que se pasa a la función `get/set` será el objeto al que se hizo referencia originalmente, no el objeto prototipo.

Si no se invoca correctamente, `Object.addProperty()` puede no funcionar y generar un error. En la tabla siguiente se describen los errores que pueden producirse:

Condición de error	Qué sucede
<i>prop</i> no es un nombre de propiedad válido; por ejemplo, una cadena vacía.	Devuelve <code>false</code> y no se agrega la propiedad.
<i>getFunc</i> no es un objeto de función válido.	Devuelve <code>false</code> y no se agrega la propiedad.
<i>setFunc</i> no es un objeto de función válido.	Devuelve <code>false</code> y no se agrega la propiedad.

## Ejemplo

**Sintaxis 1:** objeto que tiene dos métodos internos, `setQuantity()` y `getQuantity()`. Puede utilizarse una propiedad, `bookcount`, para invocar estos métodos cuando se define o recupera. Un tercer método interno, `getTitle()`, devuelve un valor de sólo lectura asociado a la propiedad `bookname`:

```
function Book() {
    this.setQuantity = function(numBooks) {
        this.books = numBooks;
    }
    this.getQuantity = function() {
        return this.books;
    }
    this.getTitle = function() {
        return "Catcher in the Rye";
    }
    this.addProperty("bookcount", this.getQuantity, this.setQuantity);
    this.addProperty("bookname", this.getTitle, null);
}
myBook = new Book();
myBook.bookcount = 5;
order = "Ha solicitado " + myBook.bookcount + " copias de " + myBook.bookname;
```

Cuando el script recupera el valor de `myBook.bookcount`, el intérprete de ActionScript invoca automáticamente `myBook.getQuantity()`. Cuando un script modifica el valor de `myBook.bookcount`, el intérprete invoca `myObject.setQuantity()`. La propiedad `bookname` no especifica una función set, de modo que los intentos de modificar `bookname` se pasan por alto.

**Sintaxis 2:** el ejemplo anterior de `bookcount` y `bookname` funciona, pero las propiedades `bookcount` y `bookname` se añaden a cada instancia del objeto `Book`. Esto significa que el coste de tener estas propiedades es que haya dos lugares de propiedades para cada instancia del objeto. Si hay muchas propiedades como `bookcount` y `bookname` en una clase, es posible que requieran gran cantidad de memoria. En su lugar, puede añadir las propiedades para `Book.prototype`:

```
function Book () {}
Book.prototype.setQuantity = function(numBooks) {
    this.books = numBooks;
}
Book.prototype.getQuantity = function() {
    return this.books;
}
Book.prototype.getTitle = function() {
    return "Catcher in the Rye";
}
Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
    Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
myBook = new Book();
myBook.bookcount = 5;
order = "Ha solicitado "+myBook.bookcount+" copias de "+myBook.bookname;
```

Ahora las propiedades `bookcount` y `bookname` sólo existen en un lugar: el objeto `Book.prototype`. El resultado, no obstante, es el mismo que el del código de la sintaxis 1, que añade `bookcount` y `bookname` directamente en cada instancia. Si se accede a `bookcount` o `bookname` en una instancia `Book`, la cadena de prototipo asciende y se encuentra la propiedad de captador/definidor en `Book.prototype`.

**Sintaxis 3:** las propiedades incorporadas `TextField.scroll` y `TextField.maxscroll` son propiedades de captador/definidor. El objeto `TextField` tiene los métodos internos `getScroll()`, `setScroll()` y `getMaxScroll()`. El constructor `TextField` crea las propiedades de captador/definidor y las dirige a los métodos internos `get/set`, tal como se muestra a continuación:

```
this.addProperty("scroll", this.getScroll, this.setScroll);
this.addProperty("maxscroll", this.getMaxScroll, null);
```

Cuando un script recupera el valor de `myTextField.scroll`, el intérprete de ActionScript invoca automáticamente `myTextField.getScroll()`. Cuando un script modifica el valor de `myTextField.scroll`, el intérprete invoca `myTextField.setScroll()`. La propiedad `maxscroll` no tiene especificada una función set, de modo que los intentos de modificar `maxscroll` se pasan por alto.

**Sintaxis 4:** aunque las propiedades `TextField.scroll` y `TextField.maxscroll` incorporadas funcionan en el ejemplo de sintaxis 3, las propiedades `scroll` y `maxscroll` se añaden a cada una de las instancias del objeto `TextField`. Esto significa que el coste de tener estas propiedades es que haya dos lugares de propiedades para cada instancia del objeto. Si hay muchas propiedades como `scroll` y `maxscroll` en una clase, es posible que requieran gran cantidad de memoria. En su lugar, puede agregar las propiedades `scroll` y `maxscroll` a `TextField.prototype`:

```
TextField.prototype.addProperty("scroll", this.getScroll, this.setScroll);
TextField.prototype.addProperty("maxscroll", this.getMaxScroll, null);
```

Ahora, las propiedades `scroll` y `maxscroll` sólo existen en un lugar: el objeto `TextField.prototype`. Sin embargo, el efecto es el mismo que el código anterior, en el que las propiedades `scroll` y `maxscroll` se agregaban directamente a cada instancia. Si se accede a `scroll` o `maxscroll` desde una instancia `TextField`, se asciende en la cadena de prototipos y se encuentra la propiedad de captador/definidor de `TextField.prototype`.

## Object.\_\_proto\_\_

### Disponibilidad

Flash Player 5.

### Sintaxis

*myObject*.\_\_proto\_\_

### Parámetros

Ninguno.

### Descripción

Propiedad; se refiere a la propiedad `prototype` de la función constructora que ha creado *myObject*. La propiedad `__proto__` se asigna automáticamente a todos los objetos al crearlos. El intérprete de ActionScript utiliza la propiedad `__proto__` para acceder a la propiedad `prototype` de la función constructor del objeto a fin de averiguar las propiedades y los métodos que el objeto hereda de su clase.

## Object.registerClass()

### Disponibilidad

Flash Player 6. Si está utilizando archivos de clase externos, en lugar de usar este método, puede usar el campo `Class` de ActionScript 2.0, en el cuadro de diálogo Propiedades de vinculación o Propiedades de símbolo, para asociar un objeto a una clase.

### Sintaxis

`Object.registerClass(symbolID, theClass)`

### Parámetros

*symbolID* Identificador de vínculo del símbolo de clip de película o identificador de cadena de la clase de ActionScript.

*theClass* Referencia a la función constructora de la clase de ActionScript, o `null` para no registrar el símbolo.

### Valor devuelto

Si el registro de clase tiene éxito, se devuelve el valor `true`; en caso contrario, se devuelve `false`.

### Descripción

Método; asocia un símbolo de clip de película con una clase de objeto de ActionScript. Si no hay ningún símbolo, Flash crea una asociación entre un identificador de cadena y una clase de objeto.

Cuando la línea de tiempo coloca una instancia del símbolo de clip de película especificado, la instancia se registra en la clase especificada por el parámetro *theClass* en lugar de en la clase *MovieClip*.

Cuando se crea una instancia del símbolo del clip de película mediante *MovieClip.attachMovie()* o *MovieClip.duplicateMovieClip()*, se registra en la clase especificada por *theClass* en lugar de registrarse en la clase *MovieClip*. Si el parámetro *theClass* es *null*, este método elimina cualquier definición de clase de *ActionScript* asociada con el símbolo de clip de película o el identificador de clase especificados. Para los símbolos de clip de película, todas las instancias existentes del clip de película permanecen inalteradas; sin embargo, las nuevas instancias del símbolo se asocian con la clase determinada *MovieClip*.

Si ya se ha registrado un símbolo en una clase, este método lo reemplaza por el nuevo registro.

Cuando la línea de tiempo coloca una instancia de clip de película, o cuando ésta se crea con el método *attachMovie()* o *duplicateMovieClip()*, *ActionScript* invoca el constructor para la clase adecuada con la palabra clave *this* dirigida al objeto. La función constructora se invoca sin parámetros.

Si se utiliza este método para registrar un clip de película con una clase de *ActionScript* que no sea *MovieClip*, el símbolo de clip de película no hereda los métodos, las propiedades ni los eventos de la clase incorporada *MovieClip* a menos que se incluya la clase *MovieClip* en la cadena prototipo de la clase nueva. El código siguiente crea una clase *ActionScript* nueva denominada *theClass* que hereda las propiedades de la clase *MovieClip*:

```
theClass.prototype = new MovieClip();
```

#### Véase también

*MovieClip.attachMovie()*, *MovieClip.duplicateMovieClip()*

## Object.toString()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myObject.toString()
```

### Parámetros

Ninguno.

### Valor devuelto

Una cadena.

### Descripción

Método; convierte el objeto especificado en una cadena y la devuelve.

## Object.unwatch()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myObject.unwatch (prop)
```

### Parámetros

*prop* Nombre de la propiedad de objeto que ya no debe observarse, como una cadena.

### Valor devuelto

Valor booleano.

### Descripción

Método; elimina un punto de observación creado por `Object.watch()`. Este método devuelve el valor `true` si el punto de observación se ha eliminado correctamente; en caso contrario, devuelve el valor `false`.

## Object.valueOf()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myObject.valueOf()
```

### Parámetros

Ninguno.

### Valor devuelto

El valor primitivo del objeto especificado, o el objeto propiamente dicho.

### Descripción

Método; devuelve el valor primitivo del objeto especificado. Si el objeto no tiene un valor primitivo, se devuelve el propio objeto.

## Object.watch()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myObject.watch( prop, callback [, userData] )
```

## Parámetros

*prop* Cadena que indica el nombre de la propiedad de objeto que se debe observar.

*callback* Función que se invoca cuando cambia la propiedad observada. Este parámetro es un objeto de función, no es un nombre de función en formato de cadena. La forma de *callback* es `callback(prop, oldval, newval, userData)`.

*userData* Datos de `ActionScript` arbitrarios que se pasan al método *callback*. Si se omite el parámetro *datosUsuario*, se pasa el valor `undefined` al método *callback*. Este parámetro es opcional.

## Valor devuelto

El valor `true` si el punto de observación se ha creado correctamente; en caso contrario, devuelve el valor `false`.

## Descripción

Método; registra el controlador de eventos que se invoca cuando cambia una propiedad específica de un objeto `ActionScript`. Si la propiedad cambia, el controlador de eventos se invoca con `myObject` como el objeto que la contiene. Debe devolver el nuevo valor del método `Object.watch()` o se asigna el valor `undefined` a la propiedad del objeto observado.

Un punto de observación puede filtrar (o anular) la asignación del valor, mediante la devolución de un `newval` (u `oldval`) modificado. Si elimina una propiedad para la que se ha establecido un punto de observación, dicho punto no desaparece. Si, posteriormente, vuelve a crear dicha propiedad, el punto de observación todavía funcionará. Para eliminar un punto de observación, utilice el método `Object.unwatch`.

Sólo se puede registrar un punto de observación en una propiedad. Las posteriores llamadas a `Object.watch()` en la misma propiedad sustituirán el punto de observación original.

El método `Object.watch()` se comporta de forma parecida a la función `Object.watch()` en Netscape JavaScript 1.2 y posteriores. La principal diferencia se encuentra en el parámetro *userData*; se trata de una adición de Flash a `Object.watch()` que no es compatible con Netscape Navigator. Puede pasar el parámetro *userData* al controlador de eventos y utilizarlo en dicho controlador.

El método `Object.watch()` no puede observar a las propiedades de captador/definidor. Las propiedades de captador/definidor funcionan según un “cálculo diferido”, es decir, el valor de la propiedad no se determina hasta que se consulta la propiedad. A menudo, el “cálculo diferido” es más eficiente puesto que la propiedad no se actualiza constantemente, sino que se calcula su resultado cuando es necesario. Sin embargo, `Object.watch()` necesita calcular el valor de una propiedad para poder activar los puntos de observación establecidos en la misma. Para trabajar con una propiedad de captador/definidor, `Object.watch()` necesita calcular constantemente el resultado la propiedad, lo cual no es eficiente.

Las propiedades predefinidas de `ActionScript`, tales como `_x`, `_y`, `_width` y `_height`, suelen ser propiedades de captador/definidor y, por consiguiente, no pueden observarse mediante `Object.watch()`.

## Ejemplo

En este ejemplo se muestra un componente `CheckBox` con métodos que definen la etiqueta o el valor de cada instancia de casilla de verificación:

```
myCheckBox1.setValue(true);
myCheckBox1.setLabel("nueva etiqueta");
...
```

Es preferible pensar en el valor y la etiqueta de una casilla de verificación como propiedades. Puede utilizar `Object.watch()` para que el acceso al valor y la etiqueta sea parecido al acceso a propiedades en lugar una invocación de método, como se muestra a continuación:

```
// Definir el constructor para (y, con ello, definir) la clase CheckBox
function CheckBox() {
    ...
    this.watch('value', function (id, oldval, newval){
        ...
    });
    this.watch('label', function(id, oldval, newval){
        ...
    });
}
```

Si se modifica la propiedad `value` o `label`, la función especificada por el componente se invoca para que realice las tareas necesarias para actualizar el aspecto y el estado del componente. El ejemplo siguiente invoca un método `Object.watch()` para notificar al componente que la variable ha cambiado, lo que hace que el componente actualice su representación gráfica.

```
myCheckBox1.value = false;
```

Esta sintaxis es más concisa que la sintaxis anterior:

```
myCheckBox1.setValue(false);
```

## Véase también

[Object.addProperty\(\)](#), [Object.unwatch\(\)](#)

# Object()

## Disponibilidad

Flash Player 5 .

## Sintaxis

```
Object( [ value ] )
```

## Parámetros

*value*    Un número, cadena o valor booleano.

## Valor devuelto

Un objeto.



## Descripción

Función de conversión; crea un nuevo objeto, vacío, o convierte el número, cadena, o valor booleano especificado en un objeto. El resultado de este comando es similar a la creación de un objeto mediante el constructor de la clase `Object` (véase [“Constructor para la clase Object” en la página 577](#)).

## on()

### Disponibilidad

Flash 2. En Flash 2 no se admiten todos los eventos.

### Sintaxis

```
on(mouseEvent) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*statement(s)* Instrucciones que se van a ejecutar cuando tiene lugar el *mouseEvent*.

Un *mouseEvent* es un desencadenante denominado “evento”. Cuando el evento tiene lugar, se ejecutan las sentencias que aparecen entre llaves a continuación. Puede especificarse cualquiera de los valores siguientes para el parámetro *mouseEvent*:

- `press` Se presiona el botón del ratón mientras el puntero se encuentra sobre el botón.
- `release` Se suelta el botón del ratón mientras el puntero se encuentra sobre el botón.
- `releaseOutside` Se suelta el botón del ratón mientras el puntero está fuera del botón después de haber presionado el botón mientras el puntero estaba dentro del botón.
- `rollOut` El puntero se desplaza fuera del área del botón.
- `rollOver` El puntero del ratón se desplaza sobre el botón.
- `dragOut` Mientras el puntero se encuentra sobre el botón, se presiona el botón del ratón y después se desplaza fuera del área del botón.
- `dragOver` Mientras el puntero se encontraba sobre el botón, se ha presionado el botón del ratón, se ha desplazado fuera del botón y, a continuación, se ha vuelto a desplazar sobre el botón.
- `keyPress ("key")` Se presiona la tecla especificada. Para la parte "key" del parámetro, especifique un código de tecla o una constante de tecla. Para una lista de los códigos de tecla asociados con las teclas de un teclado estándar, véase el [Apéndice C, “Teclas del teclado y valores de códigos de tecla”, en la página 791](#); para una lista de constantes de tecla, véase [“Resumen de propiedades para la clase Key” en la página 415](#).

### Descripción

Controlador de eventos; especifica el evento de ratón o las teclas que desencadenan una acción.

### Ejemplo

En el script siguiente, la acción `startDrag()` se ejecuta cuando se presiona el ratón y el script condicional se ejecuta cuando se suelta el ratón y el objeto.

```

on(press){
    startDrag("rabbit");
}
on(release) {
    trace(_root.rabbit._y);
    trace(_root.rabbit._x);
    stopDrag();
}

```

**Véase también**

[onClipEvent\(\)](#)

## onClipEvent()

### Disponibilidad

Flash Player 5.

### Sintaxis

```

onClipEvent(movieEvent){
    // las sentencias se escriben aquí
}

```

### Parámetros

Un *movieEvent* es un desencadenante denominado *evento*. Cuando el evento tiene lugar, se ejecutan las sentencias que aparecen entre llaves a continuación. Puede especificarse cualquiera de los valores siguientes para el parámetro *movieEvent*:

- **load** La acción se inicia en cuanto se crea una instancia del clip de película y aparece en la línea de tiempo.
- **unload** La acción se inicia en el primer fotograma después de eliminar el clip de película de la línea de tiempo. Las acciones asociadas con el evento de clip de película **unload** se procesan antes de que se asocien acciones al fotograma afectado.
- **enterFrame** La acción se desencadena de forma continua a la velocidad de los fotogramas del clip de película. Las acciones asociadas con el evento de clip **enterFrame** se procesan antes que cualquiera de las acciones de fotogramas asociadas a los fotogramas afectados.
- **mouseMove** La acción se inicia cada vez que se mueve el ratón. Utilice las propiedades `_xmouse` y `_ymouse` para determinar la posición actual del ratón.
- **mouseDown** La acción se inicia cada vez que se presiona el botón izquierdo del ratón.
- **mouseUp** La acción se inicia cada vez que se suelta el botón izquierdo del ratón.
- **keyDown** La acción se inicia cuando se presiona una tecla. Utilice el método [Key.getCode\(\)](#) para recuperar información sobre la última tecla que se ha presionado.
- **keyUp** La acción se inicia cuando se suelta una tecla. Utilice el método [Key.getCode\(\)](#) para recuperar información sobre la última tecla que se ha presionado.
- **data** La acción se inicia cuando se reciben datos en una acción `loadVariables()` o `loadMovie()`. Cuando se especifica con una acción `loadVariables()`, el evento **data** sólo se produce una vez, cuando se carga la última variable. Cuando se especifica con una acción `loadMovie()`, el evento **data** se produce repetidamente, según se recupera cada sección de datos.

## Descripción

Controlador de eventos; desencadena acciones definidas para una instancia específica de un clip de película.

## Ejemplo

La sentencia siguiente incluye el script de un archivo externo cuando se exporta el archivo SWF; las acciones del script incluido se ejecutan cuando se carga el clip de película al que están asociadas:

```
onClipEvent(load) {  
    #include "myScript.as"  
}
```

En el ejemplo siguiente se utiliza `onClipEvent()` con el evento de película `keyDown`. El evento de película `keyDown` se utiliza habitualmente junto con uno o más métodos y propiedades asociados con el objeto `Key`. En el script siguiente se utiliza `Key.getCode()` para averiguar qué tecla ha presionado el usuario; si la tecla presionada coincide con la propiedad `Key.RIGHT`, la película se envía al fotograma siguiente; si la tecla presionada coincide con la propiedad `Key.LEFT`, la película se envía al fotograma anterior.

```
onClipEvent(keyDown) {  
    if (Key.getCode() == Key.RIGHT) {  
        _parent.nextFrame();  
    } else if (Key.getCode() == Key.LEFT){  
        _parent.prevFrame();  
    }  
}
```

En el ejemplo siguiente se utiliza `onClipEvent()` con el evento de película `mouseMove`. Las propiedades `_xmouse` e `_ymouse` realizan un seguimiento de la posición del ratón cada vez que se mueve.

```
onClipEvent(mouseMove){  
    stageX=_root._xmouse;  
    stageY=_root._ymouse;  
}
```

## Véase también

[Clase Key](#), [MovieClip.\\_xmouse](#), [MovieClip.\\_ymouse](#), [on\(\)](#), [updateAfterEvent\(\)](#)

# onUpdate

## Disponibilidad

Flash Player 6.

## Sintaxis

```
function onUpdate() {  
    ...sentencias...;  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se define `onUpdate` para una película de vista previa dinámica utilizada con un componente. Cuando una instancia de un componente del escenario tiene una película de vista previa dinámica, la herramienta de edición invoca la función `onUpdate` de dicha película siempre que los parámetros de componente de la instancia de componente cambian. La herramienta de edición invoca la función `onUpdate` sin parámetros, y el valor que se devuelve se omite. La función `onUpdate` debe declararse en la línea de tiempo principal de la película de vista previa dinámica.

Definir una función `onUpdate` en una película de vista previa dinámica es algo opcional.

Para más información sobre las películas de vista previa dinámica, consulte *Utilización de componentes*.

## Ejemplo

La función `onUpdate` otorga a la película de vista previa dinámica una oportunidad para actualizar su aspecto visual de modo que coincida con los nuevos valores de los parámetros del componente. Cuando el usuario cambia el valor de un parámetro en el Inspector de propiedades o el panel Parámetros de componentes del componente, se invoca `onUpdate`. La función `onUpdate` llevará a cabo una acción para actualizarse a sí misma. Por ejemplo, si el componente incluye un parámetro `color`, la función `onUpdate` puede alterar el color de un clip de película de la película en la vista previa dinámica para reflejar el nuevo valor del parámetro. Además, puede almacenar el nuevo color en una variable interna.

A continuación, se muestra un ejemplo del uso de la función `onUpdate` para pasar valores de parámetro a través de un clip de película vacío de la película de vista previa dinámica. Suponga que tiene un componente de botón etiquetado con una variable `labelColor`, que especifica el color de la etiqueta de texto. El código siguiente se encuentra en el primer fotograma de la línea de tiempo principal de la película del componente:

```
//Definir la variable del parámetro textColor para especificar el color del  
    texto de la etiqueta del botón.  
buttonLabel.textColor = labelColor;
```

En la película de vista previa dinámica, coloque un clip de película vacío denominado “xch”. A continuación, coloque el código siguiente en el primer fotograma de la película de vista previa dinámica. Añada “xch” a la ruta de la variable `labelColor` para pasar la variable a través del clip de película `my_mc`:

```
//Escribir una función onUpdate, añadiendo "my_mc." a los nombres de variable  
    de parámetro:  
function onUpdate () {  
    buttonLabel.textColor = my_mc.labelColor;  
}
```

## or

### Disponibilidad

Flash 4. Este operador se ha eliminado y se ha sustituido por el operador `||` ([OR lógico](#)).

### Sintaxis

*condition1* o *condition2*

### Parámetros

*condition1,2* Expresión que da como resultado `true` o `false`.

### Valor devuelto

Ninguno.

### Descripción

Operador; calcula el resultado de *condition1* y *condition2* y si cualquiera de las condiciones es `true`, toda la expresión será `true`.

### Véase también

[||](#) ([OR lógico](#)), [|](#) ([OR en modo bit](#))

## ord

### Disponibilidad

Flash Player 4. Esta función se ha eliminado y se ha sustituido por los métodos y las propiedades de la clase `String`.

### Sintaxis

`ord(character)`

### Parámetros

*character* Carácter que se convierte en un número de código ASCII.

### Valor devuelto

Ninguno.

### Descripción

Función de cadena; convierte los caracteres en números de código ASCII.

### Véase también

[Clase String](#)

## \_parent

### Disponibilidad

Flash Player 5.

## Sintaxis

```
_parent.property  
_parent._parent.property
```

## Descripción

Identificador; especifica o devuelve una referencia al clip de película que contiene el objeto o el clip de película actual. El objeto actual es el objeto que contiene el código de ActionScript que hace referencia a `_parent`. Utilice `_parent` para especificar una ruta relativa a los clips de película u objetos que están por encima del clip de película u objeto actual.

## Ejemplo

En el ejemplo siguiente, el clip de película `desk` es un elemento secundario del clip de película `classroom`. Cuando el script siguiente se ejecuta dentro del clip de película `desk`, la cabeza lectora saltará al fotograma 10 de la línea de tiempo del clip de película `classroom`.

```
_parent.gotoAndStop(10);
```

## Véase también

[\\_root](#), [targetPath](#)

# parseFloat()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
parseFloat(string)
```

## Parámetros

*string* Cadena que se lee y se convierte en un número de coma flotante.

## Valor devuelto

Un número o NaN.

## Descripción

Función; convierte una cadena en un número de coma flotante. La función lee, o analiza, y devuelve los números de una cadena, hasta que encuentra un carácter que no forma parte del número inicial. Si la cadena no empieza por un número que puede analizarse, `parseFloat` devuelve NaN. El espacio en blanco que precede a los números enteros válidos se ignora, al igual que los caracteres no numéricos finales.

## Ejemplo

El ejemplo siguiente utiliza la función `parseFloat` para calcular el valor de varios tipos de números.

```
parseFloat("-2") devuelve -2  
parseFloat("2.5") devuelve 2,5  
parseFloat("3.5e6") devuelve 3,5e6, o 3500000  
parseFloat("foobar") devuelve NaN
```

```
parseFloat("5.1") devuelve 5,1  
parseFloat("3.75math") devuelve 3,75  
parseFloat("0garbage") devuelve 0
```

### Véase también

[NaN](#)

## parseInt

### Disponibilidad

Flash Player 5.

### Sintaxis

```
parseInt(expression [, radix])
```

### Parámetros

*expression*    Cadena que se convierte en un número entero.

*radix*    Opcional; número entero que representa la base del número que debe analizarse. Los valores válidos oscilan entre 2 y 36.

### Valor devuelto

Un número NaN.

### Descripción

Función; convierte una cadena en un número entero. Si la cadena especificada en los parámetros no puede convertirse en un número, la función devuelve NaN. Las cadenas que empiezan por 0x se interpretan como números hexadecimales. Los números enteros que empiezan por 0 o que especifican una base 8 se interpretan como números octales. El espacio en blanco que precede a los números enteros válidos se ignora, al igual que los caracteres no numéricos finales.

### Ejemplo

El ejemplo siguiente utiliza la función parseInt para calcular el valor de varios tipos de números.

```
parseInt("3.5")  
// devuelve 3  
  
parseInt("bar")  
// devuelve NaN  
  
parseInt("4foo")  
// devuelve 4
```

A continuación, se muestran ejemplos de conversiones hexadecimales:

```
parseInt("0x3F8")  
// devuelve 1016  
  
parseInt("3E8", 16)  
// devuelve 1000
```

En el ejemplo siguiente se muestra una conversión binaria:

```
parseInt("1010", 2)  
// devuelve 10 (la representación decimal del binario 1010)
```

A continuación se muestran ejemplos de análisis de números octales:

```
parseInt("0777")  
parseInt("777", 8)  
// devuelve 511 (la representación decimal del octal 777)
```

## play()

### Disponibilidad

Flash 2.

### Sintaxis

```
play()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; desplaza la cabeza lectora hacia delante en la línea de tiempo.

### Ejemplo

El código siguiente utiliza la sentencia `if` para comprobar el valor de un nombre que introduce el usuario. Si el usuario introduce `Steve`, se llama a la acción `play()` y la cabeza lectora avanza en la línea de tiempo. Si el usuario introduce un valor distinto a `Steve`, el archivo SWF no se reproduce y aparece un campo de texto con el nombre de variable `alert`.

```
stop();  
if (name == "Steve") {  
    play();  
} else {  
    alert="¡Usted no es Steve!";  
}
```

## prevFrame()

### Disponibilidad

Flash 2.

### Sintaxis

```
prevFrame()
```

### Parámetros

Ninguno.



**Valor devuelto**

Ninguno.

**Descripción**

Función; envía la cabeza lectora al fotograma anterior y la detiene. Si el fotograma actual es 1, la cabeza lectora no se mueve.

**Ejemplo**

Cuando el usuario hace clic en un botón que tiene asociado el controlador siguiente, la cabeza lectora se envía al fotograma anterior.

```
on(release) {  
    prevFrame();  
}
```

**Véase también**

[MovieClip.prevFrame\(\)](#)

## prevScene()

**Disponibilidad**

Flash 2.

**Sintaxis**

```
prevScene()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Función; envía la cabeza lectora al fotograma 1 de la escena anterior y la detiene.

**Véase también**

[nextScene\(\)](#)

## print()

**Disponibilidad**

Flash Player 4.20.

**Nota:** si está editando en Flash Player 7 o una versión posterior, puede crear un objeto PrintJob, que le otorga a usted y al usuario más control sobre el proceso de impresión. Para más información, consulte la entrada [Clase PrintJob](#).

**Sintaxis**

```
print(target, "Bounding box")
```

## Parámetros

*target* Nombre de instancia del clip de película que se va a imprimir. De forma predeterminada, se imprimen todos los fotogramas de la instancia de destino. Si desea imprimir determinados fotogramas del clip de película, asigne una etiqueta de fotograma #p a dichos fotogramas.

*Bounding box* Modificador que define el área de impresión del clip de película. Este parámetro debe especificarse entre comillas y debe incluir uno de los valores siguientes:

- *bmovie* Designa el recuadro de delimitación de un fotograma específico de una película como el área de impresión para todos los fotogramas de la película que se pueden imprimir. Asigne una etiqueta de fotograma #b al fotograma cuyo recuadro de delimitación desee utilizar como área de impresión.
- *bmax* Designa como área de impresión un compuesto de todos los recuadros de delimitación de todos los fotogramas que se pueden imprimir. Especifique *bmax* cuando los fotogramas de la película que se pueden imprimir varían de tamaño.
- *bframe* Indica que debe utilizarse el recuadro de delimitación de cada fotograma que se puede imprimir como área de impresión para ese fotograma. Esto cambia el área de impresión para cada fotograma y modifica la escala de los objetos para que quepan en el área de impresión. Utilice *bframe* si tiene objetos de diferentes tamaños en cada fotograma y desea que cada objeto cubra la página impresa.

## Valor devuelto

Ninguno.

## Descripción

Función; imprime el clip de película de *target* según los límites especificados en el parámetro (*bmovie*, *bmax* o *bframe*). Si desea imprimir determinados fotogramas del clip de película de destino, asocie una etiqueta de fotograma #p a dichos fotogramas. Aunque `print()` da lugar a impresiones de mayor calidad que `printAsBitmap()`, no puede utilizarse para imprimir clips de películas que utilizan transparencias alfa u otros efectos de color especiales.

Si utiliza *bmovie* para el parámetro *Bounding box* pero no asigna una etiqueta #b a un fotograma, el área de impresión quedará determinada por el tamaño del escenario de la película cargada. La película no hereda el tamaño del escenario de la película principal.

Todos los elementos imprimibles de una película deben estar cargados por completo antes de que pueda comenzar la impresión.

La función de impresión de Flash Player admite impresoras PostScript y no PostScript. Las impresoras no PostScript convierten los vectores en mapas de bits.

## Ejemplo

En el ejemplo siguiente se imprimen todos los fotogramas imprimibles del clip de película *my\_mc* con el área de impresión definida por el recuadro de delimitación del fotograma que lleva la etiqueta de fotograma #b:

```
print(my_mc,"bmovie");
```

En el ejemplo siguiente se imprimen todos los fotogramas imprimibles de *my\_mc* con un área de impresión definida por el recuadro de delimitación de cada fotograma:

```
print(my_mc,"bframe");
```

Véase también

[printAsBitmap\(\)](#), [printAsBitmapNum\(\)](#), [Clase PrintJob](#), [printNum\(\)](#)

## printAsBitmap()

### Disponibilidad

Flash Player 4.20.

**Nota:** si está editando en Flash Player 7 o una versión posterior, puede crear un objeto [PrintJob](#), que le otorga a usted y al usuario más control sobre el proceso de impresión. Para más información, consulte la entrada [Clase PrintJob](#).

### Sintaxis

```
printAsBitmap(target, "Bounding box")
```

### Parámetros

*target* Nombre de instancia del clip de película que se va a imprimir. De modo predeterminado, se imprimen todos los fotogramas de la película. Si desea imprimir determinados fotogramas de la película, adjunte una etiqueta de fotograma #*p* a dichos fotogramas.

*Bounding box* Modificador que define el área de impresión de la película. Este parámetro debe especificarse entre comillas y debe incluir uno de los valores siguientes:

- *bmovie* Designa el recuadro de delimitación de un fotograma específico de una película como el área de impresión para todos los fotogramas de la película que se pueden imprimir. Asigne una etiqueta de fotograma #*b* al fotograma cuyo recuadro de delimitación desee utilizar como área de impresión.
- *bmax* Designa como área de impresión un compuesto de todos los recuadros de delimitación de todos los fotogramas que se pueden imprimir. Especifique el parámetro *bmax* cuando los fotogramas de la película que se pueden imprimir varíen de tamaño.
- *bframe* Indica que debe utilizarse el recuadro de delimitación de cada fotograma que se puede imprimir como área de impresión para ese fotograma. Esto cambia el área de impresión para cada fotograma y modifica la escala de los objetos para que quepan en el área de impresión. Utilice *bframe* si tiene objetos de diferentes tamaños en cada fotograma y desea que cada objeto cubra la página impresa.

### Valor devuelto

Ninguno.

### Descripción

Función; imprime el clip de película *target* como un mapa de bits según los límites especificados en el parámetro (*bmovie*, *bmax* o *bframe*). Utilice [printAsBitmap\(\)](#) para imprimir películas que contienen fotogramas con objetos que utilizan transparencias o efectos de color. La acción [printAsBitmap\(\)](#) imprime con la máxima resolución disponible de la impresora para mantener la mayor calidad y definición.

Si la película no contiene transparencias alfa ni efectos de color, Macromedia recomienda utilizar [print\(\)](#) para obtener mejores resultados de calidad.

Si utiliza `bmovie` para el parámetro *Bounding box* pero no asigna una etiqueta `#b` a un fotograma, el área de impresión quedará determinada por el tamaño del escenario de la película cargada. La película no hereda el tamaño del escenario de la película principal.

Todos los elementos imprimibles de una película deben estar cargados por completo antes de que pueda comenzar la impresión.

La función de impresión de Flash Player admite impresoras PostScript y no PostScript. Las impresoras no PostScript convierten los vectores en mapas de bits.

#### Véase también

[print\(\)](#), [printAsBitmapNum\(\)](#), [Clase PrintJob](#), [printNum\(\)](#)

## printAsBitmapNum()

### Disponibilidad

Flash Player 5.

**Nota:** si está editando en Flash Player 7 o una versión posterior, puede crear un objeto `PrintJob`, que le otorga a usted y al usuario más control sobre el proceso de impresión. Para más información, consulte la entrada [Clase PrintJob](#).

### Sintaxis

```
printAsBitmapNum(level, "Bounding box")
```

### Parámetros

*level* Nivel de Flash Player que se va a imprimir. De forma predeterminada, se imprimen todos los fotogramas del nivel. Si desea imprimir determinados fotogramas del nivel, asigne una etiqueta de fotograma `#p` a dichos fotogramas.

*Bounding box* Modificador que define el área de impresión de la película. Este parámetro debe especificarse entre comillas y debe incluir uno de los valores siguientes:

- `bmovie` Designa el recuadro de delimitación de un fotograma específico de una película como el área de impresión para todos los fotogramas de la película que se pueden imprimir. Asigne una etiqueta de fotograma `#b` al fotograma cuyo recuadro de delimitación desee utilizar como área de impresión.
- `bmax` Designa como área de impresión un compuesto de todos los recuadros de delimitación de todos los fotogramas que se pueden imprimir. Especifique el parámetro `bmax` cuando los fotogramas de la película que se pueden imprimir varíen de tamaño.
- `bframe` Indica que debe utilizarse el recuadro de delimitación de cada fotograma que se puede imprimir como área de impresión para ese fotograma. Esto cambia el área de impresión para cada fotograma y modifica la escala de los objetos para que quepan en el área de impresión. Utilice `bframe` si tiene objetos de diferentes tamaños en cada fotograma y desea que cada objeto cubra la página impresa.

### Valor devuelto

Ninguno.

## Descripción

Función; imprime un nivel en Flash Player como un mapa de bits según los límites especificados en el parámetro (*bmovie*, *bmax* o *bframe*). Utilice `printAsBitmapNum()` para imprimir películas que contienen fotogramas con objetos que utilizan transparencias o efectos de color. La acción `printAsBitmapNum()` imprime con la máxima resolución de la impresora para mantener la mayor calidad y definición. Para calcular el tamaño de archivo imprimible de un fotograma designado para impresión como un mapa de bits, multiplique la anchura en píxeles por la altura en píxeles por la resolución de la impresora.

Si la película no contiene transparencias alfa ni otros efectos de color, se recomienda utilizar `printNum()` para obtener resultados de mayor calidad.

Si utiliza *bmovie* para el parámetro *Bounding box* pero no asigna una etiqueta *#b* a un fotograma, el área de impresión quedará determinada por el tamaño del escenario de la película cargada. La película no hereda el tamaño del escenario de la película principal.

Todos los elementos imprimibles de una película deben estar cargados por completo antes de que pueda comenzar la impresión.

La función de impresión de Flash Player admite impresoras PostScript y no PostScript. Las impresoras no PostScript convierten los vectores en mapas de bits.

## Véase también

`print()`, `printAsBitmap()`, [Clase PrintJob](#), `printNum()`

# Clase PrintJob

## Disponibilidad

Flash Player 7.

## Descripción

La clase `PrintJob` permite crear contenido e imprimirlo en una o más páginas. Esta clase, además de ofrecer mejoras para las funciones de impresión disponibles con el método `print()`, permite generar contenido dinámico fuera de pantalla, mostrar a los usuarios un único cuadro de diálogo de impresión e imprimir un documento sin escala con proporciones que se asignan a las de proporciones del contenido. Esta característica es muy útil para generar e imprimir contenido dinámico externo, como el contenido de una base de datos y el texto dinámico.

Además, con las propiedades completadas por `PrintJob.start()`, el documento puede acceder a los valores de configuración de la impresora del usuario (como altura, anchura y orientación de la página) y puede configurar el documento para dar formato dinámicamente al contenido de Flash que sea apropiado para la configuración de la impresora.

## Resumen de métodos para la clase PrintJob

Debe utilizar los métodos de la clase PrintJob en el orden en que se muestran en la siguiente tabla.

Método	Descripción
<code>PrintJob.start()</code>	Muestra los cuadros de diálogo de impresión del sistema operativo e inicia la cola de impresión.
<code>PrintJob.addPage()</code>	Añade una página a la cola de impresión.
<code>PrintJob.send()</code>	Envía las páginas de la cola de impresión a la impresora.

## Constructor para la clase PrintJob

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_pj = new PrintJob()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto PrintJob que puede utilizarse para imprimir una o varias páginas.

Para implementar un trabajo de impresión, debe utilizar los métodos en la secuencia que se indica:

```
// Crear un objeto PrintJob
my_pj = new PrintJob();                                // crear instancias de objeto

// Mostrar el cuadro de diálogo de impresión
my_pj.start();                                         // iniciar el trabajo de impresión

// Agregar el área especificada al trabajo de impresión
// Repetir una vez para cada página que debe imprimirse
my_pj.addPage([params]);                             // enviar páginas a la cola de
    impresión
my_pj.addPage([params]);
my_pj.addPage([params]);
my_pj.addPage([params]);

// Enviar páginas de la cola de impresión a la impresora
my_pj.send();                                         // imprimir páginas

// Limpiar
delete my_pj;                                         // eliminar objetos
```

En su propia implementación de objetos PrintJob, debe comprobar si hay valores devueltos desde `PrintJob.start()` y `PrintJob.addPage()` antes de continuar con la impresión. Consulte los ejemplos de `PrintJob.addPage()`.

No puede crear un objeto `PrintJob` hasta que no se haya desactivado cualquier otro objeto `PrintJob` que haya creado (es decir, que haya completado o abandonado la tarea). Si trata de crear un segundo objeto `PrintJob` (llamando a `new PrintJob()`) mientras el primero esté todavía activo, el segundo objeto `PrintJob` no se creará.

## Ejemplo

Véase `PrintJob.addPage()`.

## Véase también

`PrintJob.addPage()`, `PrintJob.send()`, `PrintJob.start()`

# PrintJob.addPage()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_pj.addPage(target [, printArea] [, options] [, frameNumber])
```

## Parámetros

*target* Nivel o nombre de instancia del clip de película que se va a imprimir. Pase un número para especificar un nivel (por ejemplo, 0 es la película `_root`) o una cadena entrecomillada para especificar el nombre de instancia de un clip de película.

*printArea* Objeto opcional que especifica el área que se va a imprimir, en el formato siguiente:

```
{xMin:topLeft, xMax:topRight, yMin:bottomLeft, yMax:bottomRight}
```

Las coordenadas que se especifican para *printArea* representan píxeles de la pantalla relativos al punto de registro de la película `_root` (si *target* = 0) o del nivel o clip de película especificado por *target*. Debe especificar las cuatro coordenadas. La anchura (`xMax-xMin`) y altura (`yMax-yMin`) deben ser superiores a 0.

Los puntos son unidades de medida de impresión y los píxeles son unidades de medida de pantalla; un punto equivale en tamaño a un píxel. Puede utilizar las siguientes equivalencias para convertir pulgadas o centímetros en twips, píxeles o puntos (un twip es igual a 1/20 de píxel):

- 1 píxel = 1 punto = 1/72 pulgadas = 20 twips
- 1 pulgada = 72 píxeles = 72 puntos = 1440 twips
- 1 cm = 567 twips

**Nota:** si anteriormente ha utilizado `print()`, `printAsBitmap()`, `printAsBitmapNum()` o `printNum()` para imprimir desde Flash, ha utilizado una etiqueta de fotograma `#b` para especificar el área que debe imprimirse. Cuando utilice el método `addPage()`, debe utilizar el parámetro *printArea* para especificar el área de impresión; las etiquetas de fotograma `#b` se pasan por alto.

Si se omite el parámetro *printArea*, o si se pasa incorrectamente, se imprimirá el área completa del escenario de *target*. Si no desea especificar un valor para el parámetro *printArea* pero sí un valor para *options* o *frameNumber*, debe pasar `null` para *printArea*.

*options* Parámetro opcional que especifica si debe realizarse la impresión como vector o como mapa de bits, en el formato siguiente:

```
{printAsBitmap:Boolean}
```

De forma predeterminada, las páginas se imprimen en formato de vector. Para imprimir *target* como mapa de bits, pase `true` para `printAsBitmap`. El valor predeterminado es `false`, que representa una solicitud para la impresión vectorial. Recuerde las sugerencias siguientes a la hora de determinar el valor que debe utilizarse:

- Si el contenido que desea imprimir incluye una imagen de mapa de bits, utilice `{printAsBitmap:true}` para incluir los efectos de transparencia y color que pueda haber.
- Si el contenido no incluye imágenes de mapa de bits, omita este parámetro o utilice `{printAsBitmap:false}` para imprimir el contenido en un formato de vector de mayor calidad.

Si se omite *options* o se pasa de forma incorrecta, se implementa la impresión vectorial. Si no desea especificar un valor para *options* pero sí un valor para *frameNumber*, pase `null` para *options*.

*frameNumber* Número opcional que permite especificar el fotograma que se va a imprimir; tenga en cuenta que no se invocarán los ActionScript del fotograma. Si omite este parámetro, se imprime el fotograma actual del *target*.

**Nota:** si anteriormente ha utilizado `print()`, `printAsBitmap()`, `printAsBitmapNum()` o `printNum()` para imprimir desde Flash, es posible que haya utilizado una etiqueta de fotograma *#p* en varios fotogramas para especificar las páginas que deben imprimirse. Para utilizar `PrintJob.addPage()` para imprimir varios fotogramas, debe especificar un comando `PrintJob.addPage()` para cada fotograma; las etiquetas de fotograma *#p* se pasan por alto. Si desea averiguar cómo se hace esto mediante programación, consulte el ejemplo que aparece más adelante en esta entrada.

## Valor devuelto

El valor booleano `true` si la página se ha enviado correctamente a la cola de impresión, o `false` en caso contrario.

## Descripción

Método; envía el nivel o el clip de película especificado como una única página a la cola de impresión. Antes de utilizar este método, debe utilizar `PrintJob.start()`; después de llamar a `PrintJob.addPage()` una o más veces para un trabajo de impresión, debe utilizar `PrintJob.send()` para enviar las páginas de la cola de impresión a la impresora.

Si este método devuelve `false` (por ejemplo, si no ha llamado a `PrintJob.start()` o un usuario ha cancelado el trabajo de impresión), las llamadas siguientes a `PrintJob.addPage()` generarán un error. Sin embargo, si las llamadas anteriores a `PrintJob.addPage()` han sido correctas, el comando `PrintJob.send()` final enviará a la impresora las páginas que se hayan enviado correctamente a la cola de impresión.

Si ha pasado un valor para *printArea*, las coordenadas *xMin* e *yMin* se asignarán a la esquina superior izquierda (coordenadas 0,0) del área de la página que se puede imprimir; el área que se puede imprimir se determina mediante las propiedades *pageHeight* y *pageWidth* establecidas por `PrintJob.start()`. Puesto que la salida impresa se alinea con la esquina superior izquierda del área de la página que se puede imprimir, la salida impresa se recorta en la parte derecha o inferior si el área definida en *printArea* es mayor que el área de la página que se puede imprimir. Si no ha pasado un valor para *printArea* y el escenario es más grande que el área imprimible, se realiza el mismo tipo de recorte.



Si desea cambiar la escala del clip de película antes de imprimirlo, establezca sus propiedades `MovieClip._xscale` y `MovieClip._yscale` antes de llamar a este método y luego establézcalas de nuevo en sus valores originales. La escala de un clip de película no guarda relación con *printArea*. Es decir, si especifica que va a imprimir un área de 50 x 50 píxeles, se imprimirán 2500 píxeles. Si ha modificado la escala del clip de película, se seguirán imprimiendo 2500 píxeles, pero con el tamaño de la escala actual.

La función de impresión de Flash Player admite impresoras PostScript y no PostScript. Las impresoras no PostScript convierten los vectores en mapas de bits.

## Ejemplo

En el ejemplo siguiente se muestran distintas formas de emitir el comando `addPage()`.

```
my_btn.onRelease = function()
{
    var pageCount = 0;

    var my_pj = new PrintJob();

    if (my_pj.start())
    {
        // Imprimir todo el fotograma actual de la película _root en formato
        vectorial
        if (my_pj.addPage(0))
        {
            pageCount++;

            // Empezando en 0,0, imprimir un área de 400 píxeles de anchura y 500
            píxeles de altura
            // del fotograma actual de la película _root en formato vectorial
            if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500}))
            {
                pageCount++;

                // Empezando en 0,0, imprimir un área de 400 píxeles de anchura y 500
                píxeles de altura
                // del fotograma 1 de la película _root en formato de mapa de bits
                if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500},
                    {printAsBitmap:true}, 1))
                {
                    pageCount++;

                    // Empezando en 50 píxeles a la derecha de 0,0 y 70 hacia abajo,
                    // imprimir un área de 500 píxeles de anchura y 600 píxeles de
                    altura
                    // del fotograma 4 del nivel 5 en formato vectorial
                    if (my_pj.addPage(5, {xMin:50,xMax:550,yMin:70,yMax:670},null, 4))
                    {
                        pageCount++;

                        // Empezando en 0,0, imprimir un área de 400 píxeles de anchura
                        // y 400 píxeles de altura del fotograma 3 del clip de película
                        "dance_mc"
                        // en formato de mapa de bits
                        if (my_pj.addPage("dance_mc",
                            {xMin:0,xMax:400,yMin:0,yMax:400},{printAsBitmap:true}, 3))
                        {
                            pageCount++;
```



Véase también

`PrintJob.addPage()`, `PrintJob.start()`

## PrintJob.start()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_pj.start()
```

### Parámetros

Ninguno.

### Valor devuelto

Un valor booleano `true` si el usuario hace clic en Aceptar cuando aparece el cuadro de diálogo de impresión o `false` si el usuario hace clic en Cancelar o se produce un error.

### Descripción

Método; muestra los cuadros de diálogo del sistema operativo e inicia la cola de impresión. Los cuadros de diálogo de impresión ofrecen al usuario la oportunidad de cambiar los valores de impresión y, a continuación, completar las siguientes propiedades de sólo lectura (recuerde que 1 punto equivale a 1 píxel en pantalla):

Propiedad	Tipo	Unidades	Notas
<code>PrintJob.paperHeight</code>	Número	Puntos	Altura total del papel
<code>PrintJob.paperWidth</code>	Número	Puntos	Anchura total del papel
<code>PrintJob.pageHeight</code>	Número	Puntos	Altura del área real de la página que se puede imprimir; los márgenes definidos por el usuario se pasan por alto.
<code>PrintJob.pageWidth</code>	Número	Puntos	Anchura del área real de la página que se puede imprimir; los márgenes definidos por el usuario se pasan por alto.
<code>PrintJob.orientation</code>	Cadena	No disponible	"Vertical" u "Horizontal"

Después de que el usuario haga clic en Aceptar en el cuadro de diálogo de impresión, el reproductor empieza a enviar un trabajo de impresión de la cola al sistema operativo. Emita los comandos de `ActionScript` que puedan afectar a la salida impresa, y, a continuación, puede empezar a utilizar comandos `PrintJob.addPage()` para comenzar a enviar páginas a la cola de impresión. Si lo desea, utilice las propiedades de altura, anchura y orientación que devuelve este método para determinar cómo debe ser el formato de la salida impresa.

Puesto que el usuario ve información del tipo "Imprimiendo página 1" justo después de hacer clic en Aceptar, debe llamar a los comandos `PrintJob.addPage()` y `PrintJob.send()` tan pronto como sea posible.

Si este método devuelve `false` (por ejemplo, si el usuario hace clic en Cancelar en lugar de hacer clic en Aceptar), las llamadas siguientes a `PrintJob.addPage()` y `PrintJob.send()` no podrán ejecutarse. Sin embargo, si comprueba este valor devuelto y, como resultado, no envía comandos `PrintJob.addPage()`, también deberá eliminar el objeto `PrintJob` para asegurarse de que la cola de impresión está vacía, como se muestra a continuación.

```
var my_pj = new PrintJob();
var myResult = my_pj.start();
if(myResult){
    // aquí sentencias addPage() y send()
}
delete my_pj;
```

### Ejemplo

Véase `PrintJob.addPage()`.

### Véase también

`PrintJob.addPage()`, `PrintJob.send()`

## printNum()

### Disponibilidad

Flash Player 5.

**Nota:** si está editando en Flash Player 7 o una versión posterior, puede crear un objeto `PrintJob`, que le otorga a usted y al usuario más control sobre el proceso de impresión. Para más información, consulte la entrada [Clase PrintJob](#).

### Sintaxis

```
printNum (level, "Bounding box")
```

### Parámetros

**level** Nivel de Flash Player que se va a imprimir. De forma predeterminada, se imprimen todos los fotogramas del nivel. Si desea imprimir determinados fotogramas del nivel, asigne una etiqueta de fotograma `#p` a dichos fotogramas.

**Bounding box** Modificador que define el área de impresión de la película. Este parámetro debe especificarse entre comillas y debe incluir uno de los valores siguientes:

- **bmovie** Designa el recuadro de delimitación de un fotograma específico de una película como el área de impresión para todos los fotogramas de la película que se pueden imprimir. Asigne una etiqueta de fotograma `#b` al fotograma cuyo recuadro de delimitación desee utilizar como área de impresión.
- **bmax** Designa como área de impresión un compuesto de todos los recuadros de delimitación de todos los fotogramas que se pueden imprimir. Especifique el parámetro `bmax` cuando los fotogramas de la película que se pueden imprimir varíen de tamaño.
- **bframe** Indica que debe utilizarse el recuadro de delimitación de cada fotograma que se puede imprimir como área de impresión para ese fotograma. Esto cambia el área de impresión para cada fotograma y modifica la escala de los objetos para que quepan en el área de impresión. Utilice `bframe` si tiene objetos de diferentes tamaños en cada fotograma y desea que cada objeto cubra la página impresa.

## Valor devuelto

Ninguno.

## Descripción

Función; imprime el nivel de Flash Player de acuerdo con los límites especificados en el parámetro *Bounding box* ("bmovie", "bmax", "bframe"). Si desea imprimir determinados fotogramas de la película de destino, adjunte una etiqueta de fotograma #P a dichos fotogramas. Aun cuando utilizar `printNum()` da como resultado una calidad de impresión superior a la que se obtiene con `printAsBitmapNum()`, no puede utilizar `printNum()` para imprimir películas que contengan transparencias alfa o efectos de color especiales.

Si utiliza *bmovie* para el parámetro *Bounding box* pero no asigna una etiqueta #b a un fotograma, el área de impresión quedará determinada por el tamaño del escenario de la película cargada. La película no hereda el tamaño del escenario de la película principal.

Todos los elementos imprimibles de una película deben estar cargados por completo antes de que pueda comenzar la impresión.

La función de impresión de Flash Player admite impresoras PostScript y no PostScript. Las impresoras no PostScript convierten los vectores en mapas de bits.

## Véase también

[print\(\)](#), [printAsBitmap\(\)](#), [printAsBitmapNum\(\)](#), [Clase PrintJob](#)

# private

## Disponibilidad

Flash Player 6.

## Sintaxis

```
class someClassName{
    private var name;
    private function name() {
        // las sentencias se escriben aquí
    }
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Parámetros

*name* El nombre de la variable o función que desea definir como privada.

## Descripción

Palabra clave; especifica que una variable o función sólo está disponible para la clase que la declara o define, o para subclases de ésta. De manera predeterminada, una variable o función está disponible para cualquier clase que la llame. Utilice esta palabra clave cuando desee restringir el acceso a una variable o función. Para más información, consulte [“Control del acceso de miembros” en la página 170](#).

Puede utilizar esta palabra clave únicamente en definiciones de clase, no en definiciones de interfaz.

#### Véase también

`public`, `static`

## public

Flash Player 6.

#### Sintaxis

```
class someClassName{
    public var name;
    public function name() {
        // las sentencias se escriben aquí
    }
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

#### Parámetros

*name* El nombre de la variable o función que desea definir como pública.

#### Descripción

Palabra clave; especifica que una variable o función está disponible para cualquier clase que la llame. Dado que toda variable o función es pública de manera predeterminada, esta palabra clave se usa principalmente por razones de estilo. Por ejemplo, para reforzar la consistencia de un bloque de código que también contenga variables privadas o estáticas.

#### Ejemplo

Los dos bloques de código siguientes son idénticos desde el punto de vista de funcionamiento.

```
private var age:Number;
public var name:String;
static var birth>Date;
```

```
private var age:Number;
var name:String;
static var birth>Date;
```

Para más información, consulte [“Control del acceso de miembros” en la página 170](#).

#### Véase también

`private`, `static`

## \_quality

#### Disponibilidad

Flash Player 5.

#### Sintaxis

`_quality`

## Descripción

Propiedad (global); establece o recupera la calidad de representación utilizada para una película. Las fuentes de dispositivo siempre son dentadas, de modo que no se ven afectadas por la propiedad `_quality`.

La propiedad `_quality` puede definirse en los valores siguientes:

- "LOW" Calidad de representación baja. No se suavizan ni los gráficos ni los mapas de bits.
- "MEDIUM" Calidad de representación media. Los gráficos se suavizan con una cuadrícula de 2x2, en píxeles, pero los mapas de bits no se suavizan. Apropiado para películas que no contienen texto.
- "HIGH" Calidad de representación alta. Los gráficos se suavizan con una cuadrícula de 4x4, en píxeles, y los mapas de bits se suavizan si la película es estática. Ésta es la configuración de la calidad de representación predeterminada utilizada por Flash.
- "BEST" Calidad de representación muy alta. Los gráficos se suavizan con una cuadrícula de 4x4, en píxeles, y los mapas de bits se suavizan siempre.

## Ejemplo

En el ejemplo siguiente, se define la calidad de representación en LOW:

```
_quality = "LOW";
```

## Véase también

[\\_highquality](#), [toggleHighQuality\(\)](#)

# random

## Disponibilidad

Flash Player 4. Esta función se eliminó en Flash 5 y se sustituyó por [Math.random\(\)](#).

## Sintaxis

```
random(value)
```

## Parámetros

*value* Número entero.

## Valor devuelto

Un número entero.

## Descripción

Función; devuelve un entero aleatorio entre 0 y un número menos que el entero especificado en el parámetro *value*.

## Ejemplo

La utilización de la función `random()` que se muestra a continuación devuelve el valor 0, 1, 2, 3 ó 4.

```
random(5);
```

Véase también

`Math.random()`

## removeMovieClip()

### Disponibilidad

Flash Player 4.

### Sintaxis

```
removeMovieClip(target)
```

### Parámetros

*target* Ruta de destino de una instancia de clip de película creada con `duplicateMovieClip()` o nombre de la instancia de un clip de película creado con `MovieClip.attachMovie()` o `MovieClip.duplicateMovieClip()`.

### Valor devuelto

Ninguno.

### Descripción

Función; elimina el clip de película especificado.

### Véase también

`duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, `MovieClip.attachMovie()`, `MovieClip.removeMovieClip()`

## return

### Disponibilidad

Flash Player 5.

### Sintaxis

```
return[expression]
```

### Parámetros

*expression* Número, cadena, matriz u objeto cuyo valor se debe calcular y devolver como valor de la función. Este parámetro es opcional.

### Valor devuelto

El parámetro *expression* cuyo resultado se ha calculado, si se proporciona.

### Descripción

Sentencia; especifica el valor devuelto por una función. La acción `return` calcula el valor del parámetro *expression* y devuelve el resultado como valor de la función en que se ejecuta. La acción `return` hace que la función deje de ejecutarse y la reemplaza por el valor devuelto. Si la sentencia `return` se utiliza sola, devuelve `null`.



No puede devolver valores múltiples. Si trata de hacerlo, sólo se devolverá el último valor. En el ejemplo siguiente, se devuelve c:

```
return a, b, c ;
```

### Ejemplo

En el ejemplo siguiente se utiliza la acción `return` dentro del cuerpo de la función `sum()` para devolver el valor agregado de los tres parámetros. La línea siguiente de código llama a `sum()` y asigna el valor devuelto a la variable `newValue`.

```
function sum(a, b, c){  
    return a + b + c;  
}  
newValue = sum(4, 32, 78);  
trace(newValue);  
// envía 114 al panel Salida
```

### Véase también

[function](#)

## \_root

### Disponibilidad

Flash Player 5.

### Sintaxis

```
_root.movieClip  
_root.action  
_root.property
```

### Parámetros

*movieClip* Nombre de instancia de un clip de película.

*action* Acción o método.

*property* Propiedad del objeto MovieClip.

### Descripción

Propiedad; especifica o devuelve una referencia a la línea de tiempo de la película raíz. Si una película tiene varios niveles, la línea de tiempo de la película raíz está en el nivel que contiene el script que se está ejecutando. Por ejemplo, si un script del nivel 1 calcula el valor de `_root`, se devuelve `_level1`.

Especificar `_root` es lo mismo que utilizar la notación con barras (/) para especificar una ruta absoluta dentro del nivel actual.

**Nota:** si se carga una película que contiene `_root` en otra película, `_root` hace referencia a la línea de tiempo de la película que se carga, no a la línea de tiempo que contiene `_root`. Para garantizar que `_root` hace referencia a la línea de tiempo de la película cargada aun cuando se cargue en otra película, utilice `MovieClip._lockroot`.

### Ejemplo

El ejemplo siguiente detiene la línea de tiempo del nivel que contiene el script que se está ejecutando:

```
_root.stop();
```

El ejemplo siguiente envía la línea de tiempo del nivel actual al fotograma 3:

```
_root.gotoAndStop(3);
```

### Véase también

`MovieClip._lockroot`, `_parent`, `targetPath`

## scroll

### Disponibilidad

Flash Player 4.

### Sintaxis

```
textFieldVariableName.scroll = x
```

### Descripción

Propiedad; propiedad desestimada que controla la visualización de la información en un campo de texto asociado con una variable. La propiedad `scroll` define dónde comienza a visualizarse el contenido en el campo de texto; una vez que se ha establecido esta propiedad, Flash Player la actualiza a medida que el usuario se desplaza por el campo de texto. La propiedad `scroll` es útil para dirigir a los usuarios a un párrafo específico en un pasaje largo, o para crear campos de texto con desplazamiento. Esta propiedad puede recuperarse y modificarse.

### Ejemplo

El código siguiente está asociado a un botón Arriba que desplaza el contenido del campo de texto `myText`:

```
on(release) {  
    myText.scroll = myText.scroll + 1;  
}
```

### Véase también

`TextField.maxscroll`, `TextField.scroll`

## Clase Selection

### Disponibilidad

Flash Player 5.

### Descripción

La clase `Selection` permite definir y controlar el campo de texto en el que se encuentra el punto de inserción, es decir, el campo que está seleccionado. Los índices de espacio de selección tienen base cero (por ejemplo, la primera posición es 0, la segunda 1, etc.).

No existe ninguna función de constructor para la clase `Selection`, porque sólo puede haber un campo seleccionado a la vez.

## Resumen de métodos para la clase Selection

Método	Descripción
<code>Selection.addListener()</code>	Registra un objeto para recibir una notificación al invocar <code>onSetFocus</code> .
<code>Selection.getBeginIndex()</code>	Devuelve el índice al principio del espacio de selección. Devuelve -1 si no hay índice o campo seleccionado actualmente.
<code>Selection.getCaretIndex()</code>	Devuelve la posición de intercalación (punto de inserción) actual en el espacio de selección actualmente seleccionado. Devuelve -1 si no hay posición de intercalación o espacio de selección seleccionado.
<code>Selection.getEndIndex()</code>	Devuelve el índice al final del espacio de selección. Devuelve -1 si no hay índice o campo seleccionado actualmente.
<code>Selection.getFocus()</code>	Devuelve el nombre de la variable para el campo de texto seleccionado. Devuelve el valor <code>null</code> si no hay ningún campo de texto seleccionado.
<code>Selection.removeListener()</code>	Elimina un objeto registrado con <code>addListener()</code> .
<code>Selection.setFocus()</code>	Selecciona el campo de texto siguiente asociado con la variable especificada.
<code>Selection.setSelection()</code>	Establece los índices de inicio y fin del espacio de selección.

## Resumen de detectores de la clase Selection

Detector	Descripción
<code>Selection.onSetFocus</code>	Recibe notificación cuando cambia la selección de entrada.

## Selection.addListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Selection.addListener(newListener)
```

### Parámetros

*newListener*    Objeto con un método `onSetFocus`.

### Valor devuelto

Ninguno.

### Descripción

Método; registra un objeto para que reciba notificaciones de cambio de selección de teclado. Cuando el elemento seleccionado cambia (por ejemplo, cuando se invoca `Selection.setFocus()`), se invoca el método `onSetFocus` de todos los objetos detectores registrados con `addListener()`. Varios objetos pueden detectar notificaciones de cambio de la selección. Si el detector *newListener* ya está registrado, no se produce ningún cambio.

## Selection.getBeginIndex()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Selection.getBeginIndex()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el índice al principio del espacio de selección. Si no existe ningún índice o no hay ningún campo de texto seleccionado, el método devuelve -1. Los índices de espacio de selección tienen base cero (por ejemplo, la primera posición es 0, la segunda 1, etc.).

## Selection.getCaretIndex()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Selection.getCaretIndex()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el índice de la posición del punto de inserción (carácter de intercalación) intermitente. Si no se visualiza ningún punto de inserción intermitente, el método devuelve -1. Los índices de espacio de selección tienen base cero (por ejemplo, la primera posición es 0, la segunda 1, etc.).

## Selection.getEndIndex()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
Selection.getEndIndex()
```

### Parámetros

Ninguno.

**Valor devuelto**

Un número entero.

**Descripción**

Método; devuelve el índice al final del espacio de selección actualmente seleccionado. Si no existe ningún índice o no hay ningún espacio de selección seleccionado, el método devuelve -1. Los índices de espacio de selección tienen base cero (por ejemplo, la primera posición es 0, la segunda 1, etc.).

## Selection.setFocus()

**Disponibilidad**

Flash Player 5. Los nombres de instancia para los botones y los campos de texto funcionan en Flash Player 6 y versiones posteriores.

**Sintaxis**

```
Selection.setFocus()
```

**Parámetros**

Ninguno.

**Valor devuelto**

Una cadena o `null`.

**Descripción**

Método; devuelve el nombre de variable del campo de texto que está seleccionado. Si no hay ningún campo de texto seleccionado, el método devuelve el valor `null`. Si el elemento seleccionado actualmente es un botón, y el botón es un objeto `Button`, `getFocus()` devuelve la ruta de destino como cadena. Si el elemento seleccionado actualmente es un campo de texto, y el campo de texto es un objeto `TextField`, `getFocus()` devuelve la ruta de destino como cadena.

Si el botón seleccionado actualmente es un botón de clip de película, `Selection.setFocus()` devuelve la ruta de destino del botón de clip de película. Si el elemento seleccionado es un campo de texto con un nombre de instancia, `Selection.setFocus()` devuelve la ruta de destino del objeto `TextField`. De lo contrario, devuelve el nombre de la variable de campo de texto.

## Selection.onSetFocus

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
someListener.onSetFocus = function(oldFocus, newFocus){  
    statements;  
}
```

## Descripción

Detector; recibe notificación cuando cambia la selección de entrada. Para utilizar `onSetFocus`, debe crear un objeto detector. A continuación, puede definir una función para `onSetFocus` y utilizar el método `addListener()` para registrar el detector con el objeto `Selection`, como en el caso siguiente:

```
someListener = new Object();
someListener.onSetFocus = function () { ... };
Selection.addListener(someListener);
```

Los detectores permiten que varios fragmentos de código cooperen, ya que varios detectores pueden recibir notificaciones sobre un mismo evento.

## Véase también

[Selection.addListener\(\)](#)

# Selection.removeListener()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
Selection.removeListener(listener)
```

## Parámetros

*listener* Objeto que ya no recibirá notificaciones sobre el elemento seleccionado.

## Valor devuelto

Si el *listener* se ha eliminado correctamente, el método devuelve un valor `true`. Si el *listener* no se ha eliminado correctamente, por ejemplo, si el *listener* no se encontraba en la lista de detectores del objeto `Selection`, el método devuelve el valor `false`.

## Descripción

Método; elimina un objeto previamente registrado con `addListener()`.

# Selection.setFocus()

## Disponibilidad

Flash Player 5. Los nombres de instancia para botones y clips de película sólo funcionan en Flash Player 6 y versiones posteriores.

## Sintaxis

```
Selection.setFocus("instanceName")
```

## Parámetros

*instanceName* Cadena que especifica la ruta del nombre de instancia de un botón, un clip de película o un campo de texto.

## Valor devuelto

Un evento.

## Descripción

Método; activa el campo de texto, botón o clip de película seleccionable (editable) especificado por *instanceName*. El parámetro *instanceName* debe ser un literal de cadena de la ruta a la instancia. Para especificar la ruta, puede utilizar notación con puntos o con barras. Además, puede utilizar una ruta relativa o absoluta. Si utiliza ActionScript 2.0, debe utilizar la notación con puntos.

Si se pasa el valor `null`, el elemento seleccionado se elimina.

## Ejemplo

El ejemplo siguiente selecciona un campo de texto asociado con `myVar` en la línea de tiempo principal. El parámetro *instanceName* es una ruta absoluta, de modo que puede llamar a la acción desde cualquier línea de tiempo.

```
Selection.setFocus("_root.myVar");
```

En el ejemplo siguiente, el campo de texto asociado con `myVar` se encuentra en un clip de película llamado `myClip` en la línea de tiempo principal. Para establecer el elemento seleccionado, puede utilizar cualquiera de las dos rutas siguientes; la primera es relativa, y la segunda, absoluta.

```
Selection.setFocus("myClip.myVar");  
Selection.setFocus("_root.myClip.myVar");
```

# Selection.setSelection()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
Selection.setSelection(start, end)
```

## Parámetros

*start*    Índice de inicio del espacio de selección.

*end*    Índice de final del espacio de selección.

## Valor devuelto

Ninguno.

## Descripción

Método; establece el espacio de selección del campo de texto seleccionado actualmente. El nuevo espacio de selección comenzará en el índice especificado en el parámetro *start* y finalizará en el índice especificado en el parámetro *end*. Los índices de espacio de selección tienen base cero (por ejemplo, la primera posición es 0, la segunda 1, etc.). Este método no tiene efecto si no existe ningún campo de texto seleccionado actualmente.

# set

## Disponibilidad

Flash Player 6.

## Sintaxis

```
function set property(varName) {  
    // las sentencias se escriben aquí  
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Parámetros

*property* Palabra que desee utilizar para hacer referencia a la propiedad a la que va a acceder `set`; este valor debe ser el mismo que el utilizado en el comando `get` correspondiente.

*varName* La variable local que establece el valor que se asigna.

## Valor devuelto

Ninguno.

## Descripción

Palabra clave; permite "establecer" de manera implícita propiedades asociadas a objetos basados en clases que ha definido en archivos de clase externos. El uso de métodos `set` implícitos permite acceder a las propiedades de los objetos sin acceder directamente a los objetos. Los métodos `get/set` implícitos son la abreviatura sintáctica del método `Object.addProperty()` en ActionScript 1.

Para más información, consulte [“Métodos `get/set` implícitos” en la página 179](#).

## Véase también

[get](#), [Object.addProperty\(\)](#)

# set variable

## Disponibilidad

Flash Player 4.

## Sintaxis

```
set(variable, expression)
```

## Parámetros

*variable* Identificador que alberga el valor del parámetro *expression*.

*expression* Valor asignado a la variable.

## Valor devuelto

Ninguno.

## Descripción

Sentencia; asigna un valor a una variable. Una *variable* es un contenedor que almacena información. El contenedor en sí es siempre el mismo, pero el contenido puede cambiar. La modificación del valor de una variable a medida que se reproduce el archivo SWF permite registrar y guardar información sobre las acciones del usuario, registrar valores que se modifican conforme se reproduce el archivo SWF o comprobar si una determinada condición es `true` o `false`.



Las variables pueden albergar cualquier tipo de datos (por ejemplo, cadenas, números, valores booleanos, objetos o clips de película). La línea de tiempo de cada archivo SWF y cada clip de película tiene su propio conjunto de variables, y cada variable tiene su propio valor al margen de las variables de otras líneas de tiempo.

No se puede utilizar strict data typing en una sentencia `set`. Si usa esta sentencia para establecer un valor para una variable de un tipo de datos diferente al asociado con la variable del archivo de clase, no se generarán errores de compilación.

### Ejemplo

Este ejemplo establece una variable llamada `orig_x_pos` que almacena la posición original del eje `x` del clip de película `ship` para poder restablecer `ship` a su posición de inicio más adelante en el archivo SWF.

```
on(release) {  
    set("orig_x_pos", getProperty ("ship", _x ));  
}
```

El código anterior produce el mismo resultado que el código siguiente:

```
on(release) {  
    orig_x_pos = ship._x;  
}
```

### Véase también

`var`, `call()`

## setInterval()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
setInterval(functionName, interval [, param1, param2, ..., paramN])
```

### Parámetros

*functionName* Nombre de función o referencia a una función anónima.

*interval* Tiempo entre llamada y llamada al parámetro *functionName*, expresado en milisegundos.

*param1*, *param2*, ..., *paramN* Parámetros opcionales que se pasan al parámetro *function* o *methodName*.

### Valor devuelto

Identificador de intervalo que se puede pasar a `clearInterval()` para cancelar el intervalo.

### Descripción

Función; llama a una función, un método o un objeto a intervalos regulares mientras se reproduce un archivo SWF. Puede utilizar una función de intervalo para actualizar variables desde una base de datos o actualizar una visualización de tiempo.

Si el valor de *interval* es menor que la velocidad de fotogramas del archivo SWF (por ejemplo, 10 fotogramas por segundo [fps] equivalen a 100 milisegundos), se llama a la función de intervalo tan cerca del valor de *interval* como sea posible. Debe utilizar la función `updateAfterEvent()` para asegurarse de que la pantalla se actualiza con suficiente frecuencia. Si el valor de *interval* es mayor que la velocidad de fotogramas del archivo SWF, sólo se llama a la función de intervalo cada vez que la cabeza lectora accede a un fotograma para minimizar el impacto cuando se actualiza la pantalla.

### Ejemplo

Sintaxis 1: en el ejemplo siguiente se llama a una función anónima cada 1.000 milisegundos (cada 1 segundo).

```
setInterval( function(){ trace("intervalo llamado"); }, 1000 );
```

Sintaxis 2: en el ejemplo siguiente se definen dos controladores de eventos y se emite una llamada a cada uno de ellos. Ambas llamadas a `setInterval()` envían la cadena "intervalo llamado" al panel Salida cada 1.000 milisegundos. La primera llamada a `setInterval()` llama a la función `callback1()`, que contiene una acción `trace()`. La segunda llamada a `setInterval()` pasa la cadena "intervalo llamado" a la función `callback2()` como parámetro.

```
function callback1() {  
    trace("intervalo llamado");  
}
```

```
function callback2(arg) {  
    trace(arg);  
}
```

```
setInterval( callback1, 1000 );  
setInterval( callback2, 1000, "intervalo llamado" );
```

Sintaxis 3: en este ejemplo se utiliza un método de un objeto. Si desea llamar a un método definido para un objeto, debe utilizar esta sintaxis.

```
obj = new Object();  
obj.interval = function() {  
    trace("se ha llamado a la función de intervalo");  
}
```

```
setInterval( obj, "interval", 1000 );
```

```
obj2 = new Object();  
obj2.interval = function(s) {  
    trace(s);  
}  
setInterval( obj2, "interval", 1000, "se ha llamado a la función de intervalo"  
);
```

Para llamar a un método de un objeto, debe utilizar la segunda forma de la sintaxis de `setInterval()`, tal como se indica a continuación:

```
setInterval( obj2, "interval", 1000, "se ha llamado a la función de intervalo"  
);
```

### Véase también

`clearInterval()`, `updateAfterEvent()`

## setProperty()

### Disponibilidad

Flash Player 4.

### Sintaxis

```
setProperty(target, property, value/expression)
```

### Parámetros

*target* Ruta al nombre de instancia del clip de película cuya propiedad debe establecerse.

*property* Propiedad que debe establecerse.

*value* Nuevo valor literal de la propiedad.

*expression* Ecuación que da como resultado el nuevo valor de la propiedad.

### Valor devuelto

Ninguno.

### Descripción

Función; cambia un valor de propiedad de un clip de película a medida que se reproduce la película.

### Ejemplo

Esta sentencia establece la propiedad `_alpha` de un clip de película llamado `star` en el 30% cuando se hace clic en el botón.

```
on(release) {  
    setProperty("star", _alpha, "30");  
}
```

### Véase también

[getProperty](#)

## Clase SharedObject

### Disponibilidad

Flash Player 6.

### Descripción

Los objetos compartidos presentan muchas posibilidades: ofrecen la posibilidad de compartir datos en tiempo real entre objetos que son persistentes en el equipo del usuario. Los objetos locales compartidos pueden considerarse como “cookies”.

Puede utilizarlos para mantener la constancia local. Ésta es la forma más simple de utilizar un objeto compartido. Por ejemplo, puede llamar a `SharedObject.getLocal()` para crear un objeto compartido, como una calculadora con memoria, en el reproductor. Puesto que el objeto compartido es persistente localmente, Flash guarda sus atributos de datos en el equipo del usuario cuando finaliza el archivo SWF. La próxima vez que se ejecute el archivo SWF, la calculadora contendrá los valores que tenía al finalizar el archivo SWF. Como alternativa, si ha establecido las propiedades del objeto compartido en `null` antes de que finalice el archivo SWF, la próxima vez que se ejecute el archivo SWF la calculadora se abrirá sin ningún valor.

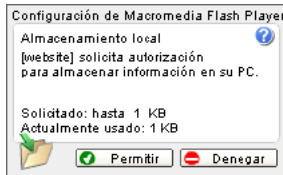
Para crear un objeto local compartido, utilice la sintaxis siguiente:

```
// Crear un objeto local compartido  
so = SharedObject.getLocal("foo");
```

## Consideraciones sobre el espacio en el disco local

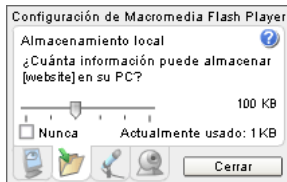
Los objetos locales compartidos siempre son persistentes en el cliente, siempre que haya memoria y espacio en disco disponibles.

De forma predeterminada, Flash puede guardar localmente objetos remotos compartidos y persistentes de hasta 100 K de tamaño. Cuando se intenta guardar un objeto de mayor tamaño, Flash Player muestra el cuadro de diálogo Almacenamiento local, desde el que el usuario puede permitir o denegar el almacenamiento local al dominio que está solicitando el acceso. Asegúrese de que establece un tamaño mínimo de 215 x 138 píxeles; éste es el tamaño mínimo que Flash necesita para visualizar el cuadro de diálogo.



Si el usuario hace clic en Permitir, el objeto se guarda y se invoca `SharedObject.onStatus` con una propiedad `code` de `SharedObject.Flush.Success`; si el usuario hace clic en Denegar, el objeto no se guarda y se invoca `SharedObject.onStatus` con una propiedad `code` de `SharedObject.Flush.Failed`.

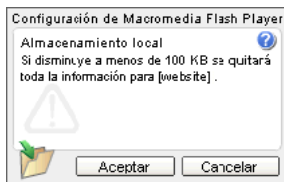
El usuario también puede especificar la configuración de almacenamiento local permanente para un dominio determinado; para ello, debe hacer clic con el botón derecho del ratón (Windows) o presionar la tecla Control y hacer clic (Macintosh) durante la reproducción de un archivo SWF, seleccionar Configuración y abrir el panel Almacenamiento local.



En la lista siguiente se indica de forma resumida la interacción entre las opciones de espacio en disco del usuario y los objetos compartidos:

- Si el usuario selecciona Nunca, los objetos no se guardan nunca localmente, y todos los comandos `SharedObject.flush()` emitidos para el objeto devuelven el valor `false`.
- Si el usuario selecciona Ilimitado (mueve la barra deslizadora hasta el extremo derecho) los objetos se almacenan localmente hasta ocupar el espacio en disco disponible.
- Si el usuario selecciona Ninguno (mueve la barra deslizadora hasta el extremo izquierdo), todos los comandos `SharedObject.flush()` emitidos para el objeto devuelven el valor "pending" y hacen que el reproductor solicite al usuario si puede asignarse espacio en disco adicional para poder almacenar el objeto, como se ha indicado anteriormente.
- Si el usuario selecciona 10 KB, 100 KB, 1 MB o 10 MB, los objetos se guardan localmente y `SharedObject.flush()` devuelve el valor `true` si el objeto cabe en el espacio especificado. En caso de necesitar más espacio, `SharedObject.flush()` devuelve el valor "pending" y el reproductor pregunta al usuario si puede asignarse más espacio en disco para almacenar el objeto, como se ha descrito anteriormente.

Adicionalmente, si el usuario selecciona un valor inferior a la cantidad de espacio en disco que se está utilizando para los datos persistentes locales, el reproductor advierte al usuario que se eliminarán los objetos compartidos que se hayan almacenado localmente.



**Nota:** cuando se ejecuta Flash Player en el entorno de edición, no hay limitaciones de tamaño.

## Resumen de métodos para la clase `SharedObject`

Método	Descripción
<code>SharedObject.clear()</code>	Purga todos los datos del objeto compartido y elimina dicho objeto del disco.
<code>SharedObject.flush()</code>	Escribe inmediatamente un objeto local compartido y persistente en un archivo local.
<code>SharedObject.getLocal()</code>	Devuelve una referencia para un objeto local compartido y persistente que sólo está disponible para el cliente actual.
<code>SharedObject.getSize()</code>	Obtiene el tamaño actual del objeto compartido, expresado en bytes.

## Resumen de propiedades para la clase `SharedObject`

Propiedad (sólo lectura)	Descripción
<code>SharedObject.data</code>	Colección de atributos asignados a la propiedad <code>data</code> del objeto; estos atributos pueden ser compartidos o almacenados.

## Resumen de controladores de eventos para la clase SharedObject

Controlador de eventos	Descripción
<code>SharedObject.onStatus</code>	Se invoca cada vez que se emite un error, una advertencia o una nota informativa para un objeto compartido.

### Constructor para la clase SharedObject

Para más información sobre la creación de objetos locales compartidos, consulte `SharedObject.getLocal()`.

### SharedObject.clear()

#### Disponibilidad

Flash Player 7.

#### Sintaxis

```
my_so.clear()
```

#### Parámetros

Ninguno.

#### Valor devuelto

Ninguno.

#### Descripción

Método; purga todos los datos del objeto compartido y elimina el objeto del disco. La referencia a *my\_so* sigue activa y *my\_so* ahora aparece vacío.

### SharedObject.data

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
miObjetoCompartidoLocal.data
```

#### Descripción

Propiedad de sólo lectura; colección de atributos asignados a la propiedad `data` del objeto; estos atributos pueden ser compartidos o almacenados. Cada atributo puede ser un objeto de cualquiera de los tipos básicos de ActionScript o JavaScript: matriz, número, booleano, etc. En las líneas siguientes, por ejemplo, se asignan valores a distintos aspectos de un objeto compartido:

```
itemsArray = new Array(101,346,483);
currentUserIsAdmin = true;
currentUserName = "Ramona";
so.data.itemNumbers = itemsArray;
so.data.adminPrivileges = currentUserIsAdmin;
so.data.userName = currentUserName;
```

Todos los atributos de la propiedad `data` de un objeto compartido se guardan si el objeto es persistente.

**Nota:** los valores no deben asignarse directamente a la propiedad `data` de un objeto compartido, como en `so.data = someValue` ya que Flash pasa por alto dichas asignaciones.

Para eliminar los atributos para los objetos locales compartidos, utilice código del tipo `delete so.data.nombreAtributo`; si establece un atributo en `null` o `undefined` para un objeto local compartido, no se elimina el atributo.

Para crear valores “privados” para un objeto compartido (valores que sólo están disponibles para la instancia de cliente cuando el objeto está en uso y no se almacenan con el objeto cuando éste se cierra), debe crear propiedades con un nombre distinto de `data` para almacenarlos, como se muestra en el ejemplo siguiente.

```
so.favoriteColor = "azul";
so.favoriteNightClub = "The Bluenote Tavern";
so.favoriteSong = "My World is Blue";
```

### Ejemplo

En el ejemplo siguiente se establece el flujo actual en la opción seleccionada por el usuario.

```
curStream = _root.so.data.msgList[selected].streamName;
```

### Véase también

[Clase Sound](#)

## SharedObject.flush()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myLocalSharedObject.flush([minimumDiskSpace])
```

### Parámetros

*minimumDiskSpace* Número entero opcional que especifica el número de bytes que deben asignarse para este objeto. El valor predeterminado es 0.

### Valor devuelto

Un valor booleano `true` o `false`, o un valor de cadena `"pending"`.

- Si el usuario ha permitido el almacenamiento de información local para los objetos de este dominio y la cantidad de espacio asignado es suficiente para almacenar el objeto, este método devuelve `true`. Si ha pasado un valor para *minimumDiskSpace*, la cantidad de espacio que se asigne debe ser como mínimo igual a este valor para que el método devuelva `true`.
- Si el usuario ha permitido el almacenamiento de información local para los objetos de este dominio, pero la cantidad de espacio asignada no es suficiente para almacenar el objeto, este método devuelve `"pending"`.
- Si el usuario ha denegado de forma permanente el almacenamiento de información local para los objetos de este dominio, o si Flash no puede guardar el objeto por algún motivo, este método devuelve `false`.

## Descripción

Método; escribe inmediatamente un objeto compartido persistente en un archivo local. Si no utiliza este método, Flash escribe el objeto compartido en un archivo cuando finaliza la sesión del objeto compartido, es decir, cuando se cierra el archivo SWF, cuando se eliminan los datos innecesarios del objeto compartido porque ya no existe ninguna referencia al mismo o cuando se llama a `SharedObject.data`.

Si este método devuelve "pending", Flash Player muestra un cuadro de diálogo en el que se solicita al usuario que aumente la cantidad de espacio en disco disponible para los objetos de este dominio. Para permitir que el espacio para el objeto compartido "crezca" cuando se guarde en un futuro, y, por consiguiente, evitar que se devuelva el valor "pending", pase un valor para `minimumDiskSpace`. Cuando Flash intenta escribir el archivo, busca el número de bytes que se han pasado a `minimumDiskSpace`, en lugar de comprobar simplemente dónde hay espacio suficiente para guardar el objeto compartido con su tamaño actual.

Por ejemplo, si espera que un objeto compartido llegue a tener un tamaño máximo de 500 bytes, aun cuando en un principio sea mucho más pequeño, pase el valor 500 para `minimumDiskSpace`. Si Flash solicita al usuario que asigne espacio en disco para el objeto compartido, solicitará 500 bytes. Una vez que el usuario haya asignado el espacio necesario, Flash no deberá solicitar de nuevo más espacio en intentos futuros de vaciar el objeto (siempre y cuando el tamaño no sea superior a 500 bytes).

Cuando el usuario ha respondido al cuadro de diálogo, se llama de nuevo a este método y devuelve `true` o `false`; asimismo, se invoca `SharedObject.onStatus` con una propiedad `code` con el valor `SharedObject.Flush.Success` o `SharedObject.Flush.Failed`.

Para más información, consulte [“Consideraciones sobre el espacio en el disco local” en la página 620](#).

## Ejemplo

La función siguiente obtiene un objeto compartido, `S0`, y rellena propiedades que pueden escribirse con los valores proporcionados por el usuario. Por último, se llama a `flush()` para guardar los valores y asignar un mínimo de 1.000 bytes de espacio en disco.

```
this.SyncSettingsCore=function(soname, override, settings)
{
    var S0=SharedObject.getLocal(soname, "http://www.mydomain.com/app/sys");

    // Índice de la lista de valores de configuración
    var i;

    // Para cada valor especificado en la configuración:
    // Si override es true, establecer la configuración persistente en el valor
    // proporcionado.
    // Si override es false, buscar la configuración persistente, a menos que
    // no exista ninguna, en cuyo caso, debe establecerse en el valor
    // proporcionado.
    for (i in settings) {
        if (override || (S0.data[i] == null)) {
            S0.data[i]= settings[i];
        } else {
            settings[i]= S0.data[i];
        }
    }
    S0.flush(1000);
}
```



# SharedObject.getLocal()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
SharedObject.getLocal(objectName [, localPath])
```

**Nota:** la sintaxis correcta es `SharedObject.getLocal`. Para asignar el objeto a una variable, utilice la sintaxis de tipo `myLocalSO = SharedObject.getLocal`.

## Parámetros

*objectName* Nombre del objeto. El nombre puede incluir barras diagonales (/); por ejemplo, `direcciones/trabajo` es un nombre válido. En un nombre de objeto compartido no pueden utilizarse ni espacios ni los caracteres siguientes:

~ % & \ ; : " ' , < > ? #

*localPath* Parámetro de cadena opcional que especifica la ruta completa o parcial al archivo SWF que ha creado el objeto compartido, y que determina la ruta local en la que se almacenará el objeto compartido. El valor predeterminado es la ruta completa.

## Valor devuelto

Una referencia a un objeto compartido que es persistente a nivel local y que sólo está disponible para el cliente actual. Si Flash no puede crear o encontrar el objeto compartido (por ejemplo, si se ha especificado una *localPath* pero dicho directorio no existe), este método devuelve `null`.

## Descripción

Método; devuelve una referencia a un objeto persistente y compartido localmente que sólo está disponible para el cliente actual.

Para evitar problemas con los nombres, Flash comprueba la ubicación del archivo SWF que está creando el objeto compartido. Por ejemplo, si un archivo SWF de `www.myCompany.com/apps/stockwatcher.swf` crea un objeto compartido denominado `portfolio`, dicho objeto compartido no entrará en conflicto con otro objeto denominado `portfolio` creado por un archivo SWF en `www.yourCompany.com/photoshoot.swf`, porque ambos archivos SWF proceden de directorios distintos.

## Ejemplo

En el ejemplo siguiente se guarda el último fotograma introducido por el usuario en un objeto compartido localmente denominado `kookie`.

```
// Obtener el objeto kookie
so = sharedobject.getLocal("kookie");

// Obtener el usuario de kookie e ir al número de fotograma guardado para este
// usuario.
if (so.data.user != undefined) {
    this.user = so.data.user;
    this.gotoAndStop(so.data.frame);
}
```

El bloque de código siguiente se coloca en cada fotograma de archivo SWF.

```
// En cada fotograma, llamar a la función rememberme para guardar el número de
// fotograma.
function rememberme() {
    so.data.frame=this._currentFrame;
    so.data.user="John";
}
```

## SharedObject.getSize()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myLocalSharedObject.getSize()
```

### Parámetros

Ninguno.

### Valor devuelto

Un valor numérico que especifica el tamaño del objeto compartido, expresado en bytes.

### Descripción

Método; obtiene el tamaño actual del objeto compartido, expresado en bytes.

Flash calcula el tamaño de un objeto compartido revisando paso a paso cada una de las propiedades de sus datos; cuantas más propiedades de datos tenga el objeto, más tiempo se necesitará para calcular su tamaño. Por eso estimar el tamaño de los objetos puede tener un coste de procesamiento considerable. Por consiguiente, es aconsejable no utilizar este método, a menos que sea realmente necesario.

### Ejemplo

En el objeto siguiente se obtiene el tamaño del objeto compartido so.

```
var soSize= this.so.getSize();
```

## SharedObject.onStatus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
myLocalSharedObject.onStatus = function(infoObject) {
    // las sentencias se escriben aquí
}
```

### Parámetros

*infoObject*    Parámetro definido de acuerdo con el mensaje de estado.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cada vez que se envía una nota de error, de advertencia o informativa para un objeto compartido. Si desea responder a este controlador de eventos, debe crear una función para procesar el objeto de información generado por el objeto compartido.

Cada objeto de información tiene una propiedad `code` con una cadena que describe el resultado del controlador `onStatus`, y una propiedad `level` que contiene una cadena que puede ser "Status" o "Error".

Además de este controlador `onStatus`, Flash también proporciona una función de superclase denominada `System.onStatus`. Si se invoca `onStatus` para un objeto determinado y no existe ninguna función asignada para responderle, Flash procesará una función asignada a `System.onStatus` si existe.

Los siguientes eventos envían una notificación si ocurren ciertas actividades de `SharedObject`.

Propiedad Code	Propiedad Level	Significado
<code>SharedObject.Flush.Failed</code>	Error	Un comando <code>SharedObject.flush()</code> que ha devuelto "pending" no ha podido ejecutarse (el usuario no ha asignado espacio en disco adicional para el objeto compartido cuando Flash Player ha mostrado el cuadro de diálogo Parámetros de almacenamiento local).
<code>SharedObject.Flush.Success</code>	Status	Un comando <code>SharedObject.flush()</code> que ha devuelto "pending" ha finalizado correctamente (el usuario ha asignado espacio adicional para el objeto compartido).

## Véase también

`SharedObject.getLocal()`, `System.onStatus`

# Clase Sound

## Disponibilidad

Flash Player 5.

## Descripción

La clase `Sound` permite controlar el sonido de una película. Puede agregar sonidos a un clip de película desde la biblioteca mientras se reproduce la película y controlar estos sonidos. Si no especifica un destino al crear un nuevo objeto `Sound`, puede utilizar los métodos para controlar el sonido de toda la película.

Antes de llamar a los métodos de la clase `Sound`, debe utilizar el constructor `new Sound` para crear un objeto `Sound`.

## Resumen de métodos para la clase Sound

Método	Descripción
<code>Sound.attachSound()</code>	Asocia el sonido especificado en el parámetro.
<code>Sound.getBytesLoaded()</code>	Devuelve el número de bytes cargados para el sonido especificado.
<code>Sound.getBytesTotal()</code>	Devuelve el tamaño del sonido en bytes.
<code>Sound.getPan()</code>	Devuelve el valor de la llamada <code>setPan()</code> anterior.
<code>Sound.getTransform()</code>	Devuelve el valor de la llamada <code>setTransform()</code> anterior.
<code>Sound.getVolume()</code>	Devuelve el valor de la llamada <code>setVolume()</code> anterior.
<code>Sound.loadSound()</code>	Carga un archivo MP3 en Flash Player.
<code>Sound.setPan()</code>	Establece el balance izquierda/derecha del sonido.
<code>Sound.setTransform()</code>	Establece el valor de cada canal, izquierdo y derecho, que debe reproducirse en cada altavoz.
<code>Sound.setVolume()</code>	Establece el nivel de volumen de un sonido.
<code>Sound.start()</code>	Comienza a reproducir un sonido desde el principio u opcionalmente desde un punto de desplazamiento establecido en el parámetro.
<code>Sound.stop()</code>	Detiene el sonido especificado o todos los sonidos que se están reproduciendo actualmente.

## Resumen de propiedades para la clase Sound

Propiedad	Descripción
<code>Sound.duration</code>	Duración de un sonido, expresada en milisegundos.
<code>Sound.ID3</code>	Proporciona acceso a los metadatos que forman parte de un archivo MP3.
<code>Sound.position</code>	Número de milisegundos durante los que se ha reproducido el sonido.

## Resumen de controladores de eventos para la clase Sound

Controlador de eventos	Descripción
<code>Sound.onID3</code>	Se invoca cada vez que existen nuevos datos ID3 disponibles.
<code>Sound.onLoad</code>	Se invoca cuando se carga un sonido.
<code>Sound.onSoundComplete</code>	Se invoca cuando se detiene la reproducción de un sonido.

## Constructor para la clase Sound

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new Sound([target])
```

### Parámetros

*target* Instancia de clip de película en la que opera el objeto Sound. Este parámetro es opcional.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto Sound para un clip de película especificado. Si no se especifica una instancia de destino, el objeto Sound controla todos los sonidos de la película.

### Ejemplo

En el ejemplo siguiente se crea un nuevo objeto Sound denominado `global_sound`. La segunda línea llama al método `setVolume()` y ajusta el volumen de todos los sonidos de la película a un 50%.

```
global_sound = new Sound();  
global_sound.setVolume(50);
```

En el ejemplo siguiente se crea un nuevo objeto Sound, se le pasa el clip de película de destino `my_mc` y se llama al método `start`, que inicia cualquier sonido en `my_mc`.

```
movie_sound = new Sound(my_mc);  
movie_sound.start();
```

## Sound.attachSound()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.attachSound(" idName ")
```

### Parámetros

*idName* Identificador de un sonido exportado en la biblioteca. El identificador se encuentra en el cuadro de diálogo Propiedades de vinculación.

### Valor devuelto

Ninguno.

### Descripción

Método; asocia el sonido especificado en el parámetro *idName* al objeto Sound especificado. El sonido debe estar en la biblioteca del archivo SWF actual y debe estar especificado para la exportación en el cuadro de diálogo Propiedades de vinculación. Debe llamar a [Sound.start\(\)](#) para empezar a reproducir el sonido.

Para asegurarse de que el sonido puede controlarse desde cualquier escena del archivo SWF, coloque el sonido en la línea de tiempo principal del archivo SWF.

## Sound.duration

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_sound.duration
```

### Descripción

Propiedad (sólo lectura); duración de un sonido, expresada en milisegundos.

## Sound.getBytesLoaded()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_sound.getBytesLoaded()
```

### Parámetros

Ninguno.

### Valor devuelto

Entero que indica el número de bytes cargados.

### Descripción

Método; devuelve el número de bytes cargados (en flujo) para el objeto Sound especificado. Puede comparar el valor de `getBytesLoaded()` con el de `getBytesTotal()` para determinar el porcentaje que se ha cargado de un sonido.

### Véase también

[Sound.getBytesTotal\(\)](#)

## Sound.getBytesTotal()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_sound.getBytesTotal()
```

### Parámetros

Ninguno.

### Valor devuelto

Entero que indica el tamaño total, en bytes, del objeto Sound especificado.

### Descripción

Método; devuelve el tamaño, en bytes, del objeto Sound especificado.

**Véase también**

[Sound.getBytesLoaded\(\)](#)

## Sound.getPan()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.getPan();
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el nivel de panorámica establecido en la última llamada `setPan()` como un entero entre -100 (izquierda) y 100 (derecha). 0 establece los canales izquierdo y derecho de la misma forma. El valor de panorámica controla el balance izquierdo-derecho de los sonidos actuales y futuros de un archivo SWE.

Este método es acumulativo con `setVolume()` o `setTransform()`.

**Véase también**

[Sound.setPan\(\)](#)

## Sound.getTransform()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.getTransform();
```

### Parámetros

Ninguno.

### Valor devuelto

Un objeto con propiedades que contienen los valores de porcentaje de canal para el objeto de sonido especificado.

### Descripción

Método; devuelve la información de transformación de sonido para el objeto Sound especificado con la última llamada a [Sound.setTransform\(\)](#).

## Sound.getVolume()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.getVolume()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve el nivel de volumen de sonido como un número entero de 0 a 100, donde 0 es apagado y 100 es a todo volumen. El valor predeterminado es 100.

### Véase también

[Sound.setVolume\(\)](#)

## Sound.ID3

### Disponibilidad

Flash Player 6; comportamiento actualizado en Flash Player 7.

### Sintaxis

```
my_sound.ID3
```

### Descripción

Propiedad (sólo lectura); proporciona acceso a los metadatos que forman parte de un archivo MP3.

Los archivos de sonido MP3 pueden contener etiquetas ID3, que proporcionan metadatos sobre el archivo. Si un sonido MP3 que ha cargado con [Sound.attachSound\(\)](#) o [Sound.loadSound\(\)](#) contiene etiquetas ID3, puede consultar estas propiedades. Sólo se admiten las etiquetas ID3 que utilizan el conjunto de caracteres UTF-8.

Flash Player 6 versión 4.0 y posteriores utilizan la propiedad `Sound.id3` para admitir etiquetas ID3 1.0 e ID3 1.1. Flash Player 7 amplía la compatibilidad para las etiquetas ID3 2.0, específicamente 2.3 y 2.4. A fin de ofrecer compatibilidad con versiones anteriores, se admite `Sound.id3` y `Sound.ID3`. Las sugerencias para el código sólo se admiten para el uso en minúsculas de `id3` (véase [“Utilización de las sugerencias para el código” en la página 66](#)).



En la tabla siguiente se enumeran las etiquetas ID3 2.0 estándar y el tipo de contenido que representan; puede consultarlas con el formato *my\_sound.ID3.COMM*, *my\_sound.ID3.TIME*, etc. Los archivos MP3 pueden contener etiquetas distintas de las indicadas en esta tabla; Sound.ID3 también proporciona acceso a dichas etiquetas.

Propiedad	Descripción
COMM	Comentario
TALB	Título del álbum/película/espectáculo
TBPM	Pulsaciones por minuto
TCOM	Compositor
TCON	Tipo de contenido
TCOP	Mensaje de copyright
TDAT	Fecha
TDLY	Demora de la lista de reproducción
TENC	Codificado por
TEXT	Autor del texto/letras de canciones
TFLT	Tipo de archivo
TIME	Hora
TIT1	Descripción del grupo de contenido
TIT2	Título/nombre de la canción/descripción del contenido
TIT3	Perfeccionamiento de subtítulo/descripción
TKEY	Tecla inicial
TLAN	Idiomas
TLEN	Longitud
TMED	Tipo de medio
TOAL	Título original del álbum, la película o la presentación
TOFN	Nombre de archivo original
TOLY	Autores de letras de canciones/texto originales
TOPE	Artistas/actores originales
TORY	Año original de estreno
TOWN	Propietario del archivo/licencia
TPE1	Actores/solistas principales
TPE2	Banda/orquesta/acompañamiento
TPE3	Perfeccionamiento del director/actor
TPE4	Interpretado, mezclado o modificado de otro modo por
TP0S	Parte de un conjunto

Propiedad	Descripción
TPUB	Editor
TRCK	Número de pista y posición en el conjunto
TRDA	Fechas de grabación
TRSN	Nombre de la emisora de radio de Internet
TRSO	Propietario de la emisora de radio de Internet
TSIZ	Tamaño
TSRC	ISRC (código de grabación estándar internacional)
TSSE	Software o hardware y configuración utilizados para la codificación
TYER	Año
WXXX	Fotograma de vínculo de URL

Flash Player 6 admitía varias etiquetas ID3 1.0. Si el archivo MP3 no contiene estas etiquetas pero sí las etiquetas ID3 2.0 correspondientes, las etiquetas ID3 2.0 se copian en las propiedades de ID3 1.0 como se muestra en la tabla siguiente. Este proceso proporciona compatibilidad con versiones anteriores de los scripts que pueda haber escrito con propiedades ID3 1.0.

Etiqueta ID3 2.0	Propiedad ID3 1.0 correspondiente
COMM	<code>Sound.id3.comment</code>
TALB	<code>Sound.id3.album</code>
TCO	<code>Sound.id3.genre</code>
TIT2	<code>Sound.id3.songname</code>
TPE1	<code>Sound.id3.artist</code>
TRCK	<code>Sound.id3.track</code>
TYER	<code>Sound.id3.year</code>

### Ejemplo

Consulte [Sound.onID3](#) para obtener un ejemplo sobre el uso de esta propiedad.

### Véase también

[Sound.attachSound\(\)](#), [Sound.loadSound\(\)](#)

## Sound.loadSound()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_sound.loadSound("url", isStreaming)
```

### Parámetros

*url* Ubicación de un archivo de sonido MP3 en un servidor.

*isStreaming* Valor booleano que indica si el sonido es un flujo de sonido (*true*) o un sonido de evento (*false*).

#### Valor devuelto

Ninguno.

#### Descripción

Método; carga un archivo MP3 en un objeto Sound. Puede utilizar el parámetro *isStreaming* para indicar si el sonido es de evento o de flujo.

Los sonidos de evento se cargan completamente antes de reproducirse. Los gestiona la clase Sound de ActionScript y responden a todos los métodos y las propiedades de esta clase.

Los sonidos de flujo se reproducen mientras se descargan. La reproducción empieza cuando se han recibido suficientes datos para iniciar el descompresor.

Todos los archivos MP3 (de evento o flujo) cargados con este método se guardan en la memoria caché de archivos del navegador del sistema del usuario.

#### Ejemplo

En el ejemplo siguiente se carga un sonido de evento:

```
my_sound.loadSound( "http://serverpath:port/mp3filename", false);
```

En el ejemplo siguiente se carga un sonido de flujo:

```
my_sound.loadSound( "http://serverpath:port/mp3filename", true);
```

#### Véase también

[Sound.onLoad](#)

## Sound.onID3

#### Disponibilidad

Flash Player 7.

#### Sintaxis

```
my_sound.onID3 = function(){  
    // las sentencias se escriben aquí  
}
```

#### Parámetros

Ninguno.

#### Valor devuelto

Ninguno.

#### Descripción

Controlador de eventos; se invoca cada vez que existen nuevos datos ID3 disponibles para un archivo MP3 que se carga mediante [Sound.attachSound\(\)](#) o [Sound.loadSound\(\)](#). Este controlador proporciona acceso a los datos ID3 sin realizar ningún sondeo. Si un archivo contiene etiquetas ID3 1.0 y 2.0, este controlador se llama dos veces.

## Ejemplo

En el ejemplo siguiente se rastrean las propiedades ID3 de song.mp3 en el panel Salida.

```
my_sound = new Sound();
my_sound.onID3 = function(){
    for( var prop in my_sound.ID3 ){
        trace( prop + " : "+ my_sound.ID3[prop] );
    }
}
my_sound.loadSound("song.mp3", false);
```

## Véase también

[Sound.attachSound\(\)](#), [Sound.ID3](#), [Sound.loadSound\(\)](#)

# Sound.onLoad

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_sound.onLoad = function(success){
    // las sentencias se escriben aquí
}
```

## Parámetros

*success* Valor booleano `true` si *my\_sound* se ha cargado correctamente; de lo contrario, tendrá el valor `false`.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca automáticamente cuando se carga un sonido. Debe crear una función que se ejecute cuando se invoque este controlador. Puede utilizar una función anónima o una función con nombre (para obtener un ejemplo, véase [Sound.onSoundComplete](#)). Debe definir este controlador antes de llamar a *my\_sound.loadSound()*.

## Véase también

[Sound.loadSound\(\)](#)

# Sound.onSoundComplete

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_sound.onSoundComplete = function() {
    // las sentencias se escriben aquí
}
```

**Parámetros**

Ninguno.

**Valor devuelto**

Ninguno.

**Descripción**

Controlador de eventos; se invoca automáticamente cuando finaliza la reproducción de un sonido. Puede utilizar este controlador para desencadenar eventos en un archivo SWF cuando finaliza la reproducción de un sonido.

Debe crear una función que se ejecute cuando se invoque este controlador de eventos. Puede utilizar una función anónima o una función con nombre.

**Ejemplo**

Sintaxis 1: en el ejemplo siguiente se utiliza una función anónima:

```
my_sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = function() {
    trace("mySoundID ha finalizado");
}
my_sound.start();
```

Sintaxis 2: en el ejemplo siguiente se utiliza una función con nombre:

```
function callback1() {
    trace("mySoundID ha finalizado");
}

my_sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = callback1;
my_sound.start();
```

**Véase también**

[Sound.onLoad](#)

## Sound.position

**Disponibilidad**

Flash Player 6.

**Sintaxis**

*my\_sound.position*

**Descripción**

Propiedad (sólo lectura); número de milisegundos durante los que se ha reproducido un sonido. Si el sonido se repite, la posición se restablecerá en 0 al principio de cada repetición.

## Sound.setPan()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.setPan(pan);
```

### Parámetros

*pan* Número entero que especifica el balance izquierda-derecha de un sonido. Los valores válidos deben estar comprendidos entre -100 y 100, donde -100 utiliza solamente el canal izquierdo, 100 utiliza solamente el canal derecho y 0 establece un balance de sonido uniforme entre los dos canales.

### Valor devuelto

Un número entero.

### Descripción

Método; determina cómo se reproduce el sonido en los canales izquierdo y derecho (altavoces). Para los sonidos mono, *pan* determina a través de qué altavoz (izquierdo o derecho) se reproduce el sonido.

### Ejemplo

En el ejemplo siguiente se crea un objeto Sound denominado *my\_sound* y se asocia un sonido con un identificador *L7* de la biblioteca. También se llama a *setVolume()* y *setPan()* para controlar el sonido *L7*.

```
onClipEvent(mouseDown) {  
    // crea un objeto Sound  
    my_sound = new Sound(this);  
    // asocia un sonido de la biblioteca  
    my_sound.attachSound("L7");  
    //establecer el volumen al 50%  
    my_sound.setVolume(50);  
    //desactivar el sonido en el canal derecho  
    my_sound.setPan(-100);  
    //empezar 30 segundos después del inicio del sonido y reproducirlo 5 veces  
    my_sound.start(30, 5);  
}
```

### Véase también

[Sound.attachSound\(\)](#), [Sound.setPan\(\)](#), [Sound.setTransform\(\)](#), [Sound.setVolume\(\)](#), [Sound.start\(\)](#)

## Sound.setTransform()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.setTransform(soundTransformObject)
```

## Parámetros

*soundTransformObject*    Objeto creado con el constructor de la clase *Object* genérica.

## Valor devuelto

Ninguno.

## Descripción

Método; establece la información de transformación, o de balance, de sonido para un objeto *Sound*.

El parámetro *soundTransformObject* es un objeto que se crea utilizando el método constructor de la clase *Object* genérica con parámetros que especifican cómo se distribuye el sonido en los canales izquierdo y derecho (altavoces).

Los sonidos necesitan una cantidad considerable de espacio en la unidad de disco y en la memoria. Debido a que los sonidos estéreo utilizan el doble de datos que los sonidos mono, normalmente es mejor utilizar sonidos mono de 6 bits a 22 KHz. Puede utilizar el método *setTransform()* para reproducir sonidos mono como sonidos estéreo, reproducir sonidos estéreo como sonidos mono y agregar efectos interesantes a los sonidos.

Las propiedades de *soundTransformObject* son las siguientes:

*ll*    Porcentaje que especifica qué cantidad de la entrada izquierda se reproduce en el altavoz izquierdo (de 0 a 100).

*lr*    Porcentaje que especifica qué cantidad de la entrada derecha se reproduce en el altavoz izquierdo (de 0 a 100).

*rr*    Porcentaje que especifica qué cantidad de la entrada derecha se reproduce en el altavoz derecho (de 0 a 100).

*rl*    Porcentaje que especifica qué cantidad de la entrada izquierda se reproduce en el altavoz derecho (de 0 a 100).

El resultado neto de los parámetros se representa mediante la fórmula siguiente:

```
leftOutput = left_input * ll + right_input * lr  
rightOutput = right_input * rr + left_input * rl
```

Los valores de *left\_input* y *right\_input* se determinan según el tipo del sonido (estéreo o mono) del archivo SWF.

Los sonidos estéreo dividen la entrada del sonido uniformemente entre los altavoces izquierdo y derecho y de forma predeterminada tienen la configuración de transformación siguiente:

```
ll = 100  
lr = 0  
rr = 100  
rl = 0
```

Los sonidos mono reproducen toda la entrada de sonido en el altavoz izquierdo y de forma predeterminada tienen la configuración de transformación siguiente:

```
ll = 100  
lr = 100  
rr = 0  
rl = 0
```

## Ejemplo

En el ejemplo siguiente se muestra una configuración que puede lograrse mediante el método `setTransform()`, pero no con los métodos `setVolume()` y `setPan()`, aun cuando se combinen.

El código siguiente crea un nuevo objeto `soundTransformObject` y establece sus propiedades de modo que el sonido de ambos canales se reproduzca sólo en el canal izquierdo.

```
mySoundTransformObject = new Object;
mySoundTransformObject.ll = 100;
mySoundTransformObject.lr = 100;
mySoundTransformObject.rr = 0;
mySoundTransformObject.rl = 0;
```

Para aplicar el objeto `soundTransformObject` a un objeto `Sound`, debe pasar el objeto al objeto `Sound` utilizando el método `setTransform()` del modo siguiente:

```
my_sound.setTransform(mySoundTransformObject);
```

En el ejemplo siguiente se reproduce un sonido estéreo como sonido mono; el objeto `soundTransformObjectMono` tiene los parámetros siguientes.

```
mySoundTransformObjectMono = new Object;
mySoundTransformObjectMono.ll = 50;
mySoundTransformObjectMono.lr = 50;
mySoundTransformObjectMono.rr = 50;
mySoundTransformObjectMono.rl = 50;
my_sound.setTransform(soundTransformObjectMono);
```

Este ejemplo reproduce el canal izquierdo a media capacidad y añade el resto del canal izquierdo al canal derecho; el objeto `soundTransformObjectHalf` tiene los parámetros siguientes.

```
mySoundTransformObjectHalf = new Object;
mySoundTransformObjectHalf.ll = 50;
mySoundTransformObjectHalf.lr = 0;
mySoundTransformObjectHalf.rr = 100;
mySoundTransformObjectHalf.rl = 50;
my_sound.setTransform(soundTransformObjectHalf);
```

## Véase también

[Clase Object](#)

# Sound.setVolume()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_sound.setVolume(volume)
```

## Parámetros

*volume* Número de 0 a 100 que representa un nivel de volumen. 100 es a todo volumen y 0 es sin volumen. El valor predeterminado es 100.

## Valor devuelto

Ninguno.



## Descripción

Método; establece el volumen para el objeto Sound.

## Ejemplo

En el ejemplo siguiente se establece el volumen al 50% y se transfiere el sonido a lo largo del tiempo desde el altavoz izquierdo al altavoz derecho.

```
onClipEvent (load) {  
    i = -100;  
    my_sound = new Sound();  
    my_sound.setVolume(50);  
}  
onClipEvent(enterFrame) {  
    if (i <= 100) {  
        my_sound.setPan(i++);  
    }  
}
```

## Véase también

[Sound.setPan\(\)](#), [Sound.setTransform\(\)](#)

# Sound.start()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_sound.start([secondOffset, loop])
```

## Parámetros

*secondOffset* Parámetro opcional que permite iniciar la reproducción del sonido en un momento específico. Por ejemplo, si tiene un sonido de 30 segundos y desea que el sonido comience a reproducirse a la mitad, especifique 15 en el parámetro *secondOffset*. El sonido no se retrasa 15 segundos, sino que comienza a reproducirse en la marca de 15 segundos.

*loop* Parámetro opcional que permite especificar el número de veces que debe reproducirse el sonido de manera consecutiva.

## Valor devuelto

Ninguno.

## Descripción

Método; comienza a reproducir el último sonido asociado desde el principio, si no se especifican parámetros, o desde un punto determinado del sonido especificado por el parámetro *secondOffset*.

## Véase también

[Sound.stop\(\)](#)

## Sound.stop()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_sound.stop(["idName"])
```

### Parámetros

*idName* Parámetro opcional que especifica que se detenga la reproducción de un sonido concreto. El parámetro *idName* debe escribirse entre comillas (" ").

### Valor devuelto

Ninguno.

### Descripción

Método; detiene todos los sonidos que se están reproduciendo actualmente si no se especifican parámetros, o solamente el sonido especificado en el parámetro *idName*.

### Véase también

[Sound.start\(\)](#)

## \_soundbuftime

### Disponibilidad

Flash Player 4.

### Sintaxis

```
_soundbuftime = integer
```

### Parámetros

*integer* Número de segundos que deben transcurrir antes de que empiece el flujo del archivo SWF.

### Descripción

Propiedad (global); establece el número de segundos de sonido que va a almacenarse previamente en una memoria intermedia. El valor predeterminado es 5 segundos.

## Clase Stage

### Disponibilidad

Flash Player 6.

### Descripción

La clase Stage es una clase de nivel superior a cuyos métodos, propiedades y controladores se puede acceder sin utilizar un constructor.

Mediante los métodos y las propiedades de esta clase podrá acceder a información sobre los límites de un archivo SWF y manipularla.

## Resumen de métodos para la clase Stage

Método	Descripción
<a href="#">Stage.addListener()</a>	Añade un objeto detector que detecta cuándo se cambia el tamaño de un archivo SWF.
<a href="#">Stage.removeListener()</a>	Elimina un objeto detector del objeto Stage.

## Resumen de propiedades para la clase Stage

Propiedad	Descripción
<a href="#">Stage.align</a>	Alineación del archivo SWF en el reproductor o en el navegador.
<a href="#">Stage.height</a>	Altura del escenario, en píxeles.
<a href="#">Stage.scaleMode</a>	Escala actual del archivo SWF.
<a href="#">Stage.showMenu</a>	Muestra u oculta los elementos predeterminados del menú contextual de Flash Player.
<a href="#">Stage.width</a>	Anchura del escenario, en píxeles.

## Resumen de controladores de eventos para la clase Stage

Controlador de eventos	Descripción
<a href="#">Stage.onResize</a>	Se invoca cuando <code>Stage.scaleMode</code> tiene el valor "noScale" y se cambia el tamaño del archivo SWF.

## Stage.addListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Stage.addListener(myListener)
```

### Parámetros

*myListener* Objeto que detecta una notificación de callback del evento [Stage.onResize](#).

### Valor devuelto

Ninguno.

### Descripción

Método; detecta cuándo se cambia el tamaño de un archivo SWF (pero sólo si `Stage.scaleMode` = "noScale"). El método `addListener()` no funciona con el valor predeterminado de escala de película ("showAll") ni con otros valores de escala ("exactFit" y "noBorder").

Para utilizar `addListener()`, primero debe crear un *objeto detector*. Los objetos detectores del escenario reciben una notificación de `Stage.onResize`.

## Ejemplo

Este ejemplo crea un nuevo objeto detector denominado `myListener`. A continuación, utiliza `myListener` para llamar al evento `onResize` y definir una función a la que se llamará cuando se desencadene el evento `onResize`. Finalmente, el código agrega el objeto `myListener` a la lista de callbacks del objeto `Stage`. Los objetos detectores permiten que varios objetos detecten notificaciones de cambio de tamaño.

```
myListener = new Object();
myListener.onResize = function () { ... }
Stage.scaleMode = "noScale"
Stage.addListener(myListener);
```

## Véase también

[Stage.onResize](#), [Stage.removeListener\(\)](#)

# Stage.align

## Disponibilidad

Flash Player 6.

## Sintaxis

`Stage.align`

## Descripción

Propiedad; indica la alineación actual del archivo SWF en el reproductor o el navegador.

La tabla siguiente muestra los valores de la propiedad `align`. Los valores que no aparecen aquí centran el archivo SWF en el área del reproductor o del navegador.

Valor	Vertical	Horizontal
"T"	superior	centro
"B"	inferior	centro
"L"	centro	izquierda
"R"	centro	derecha
"TL"	superior	izquierda
"TR"	superior	derecha
"BL"	inferior	izquierda
"BR"	inferior	derecha

# Stage.height

## Disponibilidad

Flash Player 6.

## Sintaxis

`Stage.height`

## Descripción

Propiedad (sólo lectura); indica la altura actual, en píxeles, del escenario. Cuando el valor de `Stage.scaleMode` es "noScale", la propiedad `height` representa la altura del reproductor. Cuando el valor de `Stage.scaleMode` no es "noScale", `height` representa la altura del archivo SWF.

## Véase también

`Stage.align`, `Stage.scaleMode`, `Stage.width`

# Stage.onResize

## Disponibilidad

Flash Player 6.

## Sintaxis

```
myListener.onResize = function () {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

Ninguno.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando `Stage.scaleMode` se establece en "noScale" y se modifica el tamaño del archivo SWF. Puede utilizar este controlador de eventos para escribir una función que disponga los objetos en el escenario cuando se modifica el tamaño de un archivo SWF.

## Ejemplo

En el ejemplo siguiente se envía un mensaje al panel Salida cuando se modifica el tamaño del escenario.

```
Stage.scaleMode = "noScale"  
myListener = new Object();  
myListener.onResize = function () {  
    trace("El tamaño del escenario es ahora " + Stage.width + " por " +  
        Stage.height);  
}  
Stage.addListener(myListener);  
// Más adelante, llamar a Stage.removeListener(myListener)
```

## Véase también

`Stage.addListener()`, `Stage.removeListener()`

## Stage.removeListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Stage.removeListener(myListener)
```

### Parámetros

*myListener*    Objeto que se añade a una lista callback del objeto con `addListener()`.

### Valor devuelto

Valor booleano.

### Descripción

Método; elimina un objeto detector creado con `addListener()`.

### Véase también

[Stage.addListener\(\)](#)

## Stage.scaleMode

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Stage.scaleMode = "value"
```

### Descripción

Propiedad; indica la escala actual del archivo SWF en el escenario. La propiedad `scaleMode` hace que se aplique un determinado modo de escala al archivo SWF. De forma predeterminada, el archivo SWF utiliza los parámetros HTML establecidos en el cuadro de diálogo Configuración de publicación.

La propiedad `scaleMode` puede utilizar los valores "exactFit", "showAll", "noBorder" y "noScale". Con cualquier otro valor, la propiedad `scaleMode` se establece en el valor predeterminado "showAll".

## Stage.showMenu

### Disponibilidad

Flash Player 6.

### Sintaxis

```
Stage.showMenu
```

### Descripción

Propiedad (de lectura-escritura); especifica si se mostrarán o se ocultarán los elementos predeterminados del menú contextual de Flash Player. Si se establece `showMenu` en `true` (que es el valor predeterminado), aparecen todos los elementos del menú contextual. Si se establece `showMenu` en `false`, sólo aparecen los elementos de Configuración.

### Véase también

[Clase `ContextMenu`](#), [Clase `ContextMenuItem`](#)

## Stage.width

### Disponibilidad

Flash Player 6.

### Sintaxis

`Stage.width`

### Descripción

Propiedad (sólo lectura); indica la anchura actual, en píxeles, del escenario. Cuando el valor de [Stage.scaleMode](#) es `"noScale"`, la propiedad `width` representa la anchura del reproductor. Cuando el valor de [Stage.scaleMode](#) no es `"noScale"`, `width` representa la anchura del archivo SWF.

### Véase también

[Stage.align](#), [Stage.height](#), [Stage.scaleMode](#)

## startDrag()

### Disponibilidad

Flash Player 4.

### Sintaxis

`startDrag(target, [lock, left, top, right, bottom])`

### Parámetros

*target* Ruta de destino del clip de película que desea arrastrar.

*lock* Valor booleano que especifica si el clip de película arrastrable está bloqueado en el centro de la posición del ratón (*true*) o en el punto en el que el usuario hizo clic por primera vez en el clip de película (*false*). Este parámetro es opcional.

*left, top, right, bottom* Valores relativos a las coordenadas del elemento principal del clip de película que especifican un rectángulo de limitación para el clip de película. Estos parámetros son opcionales.

### Valor devuelto

Ninguno.

## Descripción

Función; hace que el clip de película de *target* se pueda arrastrar mientras se reproduce la película. Sólo un clip de película puede arrastrarse al mismo tiempo. Una vez que se ha ejecutado una operación `startDrag`, el clip de película se puede seguir arrastrando hasta que se detiene específicamente mediante una acción `stopDrag()`, o hasta que se llame a una acción `startDrag` de otro clip de película.

## Ejemplo

Para crear un clip de película que los usuarios puedan colocar en cualquier ubicación, asocie las acciones `startDrag()` y `stopDrag()` a un botón dentro del clip de película.

```
on(press){
    startDrag(this,true);
}
on(release) {
    stopDrag();
}
```

## Véase también

`MovieClip._droptarget`, `MovieClip.startDrag()`, `stopDrag()`

# static

## Disponibilidad

Flash Player 6.

## Sintaxis

```
class someClassName{
    static var name;
    static function name() {
        // las sentencias se escriben aquí
    }
}
```

**Nota:** para utilizar esta palabra clave, debe especificar ActionScript 2.0 y Flash Player 6 o posterior en la ficha Flash del cuadro de diálogo Configuración de publicación del archivo FLA. Esta palabra clave sólo se admite si se utiliza en archivos de script externos, no en scripts escritos en el panel Acciones.

## Parámetros

*name* Nombre de la variable o función que desea especificar como estática.

## Descripción

Palabra clave; especifica que una variable o función se ha creado una sola vez para cada clase en lugar de crearse en cada objeto basado en esa clase. Para más información, consulte [“Miembros de clase e instancia” en la página 171](#).

Puede utilizar esta palabra clave sólo dentro de las definiciones de clase, no en las definiciones de interfaz.

## Véase también

`private`, `public`



## stop()

### Disponibilidad

Flash 2.

### Sintaxis

```
stop()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; detiene el archivo SWF que se está reproduciendo actualmente. La utilización más corriente de esta acción es para controlar los clips de película con botones.

## stopAllSounds()

### Disponibilidad

Flash Player 3.

### Sintaxis

```
stopAllSounds()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; detiene todos los sonidos que se están reproduciendo actualmente en un archivo SWF sin detener la cabeza lectora. Los sonidos establecidos en flujo reanudarán la reproducción cuando la cabeza lectora se mueva sobre los fotogramas en los que se encuentran.

### Ejemplo

El código siguiente podría aplicarse a un botón que, cuando se hace clic sobre él, detiene todos los sonidos del archivo SWF.

```
on(release) {  
    stopAllSounds();  
}
```

### Véase también

[Clase Sound](#)

## stopDrag()

### Disponibilidad

Flash Player 4.

### Sintaxis

```
stopDrag()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; detiene la operación de arrastre actual.

### Ejemplo

Esta sentencia detiene la acción de arrastre en la instancia `mc_mc` cuando el usuario suelta el botón del ratón.

```
on(press){  
    startDrag("my_mc");  
}  
on(release) {  
    stopdrag();  
}
```

### Véase también

[MovieClip.\\_droptarget](#), [MovieClip.stopDrag\(\)](#), [startDrag\(\)](#)

## " " (delimitador de cadena)

### Disponibilidad

Flash Player 4.

### Sintaxis

```
"text"
```

### Parámetros

*text*    Un carácter.

### Valor devuelto

Ninguno.

### Descripción

Delimitador de cadena; cuando se utilizan delante y detrás de caracteres, las comillas indican que los caracteres tienen un valor literal y se consideran un *string*, no una variable, valor numérico ni ningún otro elemento de ActionScript.

## Ejemplo

En este ejemplo se utilizan comillas para indicar que el valor de la variable *yourGuess* es la cadena literal string “Isla Príncipe Eduardo” y no el nombre de una variable. El valor de *province* es una variable, no un literal; para determinar el valor de *province*, debe localizarse el valor de *yourGuess*.

```
yourGuess = "Isla Príncipe Eduardo";
on(release) {
    province = yourGuess;
    trace(province);
}

// Muestra Isla Príncipe Eduardo en el panel Salida
```

## Véase también

[Clase String](#), [String\(\)](#)

# Clase String

## Disponibilidad

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).

## Descripción

La clase String es un envoltorio para el tipo de datos primitivo de cadena y proporciona métodos y propiedades que permiten manipular tipos de valores de cadena primitivos. Puede convertir el valor de cualquier objeto en una cadena mediante la función `String()`.

Todos los métodos de la clase String, excepto `concat()`, `fromCharCode()`, `slice()` y `substr()`, son genéricos. Esto quiere decir que los métodos propiamente dichos llaman a `this.toString()` antes de realizar sus operaciones; puede utilizar estos métodos con otros objetos que no sean String.

Puesto que todos los índices de cadena tienen base cero, el índice del último carácter de cualquier cadena `x` es `x.length - 1`.

Puede llamar a cualquiera de los métodos de la clase String utilizando el método constructor `newString` o un valor de literal de cadena. Si especifica un literal de cadena, el intérprete de ActionScript lo convierte automáticamente en un objeto String temporal, llama al método y después descarta el objeto String temporal. Puede utilizar la propiedad `String.length` con un literal de cadena.

No debe confundirse un literal de cadena con un objeto String. En el ejemplo siguiente, la primera línea de código crea el literal de cadena `s1` y la segunda línea de código crea el objeto String `s2`.

```
s1 = "foo"
s2 = new String("foo")
```

Utilice literales de cadena a menos que necesite utilizar específicamente un objeto String.

## Resumen de métodos para la clase String

Método	Descripción
<code>String.charAt()</code>	Devuelve el carácter que se encuentra en una ubicación específica de una cadena.
<code>String.charCodeAt()</code>	Devuelve el valor del carácter en el índice especificado como un número entero de 16 bits entre 0 y 65535.
<code>String.concat()</code>	Combina el texto de dos cadenas y devuelve una nueva cadena.
<code>String.fromCharCode()</code>	Devuelve una cadena formada por los caracteres especificados en los parámetros.
<code>String.indexOf()</code>	Devuelve la posición de la primera aparición de la subcadena especificada.
<code>String.lastIndexOf()</code>	Devuelve la posición de la última aparición de la subcadena especificada.
<code>String.slice()</code>	Extrae una sección de una cadena y devuelve una nueva cadena.
<code>String.split()</code>	Divide un objeto String en una matriz de cadenas dividiendo la cadena en subcadenas.
<code>String.substr()</code>	Devuelve un número especificado de caracteres en una cadena, empezando en la ubicación especificada.
<code>String.substring()</code>	Devuelve los caracteres que hay entre dos índices en una cadena.
<code>String.toLowerCase()</code>	Convierte la cadena a minúsculas y devuelve el resultado; no cambia el contenido del objeto original.
<code>String.toUpperCase()</code>	Convierte la cadena a mayúsculas y devuelve el resultado; no cambia el contenido del objeto original.

## Resumen de propiedades para la clase String

Propiedad	Descripción
<code>String.length</code>	Número entero que no es de base cero que indica el número de caracteres del objeto String especificado.

## Constructor para la clase String

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new String(value)
```

### Parámetros

*value* Valor inicial del nuevo objeto String.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto String.

### Véase también

`String()`, " " (delimitador de cadena)

## String.charAt()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.charAt(index)
```

### Parámetros

*index* Número entero que especifica la posición de un carácter en la cadena. El primer carácter se indica con 0, y el último, con *my\_string.length*-1.

### Valor devuelto

Un carácter.

### Descripción

Método; devuelve el carácter que se encuentra en la posición especificada por el *index* del parámetro. Si *index* no es un número entre 0 y *string.length*-1, se devuelve una cadena vacía.

Este método es parecido a `String.charCodeAt()`, pero el valor devuelto es un carácter, no un código de carácter entero de 16 bits.

### Ejemplo

En el ejemplo siguiente, se llama a este método en la primera letra de la cadena "Cristina".

```
my_str = new String("Cristina");  
i = my_str.charAt(0); // i = "C"
```

## String.charCodeAt()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.charCodeAt(index)
```

### Parámetros

*index* Número entero que especifica la posición de un carácter en la cadena. El primer carácter se indica con 0, y el último carácter, con *my\_str.length*-1.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve un entero de 16 bits entre 0 y 65535 que representa el carácter especificado por *index*. Si *index* no es un número de 0 a `string.length - 1`, se devuelve NaN.

Este método es parecido al método `String.charAt()`, pero el valor devuelto en este caso es un código de caracteres entero de 16 bits, no un carácter.

### Ejemplo

En el ejemplo siguiente, se llama a este método en la primera letra de la cadena "Cristina".

```
my_str = new String("Cristina");  
i = my_str.charCodeAt(0); // i = 67
```

## String.concat()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.concat(value1,...valueN)
```

### Parámetros

*value1,...valueN* Cero o más valores que deben concatenarse.

### Valor devuelto

Una cadena.

### Descripción

Método; combina el valor del objeto String con los parámetros y devuelve la cadena nueva; el valor original, *my\_str*, se mantiene inalterado.

## String.fromCharCode()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
String.fromCharCode(c1,c2,...cN)
```

### Parámetros

*c1,c2,...cN* Números enteros decimales que representan valores ASCII.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve una cadena formada por los caracteres representados por los valores ASCII de los parámetros.

## Ejemplo

En este ejemplo se utiliza `fromCharCode()` para insertar un carácter @ en la dirección de correo electrónico.

```
address_str = "dog" + String.fromCharCode(64) + "house.net";  
trace(address_str); // dog@house.net
```

## String.indexOf()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.indexOf(substring, [startIndex])
```

### Parámetros

*substring* Número entero o cadena que especifica la subcadena que se debe buscar dentro de *my\_str*.

*startIndex* Número entero opcional que especifica el punto inicial en *my\_str* para buscar la subcadena.

### Valor devuelto

La posición de la primera aparición de la subcadena especificada o -1.

### Descripción

Método; busca en la cadena y devuelve la posición de la primera aparición de *substring* que se encuentre en *startIndex* o después de *startIndex* dentro de la cadena de llamada. Si no se encuentra ninguna *substring*, el método devuelve -1.

### Véase también

[String.lastIndexOf\(\)](#)

## String.lastIndexOf()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.lastIndexOf(substring, [startIndex])
```

### Parámetros

*substring* Número entero o cadena que especifica la cadena que se va a buscar.

*startIndex* Número entero opcional que especifica el punto en el que debe iniciarse la búsqueda de *substring*.

### Valor devuelto

La posición de la última aparición de la subcadena especificada o -1.

### Descripción

Método; busca en la cadena de derecha a izquierda y devuelve el índice de la última aparición de *substring* que se encuentra antes de *startIndex* dentro de la cadena de llamada. Si no se encuentra ninguna *substring*, el método devuelve -1.

### Véase también

`String.indexOf()`

## String.length

### Disponibilidad

Flash Player 5.

### Sintaxis

`my_str.length`

### Descripción

Propiedad; número entero que no es de base cero que indica el número de caracteres del objeto String especificado.

Puesto que todos los índices de cadena tienen base cero, el índice del último carácter de cualquier cadena *x* es *x.length - 1*.

## String.slice()

### Disponibilidad

Flash Player 5.

### Sintaxis

`my_str.slice(start, [end])`

### Parámetros

*start* Número que especifica el índice del punto de inicio de la sección. Si *start* es un número negativo, el punto de inicio se determina desde el final de la cadena, donde -1 es el último carácter.

*end* Número que es 1+ el índice del punto final de la sección. El carácter indexado por el parámetro *end* no se incluye en la cadena extraída. Si el parámetro se omite, se utiliza *String.length*. Si *end* es un número negativo, el punto final se determina contando desde el final de la cadena, donde -1 es el último carácter.

### Valor devuelto

Una subcadena de la cadena especificada.

### Descripción

Método; devuelve una cadena que incluye el carácter *start* y todos los caracteres hasta el carácter *end*, pero sin incluir este último. El objeto String original no se modifica. Si no se especifica el parámetro *end*, el final de la subcadena es el final de la cadena. Si el valor de *start* es mayor o igual que el valor de *end*, el método devuelve una cadena vacía.



## Ejemplo

En el ejemplo siguiente se establece una variable, `text`, se crea un objeto `String`, `my_str`, y se pasa a la variable `text`. El método `slice()` extrae una sección de la cadena que se encuentra en la variable y `trace()` la envía al panel Salida. En el ejemplo se muestra el uso de un valor positivo y el de un valor negativo para el parámetro `end`.

```
text = "Lexington";
my_str = new String( text );
trace(my_str.slice( 1, 3 )); // "ex"
trace(my_str.slice( 1, -6 )); // "ex"
```

## Véase también

[String.substr\(\)](#), [String.substring\(\)](#)

# String.split()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_str.split("delimiter", [limit])
```

## Parámetros

*delimiter*    Carácter o cadena por donde se divide *my\_str*.

*limit*        Número de elementos que deben colocarse en la matriz. Este parámetro es opcional.

## Valor devuelto

Una matriz que contiene las subcadenas de *my\_str*.

## Descripción

Método; divide un objeto `String` en subcadenas separándolo en el punto donde aparece el parámetro *delimiter* especificado y devuelve las subcadenas en una matriz. Si se utiliza una cadena vacía ("" ) como delimitador, cada carácter de la cadena se coloca como un elemento en la matriz, tal como se muestra en el código siguiente.

```
my_str = "Joe";
i = my_str.split("");
trace (i);
```

El panel Salida muestra lo siguiente:

J,o,e

Si el parámetro *delimiter* no está definido, toda la cadena se coloca en el primer elemento de la matriz devuelta.

## Ejemplo

En el ejemplo siguiente se devuelve una matriz con cinco elementos.

```
my_str = "P, A, T, S, Y";
my_str.split(",");
```

Este ejemplo devuelve una matriz con dos elementos, "P" y "A".

```
my_str.split(",", 2);
```

## String.substr()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.substr(start, [length])
```

### Parámetros

*start* Número entero que indica la posición del primer carácter de *my\_str* que debe utilizarse para crear la subcadena. Si *start* es un número negativo, el punto inicial se determina desde el final de la cadena, donde -1 es el último carácter.

*length* Número de caracteres de la subcadena que se está creando. Si no se especifica el valor *length*, la subcadena incluye todos los caracteres desde el inicio hasta el final de la cadena.

### Valor devuelto

Una subcadena de la cadena especificada.

### Descripción

Método; devuelve los caracteres de una cadena desde el índice especificado en el parámetro *start*, hasta el número de caracteres especificados en el parámetro *length*. El método `substr` no modifica la cadena especificada por *my\_str*, sino que devuelve una nueva cadena.

## String.substring()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.substring(start, [end])
```

### Parámetros

*start* Número entero que indica la posición del primer carácter de *my\_str* utilizado para crear la subcadena. Los valores válidos para *start* van de 0 a `String.length - 1`. Si *start* es un valor negativo, se utiliza 0.

*end* Número entero que es 1+ el índice del último carácter de *my\_str* que debe extraerse. Los valores válidos para *end* van de 1 a `String.length`. El carácter indexado por el parámetro *end* no se incluye en la cadena extraída. Si el parámetro se omite, se utiliza `String.length`. Si este parámetro es un valor negativo, se utiliza 0.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve una cadena formada por los caracteres situados entre los puntos especificados por los parámetros *start* y *end*. Si no se especifica el parámetro *end*, el final de la subcadena es el final de la cadena. Si el valor de *start* es igual al de *end*, el método devuelve una cadena vacía. Si el valor de *start* es mayor que el valor de *end*, los parámetros se intercambian automáticamente antes de que la función se ejecute y el valor original permanece inalterado.

## String.toLowerCase()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.toLowerCase()
```

### Parámetros

Ninguno.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve una copia del objeto String, con todos los caracteres en mayúsculas convertidos en minúsculas. El valor original permanece inalterado.

## String.toUpperCase()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_str.toUpperCase()
```

### Parámetros

Ninguno.

### Valor devuelto

Una cadena.

### Descripción

Método; devuelve una copia del objeto String, con todos los caracteres en minúsculas convertidos en mayúsculas. El valor original permanece inalterado.

## String()

### Disponibilidad

Flash Player 4; comportamiento modificado en Flash Player 7.

## Sintaxis

`String(expression)`

## Parámetros

*expression* Expresión que debe convertirse en una cadena.

## Valor devuelto

Una cadena.

## Descripción

Función; devuelve una representación de cadena del parámetro especificado como se muestra a continuación:

Si *expression* es un número, la cadena devuelta es una representación textual del número.

Si *expression* es una cadena, la cadena devuelta es *expression*.

Si *expression* es un objeto, el valor devuelto es una representación de cadena del objeto generada por la llamada a la propiedad `string` del objeto o por la llamada a `Object.toString()`, si no existe dicha propiedad.

Si *expression* es `undefined`, los valores devueltos son los que se indican a continuación:

- En los archivos publicados para Flash Player 6 o anterior, el resultado es una cadena vacía (`""`).
- En los archivos publicados para Flash Player 7 o posterior, el resultado es `undefined`.

Si *expression* es un valor booleano, la cadena devuelta es `"true"` o `"false"`.

Si *expression* es un clip de película, el valor devuelto es la ruta de destino del clip de película en notación con barras (`/`).

**Nota:** ActionScript 2.0 no admite la notación con barras.

## Véase también

`Number.toString()`, `Object.toString()`, [Clase String](#), `" "` (delimitador de cadena)

# substring

## Disponibilidad

Flash Player 4. Esta función se ha eliminado y se ha sustituido por `String.substr()`.

## Sintaxis

`substring("string", index, count)`

## Parámetros

*string* Cadena a partir de la que se va a extraer la nueva cadena.

*index* Número del primer carácter que se va a extraer.

*count* Número de caracteres que se van a incluir en la cadena extraída, sin incluir el carácter de índice.

## Valor devuelto

Ninguno.

## Descripción

Función de cadena; extrae parte de una cadena. Esta función tiene base 1, mientras que los métodos del objeto String tienen base 0.

## Véase también

`String.substr()`

# super

## Disponibilidad

Flash Player 6.

## Sintaxis

```
super.method([arg1, ..., argN])  
super([arg1, ..., argN])
```

## Parámetros

*method* Método que se invoca en la superclase.

*arg1* Parámetros opcionales que se pasan a la versión de superclase del método (sintaxis 1) o a la función constructor de la superclase (sintaxis 2).

## Valor devuelto

Ambas formas invocan una función. La función puede devolver cualquier valor.

## Descripción

Operador: el primer estilo de sintaxis puede utilizarse dentro del cuerpo de un método de objeto para invocar la versión de superclase de un método y puede pasar parámetros (*arg1 ... argN*) de forma opcional al método de superclase. Esto resulta útil para crear métodos de subclase que agregan un comportamiento adicional a los métodos de superclase, pero también invocan los métodos de superclase para realizar su comportamiento original.

El segundo estilo de sintaxis puede utilizarse dentro del cuerpo de una función constructora para invocar la versión de superclase de la función constructora y, opcionalmente, puede pasarle parámetros. Esto resulta útil para crear una subclase que realice una inicialización adicional, pero que también invoque el constructor de superclase para llevar a cabo una inicialización de superclase.

# switch

## Disponibilidad

Flash Player 4.

## Sintaxis

```
switch (expression){  
    caseClause:  
    [defaultClause:]  
}
```

## Parámetros

*expression* Cualquier expresión.

*caseClause* Palabra clave `case` seguida por una expresión, dos puntos y un grupo de sentencias que se deben ejecutar si la expresión coincide con el parámetro *expression* de `switch` que utiliza la igualdad estricta (`===`).

*defaultClause* Palabra clave `default` seguida de sentencias que se deben ejecutar si ninguna de las expresiones `case` coincide con el parámetro *expression* de `switch` según la igualdad estricta (`===`).

## Valor devuelto

Ninguno.

## Descripción

Sentencia; crea una estructura ramificada para sentencias de `ActionScript`. Al igual que la acción `if`, la acción `switch` prueba una condición que ejecuta sentencias si la condición devuelve el valor `true`.

## Ejemplo

En el ejemplo siguiente, si el parámetro `number` da como resultado 1, se ejecuta la acción `trace()` que sigue a `case 1`; si el parámetro `number` da como resultado 2, se ejecuta la acción `trace()` que sigue a `case 2`, y así sucesivamente. Si ninguna expresión `case` coincide con el parámetro `number`, se ejecuta la acción `trace()` que sigue a la palabra clave `default`.

```
switch (number) {  
    case 1:  
        trace ("case 1 es verdadero");  
        break;  
    case 2:  
        trace ("case 2 es verdadero");  
        break;  
    case 3:  
        trace ("case 3 es verdadero");  
        break;  
    default:  
        trace ("ningún case es verdadero")  
}
```

En el ejemplo siguiente, no hay ninguna sentencia `break` en el grupo del primer caso, de modo que si el parámetro `number` es 1, tanto A como B se envían al panel Salida:

```
switch (number) {  
    case 1:  
        trace ("A");  
    case 2:  
        trace ("B");  
        break;  
    default:  
        trace ("D")  
}
```

## Véase también

`===` (igualdad estricta), `break`, `case`, `default`, `if`

# Clase System

## Disponibilidad

Flash Player 6.

## Descripción

Se trata de una clase de nivel superior que contiene el objeto de capacidades (véase [Objeto System.capabilities](#)), el objeto de seguridad (véase [Objeto System.security](#)) y los métodos, propiedades y controladores de eventos que se enumeran a continuación.

## Resumen de métodos para la clase System

Método	Descripción
<a href="#">System.setClipboard()</a>	Sustituye el contenido del portapapeles del sistema por una cadena de texto.
<a href="#">System.showSettings()</a>	Muestra un panel Configuración de Flash Player.

## Resumen de propiedades para la clase System

Método	Descripción
<a href="#">System.exactSettings</a>	Especifica si debe utilizarse las reglas de coincidencia de superdominio o de dominio exacto cuando se accede a la configuración local.
<a href="#">System.useCodepage</a>	Indica a Flash Player si debe utilizar Unicode o la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor para interpretar los archivos de texto externos.

## Resumen de controladores de eventos para la clase System

Método	Descripción
<a href="#">System.onStatus</a>	Proporciona un súper controlador de eventos para ciertos objetos

## System.exactSettings

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

`System.exactSettings`

## Descripción

Propiedad; especifica si debe utilizarse las reglas de coincidencia de superdominio o de dominio exacto cuando se accede a la configuración local (como los permisos de acceso a la cámara o al micrófono) o a los datos persistentes locales (objetos compartidos). El valor predeterminado es `true` para los archivos publicados para Flash Player 7 o versiones posteriores y `false` para los archivos publicados para Flash Player 6.

Si el valor es `true`, la configuración y los datos de un archivo SWF albergado en `here.xyz.com` se almacenan en `here.xyz.com`, la configuración y los datos de un archivo SWF albergado en `there.xyz.com` se almacenan en `there.xyz.com`, etc. Si este valor es `false`, la configuración y los datos para los archivos SWF albergados en `here.xyz.com`, `there.xyz.com` y `xyz.com` se comparten y se almacenan todos en `xyz.com`.

Si alguno de sus archivos establece esta propiedad en `false` y otros en `true`, puede suceder que archivos SWF de distintos subdominios compartan configuración y datos. Por ejemplo, si esta propiedad es `false` en un archivo SWF albergado en `here.xyz.com` y `true` en un archivo SWF albergado en `xyz.com`, ambos archivos utilizarán la misma configuración y los mismos datos, es decir, los de `xyz.com`. Si este no es el comportamiento deseado, asegúrese de que establece esta propiedad en cada archivo para representar correctamente dónde quiere almacenar la configuración y los datos.

Si desea cambiar el valor de esta propiedad para que no sea el predeterminado, emita el comando `System.exactSettings = false` en el primer fotograma del documento. La propiedad no se puede cambiar después de cualquier actividad que necesite acceso a la configuración local, como `System.ShowSettings()` o `SharedObject.getLocal()`.

Si utiliza `loadMovie()`, `MovieClip.loadMovie()` o `MovieClipLoader.loadClip()` para cargar un archivo SWF en otro, todos los archivos publicados para Flash Player 7 comparten un solo valor para `System.exactSettings` y todos los archivos publicados para Flash Player 6 comparten un solo valor para `System.exactSettings`. Por lo tanto, si especifica un valor para esta propiedad en un archivo publicado para una versión concreta de Player, debe hacerlo en todos los archivos que vaya a cargar. Si carga varios archivos, la configuración especificada en el último archivo cargado sobrescribe cualquier configuración especificada anteriormente.

Para más información sobre cómo se implementa la coincidencia entre dominios en Flash, consulte [“Funciones de seguridad de Flash Player” en la página 196](#).

## Véase también

`SharedObject.getLocal()`, `System.showSettings()`

# System.onStatus

## Disponibilidad

Flash Player 6.

## Descripción

Controlador de eventos: proporciona un "súper" controlador de eventos para ciertos objetos.



Los objetos `LocalConnection`, `NetStream` y `SharedObject` proporcionan un controlador de eventos `onStatus` que utiliza un objeto de información para proporcionar mensajes de información, de estado o de error. Para responder a este controlador de eventos, debe crear una función para procesar el objeto de información, y debe saber qué formato y contenido ha devuelto el objeto.

Además de los métodos `onStatus` específicos proporcionados para los objetos que se enumeran más arriba, Flash también proporciona una "súper" función denominada `System.onStatus`. Si se invoca `onStatus` para un objeto determinado con una propiedad `level` de "error" y no existe ninguna función asignada para responderle, Flash procesa una función asignada a `System.onStatus` si existe.

**Nota:** las clases `Camera` y `Microphone` también disponen de controladores `onStatus`, pero no pasan objetos de información con una propiedad `level` de "error". Por lo tanto, no se invoca `System.onStatus` si no especifica una función para estos controladores.

En el ejemplo siguiente se muestra cómo puede crear funciones genéricas y específicas para procesar objetos de información enviados con el método `onStatus`.

```
// Cree una función genérica
System.onStatus = function(genericError)
{
    // El script debe hacer algo más significativo aquí
    trace("Se ha producido un error. Inténtelo de nuevo.");
}

// Cree la función para el objeto NetStream
// Si el objeto NetStream devuelve un objeto de información diferente
// del que se enumera a continuación, con una propiedad level de "error",
// se invoca System.onStatus

videoStream_ns.onStatus = function(infoObject) {
    if (infoObject.code == "NetStream.Play.StreamNotFound") {
        trace("No se encuentra el archivo de vídeo.");
    }
}
```

#### Véase también

[Camera.onStatus](#), [LocalConnection.onStatus](#), [Microphone.onStatus](#),  
[NetStream.onStatus](#), [SharedObject.onStatus](#)

## System.setClipboard()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

`System.setClipboard(string)`

### Parámetros

*string* Una cadena de caracteres de texto plano para colocar en el portapapeles, que sustituye su contenido actual (si existiese). Si pasa un literal de cadena en lugar de una variable de tipo String, escriba el literal entre comillas.

### Valor devuelto

El valor booleano `true` si el texto se ha colocado correctamente en el portapapeles, o `false` en caso contrario.

### Descripción

Método; sustituye el contenido del portapapeles del sistema por una cadena de texto especificada.

## System.showSettings()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.showSettings([panel])
```

### Parámetros

*panel* Número opcional que especifica qué panel Configuración de Flash Player debe mostrarse, como se indica en la tabla siguiente.

Valor pasado para <i>panel</i>	Panel de configuración que se visualiza
Ninguno (el parámetro se omite) o un valor no admitido	El panel que estaba abierto la última vez que el usuario cerró el panel de configuración de Player
0	Privacy
1	Almacenamiento local
2	Microphone
3	Camera

### Valor devuelto

Ninguno.

### Descripción

Método; muestra el panel Configuración de Flash Player especificado, que permite a los usuarios seguir uno de estos procedimientos:

- Permitir o denegar el acceso a la cámara y al micrófono
- Especificar el espacio en disco local disponible para los objetos compartidos
- Seleccionar una cámara y un micrófono predeterminados
- Especificar los valores de ganancia de micrófono y supresión de ecos

Por ejemplo, si la aplicación requiere el uso de la cámara, puede indicar al usuario que seleccione Permitir en el panel de configuración de confidencialidad y, a continuación, emitir el comando `System.showSettings(0)`. Asegúrese de que el tamaño del escenario sea como mínimo de 215 x 138 píxeles, que es el tamaño mínimo necesario para visualizar el panel en Flash.

#### Véase también

`Camera.get()`, `Microphone.get()`, `SharedObject.getLocal()`

## System.useCodepage

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.useCodepage`

### Descripción

Propiedad; valor booleano que indica a Flash Player si debe utilizar Unicode o la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor para interpretar los archivos de texto externos. El valor predeterminado de `system.useCodepage` es `false`.

- Cuando la propiedad se establece en `false`, Flash Player interpreta los archivos de texto externos como Unicode. Estos archivos deben codificarse como Unicode al guardarlos.
- Cuando la propiedad se establece en `true`, Flash Player interpreta los archivos de texto externos utilizando la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor.

El texto que se incluye o se carga como archivo externo (mediante el comando `#include`, las acciones `loadVariables()` o `getURL`, o los objetos `LoadVars` o `XML`) debe codificarse como Unicode al guardar el archivo de texto para que Flash Player pueda reconocerlo como Unicode. Para codificar archivos externos como Unicode, guarde los archivos en una aplicación que admita Unicode, como el Bloc de notas de Windows 2000.

Si incluye o carga archivos de texto externos cuya codificación no es Unicode, debe establecer `system.useCodepage` en `true`. Añada el código siguiente como primera línea del código del primer fotograma del archivo SWF que carga los datos:

```
system.useCodepage = true;
```

Cuando este código está presente, Flash Player interpreta el texto externo mediante la página de códigos tradicional del sistema operativo en el que se ejecuta Flash Player. Generalmente es CP1252 para un sistema operativo Windows en inglés y Shift-JIS para un sistema operativo en japonés. Si establece `system.useCodepage` en `true`, Flash Player 6 y las versiones posteriores tratan el texto del mismo modo que Flash Player 5. Flash Player 5 trataba todo el texto como si estuviese en la página de códigos tradicional del sistema operativo en el que se ejecutaba el reproductor.

Si establece `system.useCodepage` en `true`, debe tener en cuenta que la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor debe incluir los caracteres que se utilizan en el archivo de texto externo para que el texto pueda visualizarse. Por ejemplo, si carga un archivo de texto externo que contiene caracteres chinos, dichos caracteres no se visualizarán en un sistema que utilice la página de códigos CP1252, ya que dicha página de códigos no contiene caracteres chinos.

Para asegurarse de que los usuarios de todas las plataformas pueden ver los archivos de texto externos que se utilizan en sus archivos SWF, debe codificar todos los archivos de texto externos como Unicode y dejar la propiedad `System.useCodepage` establecida en `false` de forma predeterminada. De este modo, Flash Player 6 y las versiones posteriores interpretarán el texto como Unicode.

## Objeto `System.capabilities`

### Disponibilidad

Flash Player 6.

### Descripción

Puede utilizar el objeto `System.capabilities` para determinar las capacidades del sistema y el reproductor que albergan un archivo SWF. Esto le permite personalizar contenido para distintos formatos. Por ejemplo, la pantalla de un teléfono móvil (en blanco y negro, de 100 píxeles cuadrados) es distinta a la pantalla de un PC en color de 1000 píxeles cuadrados. Para proporcionar el contenido adecuado a todos los usuarios posibles, puede utilizar el objeto `System.capabilities` para determinar el tipo de dispositivo que tiene un usuario. A continuación, puede especificar al servidor que envíe distintos archivos SWF según las capacidades del dispositivo, o indicar al archivo SWF que modifique su presentación en función de las capacidades del dispositivo.

Puede enviar información sobre las capacidades utilizando un método HTTP GET o POST. A continuación se muestra un ejemplo de una cadena de servidor para un dispositivo no compatible con MP3 y con una pantalla de 400 x 200 píxeles (8 x 4 centímetros):

```
"A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DEB=t&V=WIN%207%2C0%2C0%2C226&M=Macromedia%20Windows&R=1152x864&DP=72&COL=color&AR=1.0&OS=Windows%20XP&L=en&PT=External&AVD=f&LFD=f"
```

## Resumen de propiedades para el objeto `System.capabilities`

Propiedad	Descripción	Cadena de servidor
<code>System.capabilities.avHardwareDisable</code>	Sólo lectura; especifica si la cámara y el micrófono del usuario están activados o desactivados.	AVD
<code>System.capabilities.hasAccessibility</code>	Indica si el reproductor se ejecuta en un sistema que admite la comunicación entre Flash Player y las ayudas de accesibilidad.	ACC

Propiedad	Descripción	Cadena de servidor
<code>System.capabilities.hasAudio</code>	Indica si el reproductor se ejecuta en un sistema que tiene capacidades de audio.	A
<code>System.capabilities.hasAudioEncoder</code>	Indica si el reproductor se ejecuta en un sistema que puede codificar un flujo de audio, como el que proviene de un micrófono.	AE
<code>System.capabilities.hasEmbeddedVideo</code>	Indica si el reproductor se ejecuta en un sistema que admite vídeo incorporado.	EV
<code>System.capabilities.hasMP3</code>	Indica si el reproductor se ejecuta en un sistema que tiene un decodificador MP3.	MP3
<code>System.capabilities.hasPrinting</code>	Indica si el reproductor se ejecuta en un sistema que admite impresión.	PR
<code>System.capabilities.hasScreenBroadcast</code>	Indica si el reproductor admite el desarrollo de aplicaciones de difusión en pantalla que deban ejecutarse a través de Flash Communication Server.	SB
<code>System.capabilities.hasScreenPlayback</code>	Indica si el reproductor admite la reproducción de aplicaciones de difusión en pantalla que se ejecutan a través de Flash Communication Server.	SP
<code>System.capabilities.hasStreamingAudio</code>	Indica si el reproductor puede reproducir flujo de audio.	SA
<code>System.capabilities.hasStreamingVideo</code>	Indica si el reproductor puede reproducir flujo de vídeo.	SV
<code>System.capabilities.hasVideoEncoder</code>	Indica si el reproductor puede codificar un flujo de vídeo, como el que proviene de una webcam.	VE
<code>System.capabilities.isDebugger</code>	Indica si el reproductor es una versión oficial o una versión de depuración especial.	DEB
<code>System.capabilities.language</code>	Indica el idioma del sistema en el que se ejecuta el reproductor.	L
<code>System.capabilities.localFileReadDisable</code>	Sólo lectura; especifica si el reproductor intentará leer datos (incluido el primer archivo SWF con el que se abra el reproductor) del disco duro del usuario.	LFD
<code>System.capabilities.manufacturer</code>	Indica el fabricante de Flash Player.	M
<code>System.capabilities.os</code>	Indica el sistema operativo que alberga Flash Player.	OS
<code>System.capabilities.pixelAspectRatio</code>	Indica la relación de aspecto de los píxeles de la pantalla.	AR

Propiedad	Descripción	Cadena de servidor
<code>System.capabilities.playerType</code>	Indica el tipo de reproductor: independiente, externo, plug-in o ActiveX.	PT
<code>System.capabilities.screenColor</code>	Indica si la pantalla es en color, escala de grises o blanco y negro.	COL
<code>System.capabilities.screenDPI</code>	Indica la resolución de la pantalla de puntos por pulgada, expresada en píxeles.	DP
<code>System.capabilities.screenResolutionX</code>	Indica el tamaño horizontal de la pantalla.	R
<code>System.capabilities.screenResolutionY</code>	Indica el tamaño vertical de la pantalla.	R
<code>System.capabilities.serverString</code>	Cadena en formato URL codificado que especifica los valores para cada propiedad <code>System.capabilities</code> .	No disponible
<code>System.capabilities.version</code>	Una cadena que contiene información sobre la versión de Flash Player y la plataforma.	V

## System.capabilities.avHardwareDisable

### Disponibilidad

Flash Player 7.

### Sintaxis

`System.capabilities.avHardwareDisable`

### Descripción

Propiedad de sólo lectura; valor booleano que especifica si la cámara y el micrófono del usuario están activados o desactivados.

### Véase también

`Camera.get()`, `Microphone.get()`, `System.showSettings()`

## System.capabilities.hasAccessibility

### Disponibilidad

Flash Player 6 versión 65.

### Sintaxis

`System.capabilities.hasAccessibility`

### Descripción

Propiedad; valor booleano que indica si el reproductor se ejecuta en un entorno que admite la comunicación entre Flash Player y las ayudas de accesibilidad. La cadena del servidor es ACC.

**Véase también**

[Accessibility.isActive\(\)](#), [Accessibility.updateProperties\(\)](#), [\\_accProps](#)

## System.capabilities.hasAudio

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasAudio`

### Descripción

Propiedad; valor booleano que indica si el reproductor se ejecuta en un sistema que tiene capacidades de audio. La cadena del servidor es A.

## System.capabilities.hasAudioEncoder

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasAudioEncoder`

### Descripción

Propiedad; valor booleano que indica si el reproductor puede codificar un flujo de audio, como el que procede de un micrófono. La cadena del servidor es AE.

## System.capabilities.hasEmbeddedVideo

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasEmbeddedVideo`

### Descripción

Propiedad; valor booleano que indica si el reproductor se ejecuta en un sistema que admite vídeo incorporado. La cadena del servidor es EV.

## System.capabilities.hasMP3

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasMP3`

### Descripción

Propiedad; valor booleano que indica si el reproductor se ejecuta en un sistema que tiene un decodificador MP3. La cadena del servidor es MP3.

## System.capabilities.hasPrinting

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.hasPrinting
```

### Descripción

Propiedad; valor booleano que indica si el reproductor se ejecuta en un sistema que admite impresión. La cadena del servidor es PR.

## System.capabilities.hasScreenBroadcast

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.hasScreenBroadcast
```

### Descripción

Propiedad; valor booleano que indica si el reproductor admite el desarrollo de aplicaciones de difusión en pantalla que deban ejecutarse a través de Flash Communication Server. La cadena del servidor es SB.

## System.capabilities.hasScreenPlayback

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.hasScreenPlayback
```

### Descripción

Propiedad; valor booleano que indica si el reproductor admite la reproducción de aplicaciones de difusión en pantalla que se ejecutan a través de Flash Communication Server. La cadena de servidor es SP.

## System.capabilities.hasStreamingAudio

### Disponibilidad

Flash Player 6.



### Sintaxis

`System.capabilities.hasStreamingAudio`

### Descripción

Propiedad; valor booleano que indica si el reproductor puede reproducir flujo de audio. La cadena de servidor es SA.

## System.capabilities.hasStreamingVideo

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasStreamingVideo`

### Descripción

Propiedad; valor booleano que indica si el reproductor puede reproducir flujo de vídeo. La cadena de servidor es SV.

## System.capabilities.hasVideoEncoder

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.hasVideoEncoder`

### Descripción

Propiedad; valor booleano que indica si el reproductor puede codificar flujo de vídeo, como el que proviene de una webcam. La cadena del servidor es VE.

## System.capabilities.isDebugger

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.isDebugger`

### Descripción

Propiedad; valor booleano que indica si el reproductor es una versión oficial (`false`) o una versión de depuración especial (`true`). La cadena del servidor es DEB.

## System.capabilities.language

### Disponibilidad

Flash Player 6.

## Sintaxis

`System.capabilities.language`

## Descripción

Propiedad; indica el idioma del sistema en el que se ejecuta el reproductor. Esta propiedad se especifica como un código de idioma de dos letras minúsculas perteneciente a ISO 639-1 y una subetiqueta de código de país de dos letras mayúsculas opcional perteneciente a ISO 3166. Estos códigos representan el idioma del sistema en el que se ejecuta el reproductor. Los propios idiomas se denominan con las etiquetas en inglés. Por ejemplo, “fr” especifica el idioma francés.

Idioma	Etiqueta	Etiquetas y países admitidos
Checo	cs	
Danés	da	
Holandés	nl	
Inglés	en	
Finlandés	fi	
Francés	fr	
Alemán	de	
Húngaro	hu	
Italiano	it	
Japonés	ja	
Coreano	ko	
Noruego	no	
Otro/desconocido	xu	
Polaco	pl	
Portugués	pt	
Ruso	ru	
Chino simplificado	zh	República Popular de China (chino simplificado): zh-CN
Español	es	
Sueco	sv	
Chino tradicional	zh	Taiwán (chino tradicional): zh-TW
Turco	tr	

## System.capabilities.localFileReadDisable

### Disponibilidad

Flash Player 7.

### Sintaxis

`System.capabilities.localFileReadDisable`

### Descripción

Propiedad de sólo lectura; valor booleano que especifica si Flash Player intenta leer datos (incluido el primer archivo SWF con el que se abra Flash Player) del disco duro del usuario.

## System.capabilities.manufacturer

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.manufacturer
```

### Descripción

Propiedad; cadena que indica el fabricante de Flash Player, con el formato "Macromedia *OSName*" (*OSName* podría ser "Windows", "Macintosh", "Linux" o bien "Other OS Name"). La cadena del servidor es M.

## System.capabilities.os

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.os
```

### Descripción

Propiedad; cadena que indica el sistema operativo actual. La propiedad *os* puede devolver las cadenas siguientes: "Windows XP", "Windows 2000", "Windows NT", "Windows 98/ME", "Windows 95", "Windows CE" (disponible únicamente en Flash Player SDK, no en la versión de escritorio), "Linux" y "MacOS". La cadena del servidor es OS.

## System.capabilities.pixelAspectRatio

### Disponibilidad

Flash Player 6.

### Sintaxis

```
System.capabilities.pixelAspectRatio
```

### Descripción

Propiedad; un entero que indica la relación de aspecto de los píxeles de la pantalla. La cadena de servidor es AR.

## System.capabilities.playerType

### Disponibilidad

Flash Player 7.

**Sintaxis**

`System.capabilities.playerType`

**Descripción**

Propiedad; cadena que indica el tipo de reproductor. Esta propiedad puede tener el valor "StandAlone", "External", "PlugIn" o "ActiveX". La cadena de servidor es PT.

## System.capabilities.screenColor

**Disponibilidad**

Flash Player 6.

**Sintaxis**

`System.capabilities.screenColor`

**Descripción**

Propiedad; indica si la pantalla es en color (color), escala de grises (gray) o blanco y negro (bw). La cadena de servidor es COL.

## System.capabilities.screenDPI

**Disponibilidad**

Flash Player 6.

**Sintaxis**

`System.capabilities.screenDPI`

**Descripción**

Propiedad; indica la resolución de puntos por pulgada (ppp) de la pantalla, expresada en píxeles. La cadena de servidor es DP.

## System.capabilities.screenResolutionX

**Disponibilidad**

Flash Player 6.

**Sintaxis**

`System.capabilities.screenResolutionX`

**Descripción**

Propiedad; un entero que indica la resolución horizontal máxima de la pantalla. La cadena de servidor es R (que devuelve la anchura y la altura de la pantalla).

## System.capabilities.screenResolutionY

**Disponibilidad**

Flash Player 6.

### Sintaxis

`System.capabilities.screenResolutionY`

### Descripción

Propiedad; un entero que indica la resolución vertical máxima de la pantalla. La cadena de servidor es R (que devuelve la anchura y la altura de la pantalla).

## System.capabilities.serverString

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.serverString`

### Descripción

Propiedad; cadena con formato URL codificado que especifica los valores para cada propiedad `System.capabilities`, como en este ejemplo:

```
A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DEB=t&V=WIN%207%2C0%2C0%2C226&M=Macromedia%20Windows&R=1152x864&DP=72&COL=color&AR=1.0&OS=Windows%20XP&L=en&PT=External&AVD=f&LFD=f
```

## System.capabilities.version

### Disponibilidad

Flash Player 6.

### Sintaxis

`System.capabilities.version`

### Descripción

Propiedad; cadena que contiene información sobre la plataforma y la versión de Flash Player; por ejemplo, "WIN 7,0,0,231". La cadena de servidor es V.

## Objeto System.security

### Disponibilidad

Flash Player 6.

### Descripción

Este objeto contiene métodos que especifican cómo los archivos SWF de distintos dominios pueden comunicarse entre sí.

## Resumen de métodos para el objeto System.security

Método	Descripción
<code>System.security.allowDomain()</code>	Permite a los archivos SWF de los dominios identificados acceder a los objetos y variables del archivo SWF que efectúa la llamada, o en cualquier otro archivo SWF del mismo dominio que el archivo SWF que efectúa la llamada.
<code>System.security.allowInsecureDomain()</code>	Permite a los archivos SWF de los dominios identificados acceder a los objetos y variables del archivo SWF que efectúa la llamada, que se alberga mediante el protocolo HTTPS.

### System.security.allowDomain()

#### Disponibilidad

Flash Player 6; comportamiento modificado en Flash Player 7.

#### Sintaxis

```
System.security.allowDomain("domain1", "domain2", ... domainN")
```

#### Parámetros

*domain1, domain2, ... domainN* Cadenas que especifican los dominios que pueden acceder a los objetos y las variables del archivo que contiene la llamada a

`System.Security.allowDomain()`. Los dominios pueden tener los formatos siguientes:

- "domain.com"
- "http://domain.com"
- "http://IPAddress"

#### Descripción

Método; permite a los archivos SWF de los dominios identificados acceder a los objetos y variables del archivo SWF que efectúa la llamada, o en cualquier otro archivo SWF del mismo dominio que el archivo SWF que efectúa la llamada.

En los archivos que se reproducen en Flash Player 7 o en versiones posteriores, los parámetros pasados deben seguir las reglas de asignación de nombres de dominio exacto. Por ejemplo, para permitir el acceso a archivos SWF albergados en `www.domain.com` o `store.domain.com`, debe pasarse el nombre de ambos dominios:

```
// Para Flash Player 6
System.security.allowDomain("domain.com");
// Comandos correspondientes para permitir el acceso de los archivos SWF
// que se están ejecutando en Flash Player 7 o posterior
System.security.allowDomain("www.domain.com", "store.domain.com");
```

Asimismo, para los archivos que se ejecuten en Flash Player 7 o versiones posteriores, no puede utilizar este método para hacer que los archivos SWF albergados mediante un protocolo seguro (HTTPS) puedan permitir el acceso desde archivos SWF albergados en protocolos que no son seguros; en tal caso, debe utilizar `System.security.allowInsecureDomain()`.

## Ejemplo

El archivo SWF localizado en [www.macromedia.com/MovieA.swf](http://www.macromedia.com/MovieA.swf) contiene las líneas siguientes.

```
System.security.allowDomain("www.shockwave.com");  
loadMovie("http://www.shockwave.com/MovieB.swf", _root.my_mc);
```

Puesto que la película MovieA contiene el comando `allowDomain()`, la película MovieB puede acceder a los objetos y a las variables de la película MovieA. Si la película MovieA no tuviese este comando, la implementación de la seguridad de Flash evitaría que la película MovieA accediera a los objetos y a las variables de la película MovieB.

## System.security.allowInsecureDomain()

### Disponibilidad

Flash Player 7.

### Sintaxis

```
System.Security.allowInsecureDomain("domain")
```

### Parámetros

*domain* Nombre de dominio exacto, como “[www.myDomainName.com](http://www.myDomainName.com)” o “[store.myDomainName.com](http://store.myDomainName.com)”.

### Valor devuelto

Ninguno.

### Descripción

Método; permite a los archivos SWF de los dominios identificados acceder a los objetos y variables del archivo SWF que efectúa la llamada, que se alberga mediante el protocolo HTTPS.

De forma predeterminada, a los archivos SWF albergados mediante el protocolo HTTPS sólo puede acceder otro archivo SWF albergado mediante el protocolo HTTPS. Esta implementación mantiene la integridad que se proporciona mediante el protocolo HTTPS.

Macromedia no recomienda el uso de este método para suplantar el comportamiento predeterminado, ya que supone un riesgo para la seguridad de HTTPS. Sin embargo, es posible que deba hacerlo en caso de que, por ejemplo, deba permitir el acceso a archivos HTTPS publicados para Flash Player 7 o versiones posteriores desde archivos HTTP publicados para Flash Player 6.

Un archivo SWF publicado para Flash Player 6 puede utilizar `System.security.allowDomain()` para permitir el acceso HTTP a HTTPS. Sin embargo, debido a que la seguridad se implementa de forma diferente en Flash Player 7, debe utilizar `System.Security.allowInsecureDomain()` para permitir ese acceso en los archivos SWF publicados para Flash Player 7 o versiones posteriores.

## Ejemplo

En este ejemplo se alberga una prueba de matemáticas en un dominio seguro de forma que sólo pueden acceder a él los alumnos registrados. También ha desarrollado varios archivos SWF que ilustran algunos conceptos, que alberga en un dominio que no es seguro. Los alumnos deben poder acceder a la prueba desde el archivo SWF que contiene información sobre un concepto.

```
// Este archivo SWF se encuentra en https://myEducationSite.somewhere.com/  
mathTest.swf  
// Los archivos de conceptos se encuentran en http://  
myEducationSite.somewhere.com  
System.Security.allowInsecureDomain("myEducationSite.somewhere.com")
```

#### Véase también

[System.security.allowDomain\(\)](#), [System.exactSettings](#)

## targetPath

### Disponibilidad

Flash Player 5.

### Sintaxis

```
targetPath(movieClipObject)
```

### Parámetros

*movieClipObject* Referencia (por ejemplo, `_root` o `_parent`) al clip de película del que se está recuperando la ruta de destino.

### Valor devuelto

Una cadena que contiene la ruta de destino del clip de película especificado.

### Descripción

Función; devuelve una cadena que contiene la ruta de destino de *movieClipObject*. La ruta de destino se devuelve en notación con puntos. Para recuperar la ruta de destino en notación con barras, utilice la propiedad `_target`.

### Ejemplo

Este ejemplo muestra la ruta de destino de un clip de película en cuanto se carga.

```
onClipEvent (load) {  
    trace(targetPath(this));  
}
```

### Véase también

[eval\(\)](#)

## tellTarget

### Disponibilidad

Flash Player 3 (se eliminó en Flash 5; se recomienda utilizar la notación con puntos y la acción `with.`)

### Sintaxis

```
tellTarget("target") {  
    statement(s);  
}
```



## Parámetros

*target* Cadena que especifica la ruta de destino de la línea de tiempo que debe controlarse.

*statement(s)* Instrucciones que deben ejecutarse si la condición tiene el valor `true`.

## Valor devuelto

Ninguno.

## Descripción

Acción eliminada en nuevas versiones; aplica las instrucciones especificadas en el parámetro *statements* a la línea de tiempo especificada en el parámetro *target*. La acción `tellTarget` es útil para controles de navegación. Asigne `tellTarget` a botones que detienen o inician los clips de película en otras partes del escenario. También puede hacer que los clips de película vayan a un fotograma concreto de dicho clip. Por ejemplo, podría asignar `tellTarget` a botones que detienen o inician los clips de película en el escenario, o hacer que los clips de película salten a un fotograma concreto.

En Flash 5 o versiones posteriores, puede utilizar la notación con puntos en lugar de la acción `tellTarget`. Puede utilizar la acción `with` para especificar varias acciones en la misma línea de tiempo. Puede utilizar la acción `with` para usar cualquier objeto, mientras que con la acción `tellTarget` sólo puede usar clips de películas.

## Ejemplo

La sentencia `tellTarget` controla la instancia de clip de película `ball` en la línea de tiempo principal. El fotograma 1 de la instancia `ball` está vacío y tiene una acción `stop()`, de modo que es invisible en el escenario. Al hacer clic en el botón de la acción siguiente, `tellTarget` indica a la cabeza lectora de `ball` que vaya al fotograma 2, en el que empieza la animación.

```
on(release) {  
    tellTarget("ball") {  
        gotoAndPlay(2);  
    }  
}
```

En el ejemplo siguiente se utiliza la notación con puntos para obtener el mismo resultado.

```
on(release) {  
    ball.gotoAndPlay(2);  
}
```

Si necesita especificar varios comandos en la instancia `ball`, puede utilizar la acción `with`, tal como se muestra a continuación.

```
on(release) {  
    with(ball) {  
        gotoAndPlay(2);  
        _alpha = 15;  
        _xscale = 50;  
        _yscale = 50;  
    }  
}
```

## Véase también

[with](#)

# Clase TextField

## Disponibilidad

Flash Player 6.

## Descripción

Todos los campos de texto dinámico y de introducción de texto de un archivo SWF son instancias de la clase `TextField`. Puede asignar un nombre de instancia a un campo de texto en el inspector de propiedades y utilizar los métodos y las propiedades de la clase `TextField` para manipularlo con `ActionScript`. Los nombres de instancia del objeto `TextField` se muestran en el Explorador de películas y en el cuadro de diálogo Insertar ruta de destino del panel Acciones.

La clase `TextField` hereda de la [Clase Object](#).

Para crear un campo de texto dinámicamente, puede utilizar `MovieClip.createTextField()`.

## Resumen de métodos para la clase TextField

Método	Descripción
<code>TextField.addListener()</code>	Registra un objeto para recibir una notificación cuando se invocan los controladores de eventos <code>onChanged</code> y <code>onScroller</code> .
<code>TextField.getFontList()</code>	Este método devuelve los nombres de las fuentes del sistema host del reproductor en forma de matriz.
<code>TextField.getDepth()</code>	Devuelve la profundidad de un campo de texto.
<code>TextField.getNewTextFormat()</code>	Obtiene el formato de texto predeterminado asignado al texto que se acaba de insertar.
<code>TextField.getTextFormat()</code>	Devuelve un objeto <code>TextFormat</code> que contiene información sobre el formato de parte o la totalidad del texto de un campo de texto.
<code>TextField.removeListener()</code>	Elimina un objeto detector.
<code>TextField.removeTextField()</code>	Elimina un campo de texto que se ha creado con <code>MovieClip.createTextField()</code> .
<code>TextField.replaceSel()</code>	Reemplaza la selección actual.
<code>TextField.setNewTextFormat()</code>	Establece un objeto <code>TextFormat</code> para el texto insertado por un usuario o por un método.
<code>TextField.setTextFormat()</code>	Establece un objeto <code>TextFormat</code> para una parte especificada del texto de un campo de texto.

## Resumen de propiedades para la clase TextField

Propiedad	Descripción
<code>TextField._alpha</code>	Valor de transparencia de una instancia de campo de texto.
<code>TextField.autoSize</code>	Controla la alineación automática y la definición del tamaño de un campo de texto.
<code>TextField.background</code>	Indica si el campo de texto tiene un relleno de fondo.

Propiedad	Descripción
<code>TextField.backgroundColor</code>	Indica el color del relleno de fondo.
<code>TextField.border</code>	Indica si el campo de texto tiene un borde.
<code>TextField.borderColor</code>	Indica el color del borde.
<code>TextField.bottomScroll</code>	Última línea visible de un campo de texto. Es de sólo lectura.
<code>TextField.embedFonts</code>	Indica si el campo de texto utiliza contornos de fuentes incorporadas o fuentes de dispositivo.
<code>TextField._height</code>	Altura de una instancia de campo de texto en píxeles. Sólo afecta al recuadro de delimitación del campo de texto, no al grosor del borde ni al tamaño de fuente del texto.
<code>TextField._highquality</code>	Indica la calidad de representación del archivo SWF.
<code>TextField.hscroll</code>	Indica el valor de desplazamiento horizontal de un campo de texto.
<code>TextField.html</code>	Indica la posición de desplazamiento máxima de un campo de texto.
<code>TextField.htmlText</code>	Contiene la representación HTML del contenido de un campo de texto.
<code>TextField.length</code>	Número de caracteres de un campo de texto. Es de sólo lectura.
<code>TextField.maxChars</code>	Número máximo de caracteres que puede contener un campo de texto.
<code>TextField.maxhscroll</code>	Valor máximo de <code>TextField.hscroll</code> . Es de sólo lectura.
<code>TextField.maxscroll</code>	Valor máximo de <code>TextField.scroll</code> . Es de sólo lectura.
<code>TextField.menu</code>	Asocia un objeto <code>ContextMenu</code> con un campo de texto.
<code>TextField.mouseWheelEnabled</code>	Indica si Flash Player debe desplazar automáticamente el contenido de los campos de texto multilínea cuando el puntero del ratón se coloca sobre un campo de texto y el usuario hace girar la rueda del ratón.
<code>TextField.multiline</code>	Indica si el campo de texto contiene varias líneas.
<code>TextField._name</code>	Nombre de instancia de una instancia de campo de texto.
<code>TextField._parent</code>	Referencia a la instancia principal de esta instancia; ya sea de tipo <code>Button</code> o <code>MovieClip</code> .
<code>TextField.password</code>	Indica si un campo de texto debe ocultar los caracteres introducidos.
<code>TextField._quality</code>	Indica la calidad de representación de un archivo SWF.
<code>TextField.restrict</code>	Conjunto de caracteres que puede introducir un usuario en un campo de texto.
<code>TextField._rotation</code>	Grado de rotación de una instancia de campo de texto.
<code>TextField.scroll</code>	Indica la posición de desplazamiento actual de un campo de texto.
<code>TextField.selectable</code>	Indica si un campo de texto se puede seleccionar.

Propiedad	Descripción
<code>TextField._soundbuftime</code>	Tiempo durante el cual un sonido debe almacenarse previamente en una memoria intermedia antes de empezar a fluir.
<code>TextField.tabEnabled</code>	Indica si se incluye un clip de película en el orden de tabulación automático.
<code>TextField.tabIndex</code>	Indica el orden de tabulación de un objeto.
<code>TextField._target</code>	Ruta de destino de la instancia del campo de texto especificada. Es de sólo lectura.
<code>TextField.text</code>	Texto actual del campo de texto.
<code>TextField.textColor</code>	Color del texto actual en el campo de texto.
<code>TextField.textHeight</code>	Altura del recuadro de delimitación del campo de texto.
<code>TextField.textWidth</code>	Anchura del recuadro de delimitación del campo de texto.
<code>TextField.type</code>	Indica si un campo de texto es un campo de introducción de texto o un campo de texto dinámico.
<code>TextField._url</code>	URL del archivo SWF que ha creado la instancia de campo de texto. Es de sólo lectura.
<code>TextField.variable</code>	Nombre de variable asociado con el campo de texto.
<code>TextField._visible</code>	Valor booleano que determina si una instancia de campo de texto está oculta o visible.
<code>TextField._width</code>	Anchura de una instancia de campo de texto en píxeles. Sólo afecta al recuadro de delimitación del campo de texto, no al grosor del borde ni al tamaño de fuente del texto.
<code>TextField.wordWrap</code>	Indica si el texto de un campo de texto se ajusta.
<code>TextField._x</code>	Coordenada x de una instancia de campo de texto.
<code>TextField._xmouse</code>	Coordenada x del puntero en relación con una instancia de campo de texto. Sólo lectura.
<code>TextField._xscale</code>	Valor que especifica el porcentaje de escala horizontal que se aplicará a una instancia de campo de texto.
<code>TextField._y</code>	Coordenada y de una instancia de campo de texto.
<code>TextField._ymouse</code>	Coordenada y del puntero en relación con una instancia de campo de texto. Sólo lectura.
<code>TextField._yscale</code>	Valor que especifica el porcentaje de escala vertical que se aplicará a una instancia de campo de texto.

## Resumen de controladores de eventos de la clase TextField

Controlador de eventos	Descripción
<code>TextField.onChangeed</code>	Se invoca cuando se cambia el campo de texto.
<code>TextField.onKillFocus</code>	Se invoca cuando el campo de texto deja de estar seleccionado.
<code>TextField.onScroller</code>	Se invoca cuando una de las propiedades de desplazamiento del campo de texto cambia.
<code>TextField.onSetFocus</code>	Se invoca cuando el campo de texto pasa a estar seleccionado.

## Resumen de detectores para la clase TextField

Método	Descripción
<code>TextField.onChangeed</code>	Se notifica cuando cambia el campo de texto.
<code>TextField.onScroller</code>	Se notifica cuando cambia la propiedad <code>scroll</code> o <code>maxscroll</code> de un campo de texto.

## TextField.addListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.addListener(listener)
```

### Parámetros

*listener* Objeto con un controlador de eventos `onChangeed` o `onScroller`.

### Valor devuelto

Ninguno.

### Descripción

Método; registra un objeto para recibir una notificación cuando se invocan los controladores de eventos `onChangeed` y `onScroller`. Cuando un campo de texto cambia o se desplaza, se invocan los controladores de eventos `TextField.onChangeed` y `TextField.onScroller`, seguidos por los controladores de eventos `onChangeed` y `onScroller` de cualquier objeto registrado como detector. Pueden registrarse varios objetos como detectores.

Para eliminar un objeto detector de un campo de texto, llame a `TextField.removeListener()`.

El origen del evento pasa una referencia a la instancia del campo de texto como parámetro para los controladores `onScroller` y `onChangeed`. Puede capturar estos datos colocando un parámetro en el método del controlador de eventos. Por ejemplo, en el código siguiente se utiliza `txt` como parámetro que se pasa al controlador de eventos `onScroller`. A continuación, se utiliza el parámetro en una sentencia `trace` para enviar el nombre de la instancia del campo de texto al panel Salida.

```
myTextField.onScroller = function (txt) {
    trace (txt._name + " ha cambiado");
};
```

### Ejemplo

En el ejemplo siguiente se define un controlador `onChange` para el campo de introducción de texto `myText`. A continuación, se define un nuevo objeto detector, `myListener`, y se define el controlador `onChanged` para dicho objeto. Este controlador se invocará cuando el campo de texto `myText` cambie. La línea final del código llama a `TextField.addListener` para registrar el objeto detector `myListener` con el campo de texto `myText` de modo que se le notifique cuando `myText` cambie.

```
myText.onChanged = function (txt) {
    trace(txt._name + " ha cambiado");
};
myListener = new Object();
myListener.onChanged = function (txt) {
    trace(txt._name + " ha cambiado y ha notificado myListener");
};

myText.addListener(myListener);
```

### Véase también

[TextField.onChanged](#), [TextField.onScroller](#), [TextField.removeListener\(\)](#)

## TextField.\_alpha

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt*.\_alpha

### Descripción

Propiedad; establece o recupera el valor de transparencia alfa del campo de texto especificado por *my\_text*. Los valores válidos van de 0 (completamente transparente) a 100 (completamente opaco). El valor predeterminado es 100.

### Ejemplo

El código siguiente establece la propiedad `_alpha` de un campo de texto llamado `text1_txt` en un 30% al hacer clic en el botón:

```
on(release) {
    text1_txt._alpha = 30;
}
```

### Véase también

[Button.\\_alpha](#), [MovieClip.\\_alpha](#)

# TextField.autoSize

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_txt*.autoSize

## Descripción

Propiedad; controla la alineación y la definición de tamaño automáticas de los campos de texto. Los valores aceptables para autoSize son "none" (el valor predeterminado), "left", "right" y "center". Cuando se establece la propiedad autoSize, true es un sinónimo de "left" y false es un sinónimo de "none".

Los valores de autoSize, multiline y wordWrap determinan si un campo de texto se expande o se contrae hacia el lado izquierdo, hacia el lado derecho o hacia el lado inferior. Puede utilizar el código siguiente y especificar distintos valores para autoSize, multiline y wordWrap para ver cómo cambia el tamaño del campo cuando cambian estos valores.

```
createTextField("my_txt", 1, 0, 0, 200, 20);
with (my_txt) {
    border = true;
    borderColor = 0x000000;
    multiline = false;
    wordWrap = false;
    autoSize = "none";
    text = "Éste es un ejemplo de la cantidad de texto que no cabe en el campo ";
}
```

## Ejemplo

A continuación se establece la propiedad autosize del campo de texto my\_text en "center".

```
my_txt.autosize = "center";
```

# TextField.background

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_txt*.background

## Descripción

Propiedad; si su valor es true, el campo de texto tiene un relleno de fondo. Si su valor es false, el campo de texto no tiene relleno de fondo.

# TextField.backgroundColor

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_txt*.backgroundColor

### Descripción

Propiedad; color del fondo del campo de texto. El valor predeterminado es 0xFFFFFF (blanco). Esta propiedad puede recuperarse o establecerse, incluso si no hay ningún color de fondo y el color sólo es visible si el campo de texto tiene un borde.

### Véase también

[TextField.background](#)

## TextField.border

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.border*

### Descripción

Propiedad; si su valor es `true`, el campo de texto tiene un borde. Si su valor es `false`, el campo de texto no tiene borde.

## TextField.borderColor

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.borderColor*

### Descripción

Propiedad; color del borde del campo de texto; el valor predeterminado es 0x000000 (negro). Esta propiedad se puede recuperar o establecer, incluso si no hay ningún borde.

### Véase también

[TextField.border](#)

## TextField.bottomScroll

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.bottomScroll*

### Descripción

Propiedad (sólo lectura); número entero (índice con base 1) que indica la última línea visible en *my\_txt*. Imagine que el campo de texto es como una “ventana” en un bloque de texto. La propiedad [TextField.scroll](#) es el índice con base 1 de la primera línea visible de la ventana.



Todo el texto entre las líneas `TextField.scroll` y `TextField.bottomScroll` se ve en el campo de texto.

## TextField.condenseWhite

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.condenseWhite
```

### Descripción

Propiedad; valor booleano que especifica si el espacio en blanco adicional (espacios, saltos de línea, etc.) en un campo de texto HTML debe eliminarse cuando el campo se muestra en un navegador. El valor predeterminado es `false`.

Si le asigna el valor `true`, debe utilizar comandos HTML estándar como `<BR>` y `<P>` para colocar saltos de línea en el campo de texto.

Si `my_txt.html` es `false`, esta propiedad se pasa por alto.

### Véase también

[TextField.html](#)

## TextField.embedFonts

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.embedFonts
```

### Descripción

Propiedad; valor booleano que, cuando es `true`, genera el campo de texto utilizando contornos de fuentes incorporadas. Si es `false`, genera el campo de texto mediante fuentes de dispositivo.

## TextField.getDepth()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.getDepth()
```

### Parámetros

Ninguno.

### Valor devuelto

Un número entero.

### Descripción

Método; devuelve la profundidad de un campo de texto.

## TextField.getFontList()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
TextField.getFontList()
```

### Parámetros

Ninguno.

### Valor devuelto

Una matriz.

### Descripción

Método; método estático de la [Clase TextField](#) global. Cuando llama a este método no indica un campo de texto específico (como `my_txt`). Este método devuelve los nombres de las fuentes del sistema host del reproductor en forma de matriz. No devuelve los nombres de todas las fuentes de los archivos SWF cargados actualmente. Los nombres son de tipo `string`.

### Ejemplo

En el código siguiente se muestra una lista de fuentes devuelta por `getFontList()`.

```
font_array = TextField.getFontList();
for(i in font_array){
    trace(font_array[i]);
}
```

## TextField.getNewTextFormat()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.getNewTextFormat()
```

### Parámetros

Ninguno.

### Valor devuelto

Un objeto `TextFormat`.

## Descripción

Método; devuelve un objeto `TextFormat` que contiene una copia del objeto de formato de texto del campo de texto. El objeto de formato de texto es el formato que recibe el texto recién insertado, por ejemplo, con el método `replaceSel()` o por un usuario. Cuando se invoca `getNewTextFormat()`, se definen todas las propiedades del objeto `TextFormat` devuelto. Ninguna propiedad tiene el valor `null`.

## TextField.getTextFormat()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.getTextFormat()  
my_txt.getTextFormat(index)  
my_txt.getTextFormat(beginIndex, endIndex)
```

### Parámetros

*index* Número entero que especifica un carácter de una cadena.  
*beginIndex, endIndex* Números enteros que especifican las ubicaciones inicial y final de un espacio de texto dentro de *my\_txt*.

### Valor devuelto

Un objeto.

### Descripción

Método; sintaxis 1: devuelve un objeto `TextFormat` que contiene información sobre el formato de la totalidad del texto de un campo de texto. En el objeto `TextFormat` resultante, sólo se establecen las propiedades comunes a todo el texto del campo de texto. El valor de las propiedades *mezcladas*, aquellas que tienen distintos valores en diferentes puntos del texto, se establece en `null`.

Sintaxis 2: devuelve un objeto `TextFormat` que contiene una copia del formato de texto del campo de texto del parámetro *index*.

Sintaxis 3: devuelve un objeto `TextFormat` que contiene información de formato del espacio del texto de *beginIndex* a *endIndex*.

### Véase también

[TextField.getNewTextFormat\(\)](#), [TextField.setNewTextFormat\(\)](#),  
[TextField.setTextFormat\(\)](#)

## TextField.\_height

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt._height
```

### Descripción

Propiedad; altura del campo de texto, expresada en píxeles.

### Ejemplo

El ejemplo de código siguiente establece la altura y la anchura de un campo de texto.

```
my_txt._width = 200;  
my_txt._height = 200;
```

## TextField.\_highquality

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.\_highquality*

### Descripción

Propiedad (global); especifica el nivel de suavizado aplicado al archivo SWF actual. Especifique 2 (calidad óptima) para aplicar alta calidad con el suavizado de mapa de bits siempre activado. Especifique 1 (alta calidad) para aplicar suavizado; esto suavizará los mapas de bits si el archivo SWF no contiene animación. Especifique 0 (baja calidad) para evitar el suavizado.

### Véase también

[\\_quality](#)

## TextField.hscroll

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.hscroll*

### Valor devuelto

Un número entero.

### Descripción

Propiedad; indica la posición actual de desplazamiento horizontal. Si la propiedad `hscroll` es 0, el texto no se desplaza horizontalmente.

Para más información sobre texto desplazable, consulte [“Creación de texto desplazable” en la página 158](#).

### Ejemplo

En el ejemplo siguiente se desplaza el texto horizontalmente.

```
on(release) {  
    my_txt.hscroll += 1;  
}
```

Véase también

[TextField.maxhscroll](#), [TextField.scroll](#)

## TextField.html

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.html
```

### Descripción

Propiedad; etiqueta que indica si el campo de texto contiene una representación HTML. Si el valor de la propiedad `html` es `true`, el campo de texto es un campo de texto HTML. Si el valor de `html` es `false`, el campo de texto no es un campo de texto HTML.

Véase también

[TextField.htmlText](#)

## TextField.htmlText

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.htmlText
```

### Descripción

Propiedad; si el campo de texto es HTML, esta propiedad contiene la representación HTML del contenido del campo de texto. Si el campo de texto no es HTML, se comporta exactamente igual que la propiedad `text`. Puede indicar que un campo de texto es un campo de texto HTML en el inspector de propiedades, o bien estableciendo la propiedad `html` del campo de texto en `true`.

### Ejemplo

En el ejemplo siguiente, el texto del campo de texto `text2` aparece en negrita.

```
text2.html = true;  
text2.htmlText = "<b> texto en negrita </b>";
```

Véase también

[TextField.html](#)

## TextField.length

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.length
```

**Valor devuelto**

Un número.

**Descripción**

Propiedad (sólo lectura); indica el número de caracteres de un campo de texto. Esta propiedad devuelve el mismo valor que `text.length`, pero es más rápida. El carácter de tabulación (“\t”) se cuenta como un carácter.

## TextField.maxChars

**Disponibilidad**

Flash Player 6.

**Sintaxis**

*my\_txt.maxChars*

**Descripción**

Propiedad; indica el número máximo de caracteres que puede contener el campo de texto. Un script puede insertar más texto del que permite `maxChars`; la propiedad `maxChars` sólo indica cuánto texto puede introducir un usuario. Si el valor de esta propiedad es `null`, no hay límite en cuanto a la cantidad de texto que puede introducir un usuario.

## TextField.maxhscroll

**Disponibilidad**

Flash Player 6.

**Sintaxis**

*my\_txt.maxhscroll*

**Descripción**

Propiedad (sólo lectura); indica el valor máximo de `TextField.hscroll`.

## TextField.maxscroll

**Disponibilidad**

Flash Player 6.

**Sintaxis**

*TextField.maxscroll*

**Descripción**

Propiedad (sólo lectura); indica el valor máximo de `TextField.scroll`.

Para más información sobre texto desplazable, consulte [“Creación de texto desplazable” en la página 158](#).

## TextField.menu

### Disponibilidad

Flash Player 7.

### Utilización

```
my_txt.menu = contextMenu
```

### Parámetros

*contextMenu*    Objeto ContextMenu.

### Descripción

Propiedad; asocia el objeto ContextMenu *contextMenu* con el campo de texto *my\_txt*. La clase ContextMenu permite modificar el menú contextual que aparece al hacer clic con el botón derecho del ratón (Windows) o al mantener presionada la tecla Control y hacer clic (Macintosh) en Flash Player.

Esta propiedad sólo funciona con campos de texto que se puedan seleccionar (editar); no tiene ningún efecto sobre los campos de texto que no pueden seleccionarse.

### Ejemplo

En el ejemplo siguiente se asigna el objeto ContextMenu *menu\_cm* al campo de texto *news\_txt*. El objeto ContextMenu contiene un elemento de menú personalizado etiquetado “Imprimir” con un controlador callback asociado denominado *doPrint()*, que lleva a cabo operaciones de impresión (no se muestran).

```
var menu_cm = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Imprimir...", doPrint));
function doPrint(menu, obj) {
    // Aquí aparece el código "Imprimir"
}
news_txt.menu = menu_cm;
```

### Véase también

[Button.menu](#), [Clase ContextMenu](#), [Clase ContextMenuItem](#), [MovieClip.menu](#)

## TextField.mouseWheelEnabled

### Disponibilidad

Flash Player 7.

### Sintaxis

```
my_txt.mouseWheelEnabled
```

### Descripción

Propiedad; valor booleano que indica si Flash Player debe desplazar automáticamente el contenido de los campos de texto multilínea cuando el puntero del ratón se coloca sobre un campo de texto y el usuario hace girar la rueda del ratón. De forma predeterminada, este valor es *true*. Esta propiedad es útil si desea evitar que la rueda del ratón se desplace en los campos de texto o si desea implementar su propio desplazamiento de los campos de texto.

Véase también

[Mouse.onMouseWheel](#)

## TextField.multiline

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.multiline
```

### Descripción

Propiedad; indica si el campo de texto contiene varias líneas. Si el valor es `true`, el campo de texto tiene varias líneas; si el valor es `false`, el campo de texto sólo tiene una línea.

## TextField.\_name

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt._name
```

### Descripción

Propiedad; nombre de instancia del campo de texto especificado por *my\_txt*.

## TextField.onChangeed

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.onChangeed = function(){  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

El nombre de la instancia del campo de texto.

### Descripción

Controlador de eventos; se invoca cuando cambia el contenido de un campo de texto. De forma predeterminada, no está definido; puede definirlo en un script.



Se pasa una referencia a la instancia de campo de texto como parámetro para el controlador `onChanged`. Puede capturar estos datos colocando un parámetro en el método del controlador de eventos. Por ejemplo, en el código siguiente se utiliza `txt` como parámetro que se pasa para el controlador de eventos `onChanged`. A continuación, se utiliza el parámetro en una sentencia `trace()` para enviar el nombre de instancia del campo de texto al panel Salida.

```
myTextField.onChanged = function (txt) {  
    trace (txt._name + " ha cambiado");  
};
```

## TextField.onKillFocus

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.onKillFocus = function(newFocus){  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*newFocus*    Objeto que está seleccionado.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca cuando un campo de texto deja de seleccionarse mediante el teclado. El método `onKillFocus` recibe un parámetro, *newFocus*, que es un objeto que representa el nuevo objeto seleccionado. Si no hay ningún objeto seleccionado, *newFocus* contiene el valor `null`.

## TextField.onScroller

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.onScroller = function(textFieldInstance){  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*textFieldInstance*    Referencia al objeto `TextField` cuya posición de desplazamiento ha cambiado.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando una de las propiedades de desplazamiento del campo de texto cambia.

Se pasa una referencia a la instancia de campo de texto como parámetro para el controlador `onScroller`. Puede capturar estos datos colocando un parámetro en el método del controlador de eventos. Por ejemplo, en el código siguiente se utiliza `txt` como parámetro que se pasa al controlador de eventos `onScroller`. A continuación, se utiliza el parámetro en una sentencia `trace()` para enviar el nombre de instancia del campo de texto al panel Salida.

```
myTextField.onScroller = function (txt) {  
    trace (txt._name + " se ha desplazado");  
};
```

## Véase también

[TextField.hscroll](#), [TextField.maxhscroll](#), [TextField.maxscroll](#), [TextField.scroll](#)

# TextField.onSetFocus

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_txt.onSetFocus = function(oldFocus) {  
    // las sentencias se escriben aquí  
}
```

## Parámetros

*oldFocus*    Objeto que dejará de estar seleccionado.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando un campo de texto se selecciona mediante el teclado. El parámetro *oldFocus* es el objeto que deja de estar seleccionado. Por ejemplo, si el usuario presiona la tecla Tabulador para cambiar la selección de entrada de un botón a un campo de texto, *oldFocus* contiene la instancia de campo de texto.

Si no hay ningún objeto seleccionado anteriormente, *oldFocus* contiene un valor `null`.

# TextField.\_parent

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_txt._parent.property  
_parent.property
```

### Descripción

Propiedad; referencia al clip de película u objeto que contiene el campo de texto o el objeto actual. El objeto actual es el que contiene el código de ActionScript que hace referencia a `_parent`.

Utilice `_parent` para especificar una ruta relativa a los clips de película u objetos que están por encima del campo de texto actual. Puede utilizar `_parent` para subir varios niveles en la lista de visualización, como se muestra a continuación:

```
_parent._parent._alpha = 20;
```

### Véase también

[Button.\\_parent](#), [MovieClip.\\_parent](#), [\\_root](#), [targetPath](#)

## TextField.password

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.password
```

### Descripción

Propiedad; si el valor de `password` es `true`, el campo de texto es un campo de texto de contraseña y oculta los caracteres de entrada. Si su valor es `false`, el campo de texto no es de contraseña.

## TextField.\_quality

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt._quality
```

### Descripción

Propiedad (global); establece o recupera la calidad de representación utilizada para un archivo SWF. Las fuentes de dispositivo siempre son dentadas, de modo que no se ven afectadas por la propiedad `_quality`.

**Nota:** aunque puede especificar esta propiedad para un objeto `TextField`, en realidad es una propiedad global, y su valor puede especificarse simplemente como `_quality`. Para más información, consulte [\\_quality](#).

## TextField.removeListener()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.removeListener(listener)
```

### Parámetros

*listener* Objeto que ya no recibirá notificaciones de `TextField.onChangeed` ni `TextField.onScroller`.

### Valor devuelto

Si *listener* se ha eliminado correctamente, el método devuelve un valor `true`. Si *listener* no se ha eliminado correctamente (por ejemplo, si *listener* no se encontraba en la lista de detectores del objeto `TextField`), el método devuelve el valor `false`.

### Descripción

Método; elimina un objeto detector previamente registrado en una instancia de campo de texto con `TextField.addListener()`.

## TextField.removeTextField()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.removeTextField()
```

### Descripción

Método; elimina el campo de texto especificado por *my\_text*. Esta operación sólo puede realizarse en un campo de texto creado con `MovieClip.createTextField()`. Cuando se llama a este método, el campo de texto se elimina. Este método es parecido a `MovieClip.removeMovieClip()`.

## TextField.replaceSel()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.replaceSel(text)
```

### Parámetros

*text* Una cadena.

### Valor devuelto

Ninguno.

### Descripción

Método; reemplaza el elemento seleccionado actualmente por el contenido del parámetro *text*. El texto se inserta en la posición de la selección actual, con los formatos de carácter y párrafo predeterminados. El texto no se trata como si fuera HTML, ni siquiera cuando el campo de texto es HTML.

Puede utilizar el método `replaceSel()` para insertar y eliminar texto sin que el formato de carácter y de párrafo del resto del texto se vea afectado.

Debe utilizar `Selection.setFocus()` para seleccionar el campo antes de emitir este comando.

#### Véase también

`Selection.setFocus()`

## TextField.replaceText()

#### Disponibilidad

Flash Player 7.

#### Sintaxis

```
my_txt.replaceText(beginIndex, endIndex, text)
```

#### Descripción

Método; sustituye un grupo de caracteres, especificado por los parámetros *beginIndex* y *endIndex*, en el campo de texto especificado por el contenido del parámetro *text*.

## TextField.restrict

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_txt.restrict
```

#### Descripción

Propiedad; indica el conjunto de caracteres que los usuarios pueden introducir en el campo de texto. Si el valor de la propiedad `restrict` es `null`, se puede introducir cualquier carácter. Si el valor de la propiedad `restrict` es una cadena vacía, no se puede introducir ningún carácter. Si el valor de la propiedad `restrict` es una cadena de caracteres, sólo puede introducir los caracteres de la cadena en el campo de texto. La cadena se explora de izquierda a derecha. Puede especificarse un intervalo utilizando un guión (-). De este modo sólo se limita la interacción del usuario; un script puede introducir cualquier texto en el campo de texto. Esta propiedad no se sincroniza con las casillas de verificación de Incorporar contornos de fuentes del inspector de propiedades.

Si la cadena empieza por `^`, inicialmente se aceptan todos los caracteres y los caracteres posteriores de la cadena se excluyen del conjunto de caracteres aceptados. Si la cadena no empieza por `^`, inicialmente no se acepta ningún carácter y los caracteres posteriores de la cadena se incluyen en el conjunto de caracteres aceptados.

#### Ejemplo

En el ejemplo siguiente sólo se pueden introducir caracteres en mayúsculas, espacios y números en los campos de texto:

```
my_txt.restrict = "A-Z 0-9";
```

En el ejemplo siguiente se incluyen todos los caracteres, pero se excluyen las letras minúsculas:

```
my_txt.restrict = "^a-z";
```

Para introducir los caracteres ^ o -, utilice una barra inversa. Las secuencias aceptadas con barra inversa son \-, \^ y \\. La barra inversa debe ser un carácter real de la cadena, de modo que cuando se especifique en ActionScript, deben utilizarse dos. Por ejemplo, el código siguiente incluye sólo el guión (-) y el carácter de intercalación (^):

```
my_txt.restrict = "\\-\\^";
```

El carácter ^ puede utilizarse en cualquier posición de la cadena para alternar la inclusión y la exclusión de caracteres. El código siguiente incluye sólo letras mayúsculas, pero excluye la letra mayúscula Q:

```
my_txt.restrict = "A-Z^Q";
```

Puede utilizar la secuencia de escape \u para crear cadenas restrict. El código siguiente incluye sólo los caracteres comprendidos entre ASCII 32 (espacio) y ASCII 126 (tilde).

```
my_txt.restrict = "\u0020-\u007E";
```

## TextField.\_rotation

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt._rotation
```

### Descripción

Propiedad; la rotación del campo de texto, en grados, de su orientación original. Los valores entre 0 y 180 corresponden a la rotación en el sentido de las agujas del reloj, mientras que los valores entre 0 y -180 corresponden a la rotación en el sentido contrario a las agujas del reloj. Los valores que quedan fuera de este rango se suman o se restan de 360 para obtener un valor dentro del rango. Por ejemplo, la sentencia `my_txt._rotation = 450` es la misma que `my_txt._rotation = 90`.

### Véase también

[Button.\\_rotation](#), [MovieClip.\\_rotation](#)

## TextField.scroll

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.scroll
```

### Descripción

Propiedad; define la posición vertical del texto de un campo de texto. La propiedad `scroll` es útil para dirigir a los usuarios a un párrafo específico en un pasaje largo, o para crear campos de texto con desplazamiento. Esta propiedad puede recuperarse y modificarse.

Para más información sobre texto desplazable, consulte [“Creación de texto desplazable” en la página 158](#).

## Ejemplo

El código siguiente está asociado a un botón Arriba que desplaza el contenido del campo de texto `my_txt`.

```
on(release) {  
    my_txt.scroll = myText.scroll + 1;  
}
```

## Véase también

[TextField.hscroll](#), [TextField.maxscroll](#)

# TextField.selectable

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_txt.selectable
```

## Descripción

Propiedad; valor booleano que indica si el campo de texto se puede seleccionar (editar). El valor `true` indica que el texto se puede seleccionar.

# TextField.setNewTextFormat()

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_txt.setNewTextFormat(textFormat)
```

## Parámetros

*textFormat*    Objeto `TextFormat`.

## Valor devuelto

Ninguno.

## Descripción

Método; establece un objeto `TextFormat` para el texto recién insertado, por ejemplo, texto insertado con el método `replaceSel()` o texto introducido por un usuario en un campo de texto. Cada campo de texto tiene un nuevo formato de texto. Cuando se inserta el texto, se asigna el nuevo formato de texto al nuevo texto.

El formato de texto se establece en un nuevo objeto `TextFormat`. Contiene información de formato de carácter y de párrafo. La información de formato de carácter describe el aspecto de cada carácter, como por ejemplo, el nombre de la fuente, el tamaño en puntos, el color y la URL asociada. La información de formato de párrafo describe el aspecto de un párrafo, como por ejemplo, el margen izquierdo, el margen derecho, la sangría de la primera línea y la alineación a la izquierda, a la derecha o centrada.

Véase también

[TextField.getNewTextFormat\(\)](#), [TextField.getTextFormat\(\)](#),  
[TextField.setTextFormat\(\)](#)

## TextField.setTextFormat()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.setTextFormat (textFormat)
my_txt.setTextFormat (index, textFormat)
my_txt.setTextFormat (beginIndex, endIndex, textFormat)
```

### Parámetros

*textFormat* Objeto TextFormat que contiene información sobre el formato de los caracteres y los párrafos.

*index* Número entero que especifica un carácter dentro de my\_txt.

*beginIndex* Un número entero.

*endIndex* Número entero que especifica el primer carácter después del espacio de texto deseado.

### Valor devuelto

Ninguno.

### Descripción

Método; establece un objeto TextFormat para una parte especificada del texto de un campo de texto. Puede asignar un formato de texto a cada carácter de un campo de texto. El formato de texto del primer carácter de un párrafo se examina para asignar formato de párrafo a todo el párrafo. El método `setTextFormat()` cambia el formato de texto aplicado a caracteres individuales, a grupos de caracteres o a todo el cuerpo del texto de un campo de texto.

El formato de texto se establece en un nuevo objeto TextFormat. Contiene información de formato de carácter y de párrafo. La información de formato de carácter describe el aspecto de cada carácter; por ejemplo, el nombre de fuente, el tamaño en puntos, el color y la URL asociada. La información de formato de párrafo describe el aspecto de un párrafo, como por ejemplo, el margen izquierdo, el margen derecho, la sangría de la primera línea y la alineación a la izquierda, a la derecha o centrada.

Sintaxis 1: aplica las propiedades del parámetro *textFormat* a todo el texto del campo de texto.

Sintaxis 2: aplica las propiedades del parámetro *textFormat* al carácter de la posición *index*.

Sintaxis 3: aplica las propiedades del parámetro *textFormat* al espacio de texto desde el parámetro *beginIndex* hasta el parámetro *endIndex*.

Debe tener en cuenta que el texto insertado manualmente por el usuario, o sustituido mediante [TextField.replaceSel\(\)](#), no adquiere el formato especificado en una llamada a `setTextFormat()`. Para establecer el formato predeterminado de un objeto TextField, utilice [TextField.setNewTextFormat\(\)](#).



## Ejemplo

Este ejemplo crea un nuevo objeto `TextFormat` llamado `myTextFormat` y establece su propiedad `bold` en `true`. A continuación llama a `setTextFormat()` y aplica el nuevo formato de texto al campo de texto `my_txt`.

```
myTextFormat = new TextFormat();
myTextFormat.bold = true;
my_txt.setTextFormat(myTextFormat);
```

## Véase también

[TextField.setNewTextFormat\(\), Clase TextFormat](#)

# TextField.\_soundbuftime

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_txt.\_soundbuftime*

## Descripción

Propiedad (global); entero que especifica el número de segundos que un sonido se almacena previamente en una memoria intermedia antes de que empiece a reproducirse.

# Clase TextField.StyleSheet

## Disponibilidad

Flash Player 7.

## Descripción

La clase `TextField.StyleSheet` permite crear un objeto de hoja de estilos que contiene las reglas de formato de texto que controlan, por ejemplo, el tamaño y el color de fuente, así como otros estilos de formato. A continuación, puede aplicar estilos definidos por una hoja de estilos a un objeto `TextField` que contenga texto en formato HTML o XML. El texto del objeto `TextField` adquiere automáticamente el formato especificado por los estilos de etiqueta definidos en el objeto de hoja de estilos. Puede utilizar estilos de texto para definir nuevas etiquetas de formato, volver a definir etiquetas HTML incorporadas o crear clases de estilos que puedan aplicarse a determinadas etiquetas HTML.

Para aplicar estilos a un objeto `TextField`, asigne el objeto de hoja de estilos a la propiedad `styleSheet` del objeto `TextField`.

Para más información, consulte [“Aplicación de formato al texto con hojas de estilos en cascada” en la página 143](#).

## Resumen de métodos para la clase `TextField.StyleSheet`

Método	Descripción
<code>TextField.StyleSheet.getStyle()</code>	Devuelve una copia del objeto de hoja de estilos asociado con el nombre de estilo especificado.
<code>TextField.StyleSheet.getStyleNames()</code>	Devuelve una matriz que contiene los nombres de todos los estilos registrados en el objeto de hoja de estilos.
<code>TextField.StyleSheet.load()</code>	Empieza a cargar un archivo CSS en un objeto de hoja de estilos.
<code>TextField.StyleSheet.parseCSS()</code>	Analiza una cadena de texto CSS y crea el estilo especificado.
<code>TextField.StyleSheet.setStyle()</code>	Añade un nuevo estilo al objeto de hoja de estilos.

## Resumen de controladores de eventos para la clase `TextField.StyleSheet`

Método	Descripción
<code>TextField.StyleSheet.onLoad</code>	Controlador callback que se invoca cuando finaliza una operación <code>TextField.StyleSheet.load()</code> .

## Constructor para la clase `TextField.StyleSheet`

### Disponibilidad

Flash Player 7.

### Sintaxis

```
new TextField.StyleSheet()
```

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un objeto `TextField.StyleSheet`.

## `TextField.StyleSheet.getStyle()`

### Disponibilidad

Flash Player 7.

### Sintaxis

```
styleSheet.getStyle(styleName)
```

### Parámetros

*styleName* Cadena que especifica el nombre del estilo que debe recuperarse.

### Valor devuelto

Un objeto.

## Descripción

Método; devuelve una copia del objeto de estilo asociado con el estilo denominado *styleName*. Si *styleName* no tiene ningún objeto de estilo asociado, se devuelve *null*.

## Ejemplo

Tomemos por ejemplo un objeto de hoja de estilos denominado `textStyles` que carga un archivo de hoja de estilos externo denominado `styles.css` que contiene un único estilo denominado `heading`, que a su vez define las propiedades `font-family`, `font-size` y `font-weight`, como se muestra a continuación.

```
// En styles.css
heading {
    font-family: Arial;
    font-size: 24px;
    font-weight: bold;
}
```

El código siguiente carga los estilos del archivo CSS y muestra el nombre de cada propiedad con el valor correspondiente en el panel Salida.

```
var styleSheet = new TextField.styleSheet();
styleSheet.load("styles.css");
var sectionStyle = styleSheet.getStyle("heading");
for(property in sectionStyle) {
    var propName = property;
    var propValue = sectionStyle[property];
    trace(propName + " : " + propValue);
}
```

El panel Salida mostraría lo siguiente:

```
fontfamily : Arial
fontsize : 24px
fontweight : bold
```

## Véase también

[TextField.StyleSheet.setStyle\(\)](#)

# TextField.StyleSheet.getStyleNames()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
styleSheet.getStyleNames()
```

## Parámetros

Ninguno.

## Valor devuelto

Una matriz.

## Descripción

Método; devuelve una matriz que contiene los nombres (como cadenas) de todos los estilos registrados en esta hoja de estilos.

## Ejemplo

Este ejemplo crea un objeto de hoja de estilos denominado `styleSheet` que contiene dos estilos: `heading` y `bodyText`. A continuación invoca el método `getStyleNames()` del objeto de la hoja de estilos, asigna los resultados a la matriz `names_array` y muestra el contenido de la matriz en el panel Salida.

```
var styleSheet= new TextField.StyleSheet();
styleSheet.setStyle("heading", {
    fontsize: '24px'
});
styleSheet.setStyle("bodyText", {
    fontsize: '12px'
});
var names_array = styleSheet.getStyleNames();
trace(names_array.join("\n"));
```

El panel Salida muestra lo siguiente:

```
bodyText
heading
```

## Véase también

[TextField.StyleSheet.getStyle\(\)](#)

# TextField.StyleSheet.load()

## Disponibilidad

Flash Player 7.

## Sintaxis

```
styleSheet.load(url)
```

## Parámetros

*url* URL del archivo CSS que se va a cargar. La URL debe pertenecer al mismo dominio que la URL donde reside actualmente el archivo SWF.

## Valor devuelto

Ninguno.

## Descripción

Método; empieza a cargar el archivo CSS en la *styleSheet*. La operación de carga es asíncrona; utilice el controlador callback [TextField.StyleSheet.onLoad](#) para determinar cuándo finaliza el proceso de carga del archivo.

El archivo CSS debe residir exactamente en el mismo dominio que el archivo SWF que lo está cargando. Para más información sobre restricciones al cargar datos de distintos dominios, consulte [“Funciones de seguridad de Flash Player” en la página 196](#).

## Ejemplo

En el ejemplo siguiente se carga el archivo CSS denominado `styles.css` (no se muestra) en el objeto de hoja de estilos `styleObj`. Una vez que el archivo se ha cargado correctamente, se aplica el objeto de hoja de estilos a un objeto `TextField` denominado `news_txt`.

```
var styleObj = new TextField.StyleSheet();
styleObj.load("styles.css");
styleObj.onLoad = function (success) {
    if (success){
        news_txt.styleSheet = styleObj;
    }
}
```

## Véase también

[TextField.StyleSheet.onLoad](#)

# TextField.StyleSheet.onLoad

## Disponibilidad

Flash Player 7.

## Sintaxis

```
styleSheet.onLoad = function (success) {}
```

## Parámetros

*success* Valor booleano que indica si el archivo CSS se ha cargado correctamente.

## Valor devuelto

Ninguno.

## Descripción

Controlador callback; se invoca cuando finaliza una operación [TextField.StyleSheet.load\(\)](#). Si la hoja de estilos se ha cargado correctamente, el valor del parámetro *success* es `true`. Si el documento no se ha recibido o se ha producido un error al recibir la respuesta del servidor, el valor del parámetro *success* es `false`.

## Ejemplo

En el ejemplo siguiente se carga el archivo CSS denominado `styles.css` (no se muestra) en el objeto de hoja de estilos `styleObj`. Una vez que el archivo se ha cargado correctamente, se aplica el objeto de hoja de estilos a un objeto `TextField` denominado `news_txt`.

```
var styleObj = new TextField.StyleSheet();
styleObj.load("styles.css");
styleObj.onLoad = function (success) {
    if (success){
        news_txt.styleSheet = styleObj;
    }
}
```

## Véase también

[TextField.StyleSheet.load\(\)](#)

## TextField.StyleSheet.parseCSS()

Disponibilidad

Flash Player 7.

### Sintaxis

```
styleSheet.parseCSS(cssText)
```

### Parámetros

*cssText* Texto CSS que debe analizarse (una cadena).

### Valor devuelto

Valor booleano que indica si el texto se ha analizado correctamente (`true`) o no (`false`).

### Descripción

Método; analiza el texto CSS de *cssText* y carga con él la hoja de estilos. Si *styleSheet* ya contiene un estilo de *cssText*, las propiedades de *styleSheet* se mantendrán, y en *styleSheet* sólo se añadirán o cambiarán los estilos de *cssText*.

Para ampliar la capacidad de análisis CSS nativa, puede cambiar este método mediante la creación de una subclase de la clase `TextField.StyleSheet`. Para más información, consulte [“Creación de subclases” en la página 169](#).

## TextField.StyleSheet.setStyle()

Disponibilidad

Flash Player 7.

### Sintaxis

```
styleSheet.setStyle(name, style)
```

### Parámetros

*name* Cadena que especifica el nombre del estilo que debe añadirse a la hoja de estilos.

*style* Objeto que describe el estilo, o el valor `null`.

### Valor devuelto

Ninguno.

### Descripción

Método; añade un nuevo estilo con el nombre especificado en el objeto de hoja de estilos. Si la hoja de estilos no contiene el estilo nombrado, se añade. En el caso de que sí exista, se reemplaza. Si el parámetro *style* es `null`, el estilo nombrado se elimina.

Flash Player crea una copia del objeto de estilos que se ha pasado a este método.

## Ejemplo

En el código siguiente se añade un estilo denominado `emphasized` a la hoja de estilos `myStyleSheet`. El estilo incluye dos propiedades de estilo: `color` y `fontWeight`. El objeto de estilo se define mediante el operador `{}`.

```
myStyleSheet.setStyle("emphasized", {color:'#000000',fontWeight:'bold'});
```

También puede crear un objeto de estilos utilizando una instancia de la clase `Object` y, a continuación, pasar dicho objeto como parámetro `style`, como se muestra en el ejemplo siguiente.

```
var styleObj = new Object();
styleObj.color = '#000000';
styleObj.fontWeight = 'bold';
myStyleSheet.setStyle("emphasized", styleObj);
delete styleObj;
```

**Nota:** la última línea de código (`delete styleObj`) elimina el objeto de estilo original que se ha pasado a `setStyle()`. Si bien este paso no es necesario, sirve para reducir el uso de memoria, ya que Flash Player crea una copia del objeto de estilo que se pasa a `setStyle()`.

## Véase también

`{}` (inicializador de objeto)

# TextField.styleSheet

## Disponibilidad

Flash Player 7.

## Sintaxis

```
my_txt.styleSheet = TextField StyleSheet
```

## Descripción

Propiedad; asocia una hoja de estilos al campo de texto especificado por `my_txt`. Para más información sobre cómo crear hojas de estilos, consulte la entrada [Clase TextField.StyleSheet](#) y “Aplicación de formato al texto con hojas de estilos en cascada” en la página 143.

# TextField.tabEnabled

## Disponibilidad

Flash Player 6.

## Sintaxis

```
my_txt.tabEnabled
```

## Descripción

Propiedad; especifica si `my_txt` se incluye en el orden de tabulación automático. El valor predeterminado es `undefined`.

Si la propiedad `tabEnabled` es `undefined` o `true`, el objeto se incluye en el orden de tabulación automático. Si la propiedad `tabIndex` también se establece en un valor, el objeto se incluye en el orden de tabulación personalizado. Si `tabEnabled` es `false`, el objeto no se incluye en el orden de tabulación personalizado o automático, aunque se establezca la propiedad `tabIndex`.

### Véase también

[Button.tabEnabled](#), [MovieClip.tabEnabled](#)

## TextField.tabIndex

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt.tabIndex
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Propiedad; permite personalizar el orden de tabulación de los objetos de un archivo SWF. Puede establecer la propiedad `tabIndex` de un botón, un clip de película o una instancia de campo de texto; de manera predeterminada el valor es `undefined`.

Si alguno de los objetos que se muestran actualmente en el archivo SWF contiene una propiedad `tabIndex`, el orden de tabulación automático está desactivado, y el orden de tabulación se calcula a partir de las propiedades `tabIndex` de los objetos del archivo SWF. El orden de tabulación personalizado sólo incluye objetos que tienen propiedades `tabIndex`.

La propiedad `tabIndex` debe ser un entero positivo. Los objetos se ordenan de acuerdo con sus propiedades `tabIndex`, en orden ascendente. Un objeto cuyo valor de `tabIndex` sea 1 precede a un objeto cuyo valor de `tabIndex` sea 2. Si dos objetos tienen el mismo valor de `tabIndex`, el que precede al otro en el orden de tabulación es `undefined`.

El orden de tabulación personalizado definido por la propiedad `tabIndex` es *flat*. Esto significa que no se tienen en cuenta las relaciones jerárquicas de los objetos del archivo SWF. Todos los objetos del archivo SWF con propiedades `tabIndex` se colocan en el orden de tabulación, que viene determinado por el orden de los valores `tabIndex`. Si dos valores tienen el mismo valor de `tabIndex`, el precedente será `undefined`. No debe utilizarse el mismo valor de `tabIndex` para varios objetos.

### Véase también

[Button.tabIndex](#), [MovieClip.tabIndex](#)

## TextField.\_target

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_txt._target
```



### Descripción

Propiedad (sólo lectura); ruta de destino de la instancia de campo de texto especificada por *my\_txt*.

## TextField.text

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt*.text

### Descripción

Propiedad; indica el texto actual en el campo de texto. Las líneas se separan mediante el carácter de retorno de carro ('\r', ASCII 13). Esta propiedad contiene el texto normal sin formato del campo de texto, sin etiquetas HTML, aunque el campo de texto sea HTML.

### Véase también

[TextField.htmlText](#)

## TextField.textColor

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt*.textColor

### Descripción

Propiedad; indica el color del texto de un campo de texto.

## TextField.textHeight

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt*.textHeight

### Descripción

Propiedad; indica la altura del texto.

## TextField.textWidth

### Disponibilidad

Flash Player 6.

**Sintaxis**

```
my_txt.textWidth
```

**Descripción**

Propiedad; indica la anchura del texto.

## TextField.type

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_txt.type
```

**Descripción**

Propiedad; especifica el tipo de campo de texto. Hay dos valores: "dynamic", que especifica un campo de texto dinámico que el usuario no puede editar e "input", que especifica un campo de introducción de texto.

**Ejemplo**

```
my_txt.type = "dynamic";
```

## TextField.\_url

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_txt._url
```

**Descripción**

Propiedad (sólo lectura); recupera la URL del archivo SWF que ha creado el campo de texto.

## TextField.variable

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_txt.variable
```

**Descripción**

Propiedad; nombre de la variable con la que está asociado el campo de texto. El tipo de esta propiedad es String.

## TextField.\_visible

### Disponibilidad

Flash Player 6.

### Sintaxis

`my_txt._visible`

### Descripción

Propiedad; valor booleano que indica si el campo de texto `my_txt` es visible. Los campos de texto que no están visibles (con la propiedad `_visible` establecida en `false`) están desactivados.

### Véase también

[Button.\\_visible](#), [MovieClip.\\_visible](#)

## TextField.\_width

### Disponibilidad

Flash Player 6.

### Sintaxis

`my_txt._width`

### Descripción

Propiedad; anchura del campo de texto, expresada en píxeles.

### Ejemplo

En el ejemplo siguiente se establecen las propiedades de altura y anchura de un campo de texto:

```
my_txt._width=200;  
my_txt._height=200;
```

### Véase también

[MovieClip.\\_height](#)

## TextField.wordWrap

### Disponibilidad

Flash Player 6.

### Sintaxis

`my_txt.wordWrap`

### Descripción

Propiedad; valor booleano que indica si el campo de texto tiene un ajuste de texto. Si el valor de `wordWrap` es `true`, significa que el campo de texto tiene un ajuste de texto; si el valor es `false`, el campo de texto no tiene ningún ajuste de texto.

## TextField.\_x

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.\_x*

### Descripción

Propiedad; número entero que establece la coordenada *x* de un campo de texto en relación con las coordenadas locales del clip de película principal. Si un campo de texto está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el campo de texto se encuentra dentro de un clip de película que tiene transformaciones, el campo de texto está en el sistema de coordenadas local del clip de película que lo contiene. Por consiguiente, para un clip de película que se ha girado 90 grados en sentido contrario a las agujas del reloj, el campo de texto incluido hereda un sistema de coordenadas que está girado 90 grados en el mismo sentido. Las coordenadas del campo de texto hacen referencia a la posición del punto de registro.

### Véase también

[TextField.\\_xscale](#), [TextField.\\_y](#), [TextField.yscale](#)

## TextField.xmouse

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.xmouse*

### Descripción

Propiedad (sólo lectura); devuelve la coordenada *x* de la posición del ratón respecto al campo de texto.

### Véase también

[TextField.ymouse](#)

## TextField.xscale

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.xscale*

### Descripción

Propiedad; determina la escala horizontal del campo de texto que se aplica desde el punto de registro del campo de texto, expresada como porcentaje. El punto de registro predeterminado es (0,0).

### Véase también

[TextField.\\_x](#), [TextField.\\_y](#), [TextField.\\_yscale](#)

## TextField.\_y

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.\_y*

### Descripción

Propiedad; coordenada *y* de un campo de texto en relación con las coordenadas locales del clip de película principal. Si un campo de texto está en la línea de tiempo principal, su sistema de coordenadas hace referencia a la esquina superior izquierda del escenario como (0, 0). Si el campo de texto se encuentra dentro de un clip de película que tiene transformaciones, el campo de texto está en el sistema de coordenadas locales del clip de película que lo contiene. Por consiguiente, para un clip de película que se ha girado 90 grados en sentido contrario a las agujas del reloj, el campo de texto incluido hereda un sistema de coordenadas que está girado 90 grados en el mismo sentido. Las coordenadas del campo de texto hacen referencia a la posición del punto de registro.

### Véase también

[TextField.\\_x](#), [TextField.\\_xscale](#), [TextField.\\_yscale](#)

## TextField.\_ymouse

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.\_ymouse*

### Descripción

Propiedad (sólo lectura); indica la coordenada *y* de la posición del ratón en relación con el campo de texto.

### Véase también

[TextField.\\_xmouse](#)

## TextField.\_yscale

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_txt.\_yscale*

### Descripción

Propiedad; determina la escala vertical del campo de texto que se aplica desde el punto de registro del campo de texto, expresada como porcentaje. El punto de registro predeterminado es (0,0).

### Véase también

[TextField.\\_x](#), [TextField.\\_xscale](#), [TextField.\\_y](#)

## Clase TextFormat

### Disponibilidad

Flash Player 6.

### Descripción

La clase TextFormat representa la información de formato de caracteres.

Antes de llamar a sus métodos, debe utilizar el constructor `new TextFormat()` para crear un objeto TextFormat.

Puede establecer los parámetros TextFormat en `null` para indicar que no están definidos. Al aplicar un objeto TextFormat a un campo de texto con el método [TextField.setTextFormat\(\)](#), sólo se aplican sus propiedades definidas, como ocurre en el ejemplo siguiente:

```
my_fmt = new TextFormat();
my_fmt.bold = true;
my_txt.setTextFormat(my_fmt);
```

Primero, este código crea un objeto TextFormat vacío con todas sus propiedades sin definir y, a continuación, establece la propiedad `bold` en un valor definido.

El código `my_txt.setTextFormat(my_fmt)` sólo cambia la propiedad `bold` del formato de texto predeterminado del campo de texto, ya que la propiedad `bold` es la única definida de `my_fmt`. Todos los demás aspectos del formato de texto predeterminado del campo de texto permanecen iguales.

Cuando se invoca [TextField.getTextFormat\(\)](#), se devuelve un objeto TextFormat con todas sus propiedades definidas; ninguna propiedad tiene el valor `null`.

## Resumen de métodos para la clase TextFormat

Método	Descripción
<a href="#">TextFormat.getTextExtent()</a>	Devuelve información sobre medidas del texto correspondiente a una cadena de texto.

## Resumen de propiedades para la clase TextFormat

Propiedad	Descripción
<a href="#">TextFormat.align</a>	Indica la alineación de un párrafo.
<a href="#">TextFormat.blockIndent</a>	Indica la sangría de bloque, expresada en puntos.
<a href="#">TextFormat.bold</a>	Indica si el texto está en negrita.
<a href="#">TextFormat.bullet</a>	Indica si el texto se encuentra en una lista con viñetas.

Propiedad	Descripción
<code>TextFormat.color</code>	Indica el color del texto.
<code>TextFormat.font</code>	Indica el nombre de fuente del texto con un formato de texto.
<code>TextFormat.indent</code>	Indica la sangría desde el margen izquierdo al primer carácter del párrafo.
<code>TextFormat.italic</code>	Indica si el texto está en cursiva.
<code>TextFormat.leading</code>	Indica la cantidad de espacio vertical ( <i>interlineado</i> ) entre las líneas.
<code>TextFormat.leftMargin</code>	Indica el margen izquierdo del párrafo, en puntos.
<code>TextFormat.rightMargin</code>	Indica el margen derecho del párrafo, en puntos.
<code>TextFormat.size</code>	Indica el tamaño del texto en puntos.
<code>TextFormat.tabStops</code>	Especifica tabulaciones personalizadas.
<code>TextFormat.target</code>	Indica la ventana de un navegador en la que aparece un hipervínculo.
<code>TextFormat.underline</code>	Indica si el texto está subrayado.
<code>TextFormat.url</code>	Indica la URL a la que se vincula el texto.

## Constructor para la clase TextFormat

### Disponibilidad

Flash Player 6.

### Sintaxis

```
new TextFormat([font, [size, [color, [bold, [italic, [underline, [url,
    [target, [align, [leftMargin, [rightMargin, [indent, [leading]]]]]]]]]]))
```

### Parámetros

*font* Nombre de una fuente para el texto en forma de cadena.

*size* Número entero que indica el tamaño en puntos.

*color* Color de texto que utiliza este formato de texto. Número que contiene tres componentes RGB de 8 bits; por ejemplo, 0xFF0000 es rojo y 0x00FF00 es verde.

*bold* Valor booleano que indica si el texto está en negrita.

*italic* Valor booleano que indica si el texto está en cursiva.

*underline* Valor booleano que indica si el texto está subrayado.

*url* URL con la que el texto de este formato tiene un hipervínculo. Si *url* es una cadena vacía, el texto no tiene hipervínculo.

*target* Ventana de destino en la que se muestra el hipervínculo. Si la ventana de destino es una cadena vacía, el texto se muestra en la ventana de destino predeterminada `_self`. Si se establece el parámetro *url* en una cadena vacía o en el valor `null`, puede obtener o establecer esta propiedad, pero la propiedad no tendrá ningún efecto.

*align* Alineación del párrafo, representado como una cadena. Si su valor es "left", el párrafo se alinea a la izquierda. Si su valor es "center", el párrafo se centra. Si su valor es "right", el párrafo se alinea a la derecha.

*leftMargin* Indica el margen izquierdo del párrafo, expresado en puntos.

*rightMargin* Indica el margen derecho del párrafo, expresado en puntos.

*indent* Número entero que indica la sangría desde el margen izquierdo al primer carácter del párrafo.

*leading* Número que indica la cantidad de espacio vertical entre líneas.

#### Valor devuelto

Ninguno.

#### Descripción

Constructor; crea un objeto `TextFormat` con las propiedades especificadas. A continuación, puede cambiar las propiedades del objeto `TextFormat` para cambiar el formato de los campos de texto.

Cualquier parámetro puede establecerse en el valor `null` para indicar que no está definido. Todos los parámetros son opcionales; todos los parámetros omitidos se tratarán como si tuvieran el valor `null`.

## TextFormat.align

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_fmt.align
```

#### Descripción

Propiedad; indica la alineación del párrafo, que se representa como una cadena. Alineación del párrafo, que se representa como una cadena. Si su valor es "left", el párrafo se alinea a la izquierda. Si su valor es "center", el párrafo se centra. Si su valor es "right", el párrafo se alinea a la derecha. El valor predeterminado es `null` e indica que la propiedad no está definida.

## TextFormat.blockIndent

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_fmt.blockIndent
```

#### Descripción

Propiedad; número que indica la sangría de bloque en puntos. La sangría de bloque se aplica a todo un bloque de texto; es decir, a todas las líneas del texto. En cambio, la sangría normal (`TextFormat.indent`) sólo afecta a la primera línea de cada párrafo. Si el valor de esta propiedad es `null`, el objeto `TextFormat` no especifica sangría de bloque.



## TextFormat.bold

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.bold
```

### Descripción

Propiedad; valor booleano que indica si el texto está en negrita. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.bullet

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.bullet
```

### Descripción

Propiedad; valor booleano que indica que el texto forma parte de una lista con viñetas. En una lista con viñetas, todos los párrafos del texto tienen sangría. A la izquierda de la primera línea de cada párrafo, aparece un símbolo de viñeta. El valor predeterminado es `null`.

## TextFormat.color

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.color
```

### Descripción

Propiedad; indica el color del texto. Número que contiene tres componentes RGB de 8 bits; por ejemplo, `0xFF0000` es rojo y `0x00FF00` es verde.

## TextFormat.font

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.font
```

### Descripción

Propiedad; nombre de la fuente para el texto de este formato de texto, en forma de cadena. El valor predeterminado es `null`, que indica que la propiedad no está definida.

# TextFormat.getTextExtent()

## Disponibilidad

Flash Player 6. El parámetro *width* opcional es compatible con Flash Player 7.

## Sintaxis

```
my_fmt.getTextExtent(text, [width])
```

## Parámetros

*text* Una cadena.

*width* Número opcional que representa la anchura, expresada en píxeles, a la que debe ajustarse el texto especificado.

## Valor devuelto

Un objeto con las propiedades *width*, *height*, *ascent*, *descent*, *textFieldHeight*, *textFieldWidth*.

## Descripción

Método; devuelve información sobre las medidas del texto para la cadena de texto *text* en el formato especificado por *my\_fmt*. La cadena *text* se trata como texto simple (no como texto HTML).

El método devuelve un objeto con seis propiedades: *ascent*, *descent*, *width*, *height*, *textFieldHeight* y *textFieldWidth*. Todas las medidas se expresan en píxeles.

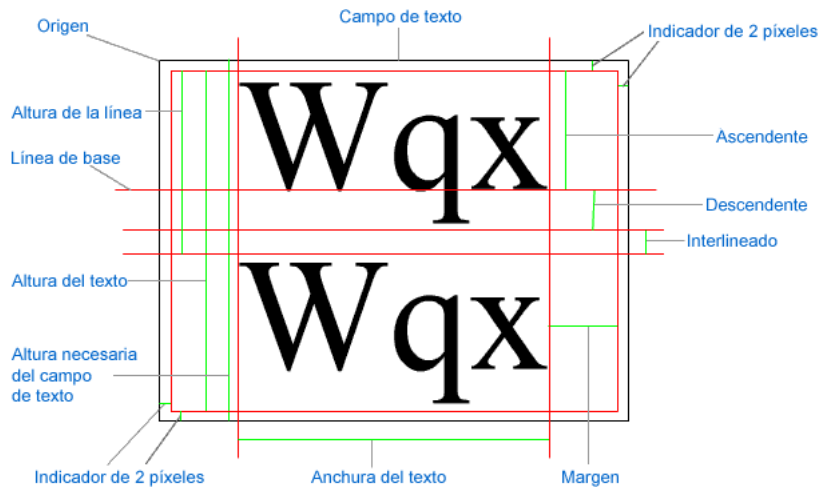
Si se especifica un parámetro *width*, el ajuste de texto se aplica al texto especificado. Esto permite determinar la altura a la que un cuadro de texto mostrará todo el texto especificado.

Las medidas *ascent* y *descent* proporcionan, respectivamente, la distancia por encima y por debajo de la línea de base para una línea de texto. La línea de base para la primera línea de texto se coloca en el origen del campo de texto más su medida *ascent*.

Las medidas *width* y *height* proporcionan la anchura y la altura de la cadena de texto. Las medidas *textFieldHeight* y *textFieldWidth* proporcionan la altura y anchura necesarias para que un objeto de campo texto pueda mostrar toda la cadena de texto. Los campos de texto tienen un “indicador” de 2 píxeles de anchura, de modo que el valor de *textFieldHeight* es igual al valor de *height* + 4; del mismo modo, el valor de *textFieldWidth* siempre es igual al valor de *width* + 4.

Si crea un campo de texto en función de la métrica del texto, utilice *textFieldHeight* en lugar de *height*, y *textFieldWidth*, en lugar de *width*.

En la figura siguiente se ilustran estas medidas.



Cuando configure el objeto `TextFormat`, debe establecer todos los atributos exactamente como se establecerán para crear el campo de texto, incluidos el nombre de fuente, el tamaño de fuente y el interlineado. El valor predeterminado para el interlineado es 2.

### Ejemplo

En este ejemplo se crea un campo de texto de una sola línea que es lo suficientemente grande para mostrar una cadena de texto utilizando el formato especificado.

```
var text = "Cadena corta";

// Crear un objeto TextFormat
// y aplicar sus propiedades.
var txt_fmt = new TextFormat();
with(txt_fmt) {
    font = "Arial";
    bold = true;
}

// Obtener información sobre la métrica para la cadena de texto
// con el formato especificado.
var metrics = txt_fmt.getTextExtent(text);

// Crear un campo de texto lo suficientemente grande como para visualizar el
// texto.
this.createTextField("textField", 0, 100, 100, metrics.textFieldWidth,
    metrics.textFieldHeight);
textField.border = true;
textField.wordWrap = true;
// Asignar la misma cadena de texto y
// el objeto TextFormat al objeto TextField.
textField.text = text;
textField.setTextFormat(txt_fmt);
```

En el ejemplo siguiente se crea un campo de texto multilinea de 100 píxeles de ancho que tiene la altura suficiente para mostrar una cadena con el formato especificado.

```
// Crear un objeto TextFormat.
var txt_fmt:TextFormat= new TextFormat();

// Especificar propiedades de formato para el objeto TextFormat:
txt_fmt.font = "Arial";
txt_fmt.bold = true;
txt_fmt.leading = 4;

// Cadena de texto que debe mostrarse
var textToDisplay:String = "Macromedia Flash 7 incluye métricas de texto
mejoradas.";

// Obtener información de las medidas del texto para la cadena,
// ajustada a 100 píxeles.
var metrics:Object = txt_fmt.getTextExtent(textToDisplay, 100);

// Crear un nuevo objeto TextField utilizando la
// información sobre métrica que se acaba de obtener.
this.createTextField ("textField", 0, 50, 50-metrics.ascent, 100,
    metrics.textFieldHeight)
textField.wordWrap = true;
textField.border = true;
// Asignar el texto y el objeto TextFormat a TextObject:
textField.text = textToDisplay;
textField.setTextFormat(aformat);
```

## TextFormat.indent

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.indent
```

### Descripción

Propiedad; número entero que indica la sangría desde el margen izquierdo al primer carácter del párrafo. El valor predeterminado es `null`, que indica que la propiedad no está definida.

### Véase también

[TextFormat.blockIndent](#)

## TextFormat.italic

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.italic
```

### Descripción

Propiedad; valor booleano que indica si el texto del formato de texto está en cursiva. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.leading

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.leading
```

### Descripción

Propiedad; cantidad de espacio vertical (*interlineado*) entre las líneas. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.leftMargin

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.leftMargin
```

### Descripción

Propiedad; margen izquierdo del párrafo, en puntos. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.rightMargin

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.rightMargin
```

### Descripción

Propiedad; margen derecho del párrafo, en puntos. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.size

### Disponibilidad

Flash Player 6.

### Sintaxis

```
my_fmt.size
```

### Descripción

Propiedad; tamaño en puntos del texto de este formato de texto. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.tabStops

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_fmt.tabStops*

### Descripción

Propiedad; especifica tabulaciones personalizadas en forma de matriz de números enteros no negativos. Cada tabulación se especifica en puntos. Si no se especifican tabulaciones personalizadas (`null`), la tabulación predeterminada es 4 (promedio de anchura de carácter).

## TextFormat.target

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_fmt.target*

### Descripción

Propiedad; indica la ventana de destino en la que se muestra el hipervínculo. Si la ventana de destino es una cadena vacía, el texto se muestra en la ventana de destino predeterminada `_self`. Si la propiedad `TextFormat.url` es una cadena vacía o `null`, puede obtener o establecer esta propiedad, pero la propiedad no tendrá ningún efecto.

## TextFormat.underline

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_fmt.underline*

### Descripción

Propiedad; valor booleano que indica si el texto que utiliza este formato de texto está subrayado (`true`) o no (`false`). Es un subrayado similar al que se obtiene mediante la etiqueta `<U>`, pero este último no es un subrayado “verdadero”, ya que no omite los trazos descendentes correctamente. El valor predeterminado es `null`, que indica que la propiedad no está definida.

## TextFormat.url

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_fmt.url*

## Descripción

Propiedad; indica la URL con la que el texto de este formato tiene un hipervínculo. Si la propiedad `url` es una cadena vacía, el texto no tiene ningún hipervínculo. El valor predeterminado es `null`, que indica que la propiedad no está definida.

# Objeto TextSnapshot

## Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

## Descripción

Los objetos TextSnapshot le permiten trabajar con texto estático en un clip de película. Puede utilizarlos, por ejemplo, para diseñar texto con mayor precisión de lo que permite el texto dinámico, y mantener el texto como de sólo lectura.

No debe utilizar un constructor para crear un objeto TextSnapshot; `MovieClip.getTextSnapshot()` lo devuelve.

## Resumen de los métodos del objeto TextSnapshot

Método	Descripción
<code>TextSnapshot.findText()</code>	Devuelve la posición de la primera aparición del texto especificado.
<code>TextSnapshot.getCount()</code>	Devuelve el número de caracteres.
<code>TextSnapshot.getSelected()</code>	Especifica si alguna parte del segmento de texto especificado ha sido seleccionado por <code>TextSnapshot.setSelected()</code> .
<code>TextSnapshot.getSelectedText()</code>	Devuelve una cadena que contiene todos los caracteres que ha especificado <code>TextSnapshot.setSelected()</code> .
<code>TextSnapshot.getText()</code>	Devuelve una cadena que contiene todos los caracteres del rango especificado.
<code>TextSnapshot.hitTestTextNearPos()</code>	Permite determinar el carácter del objeto que se encuentra en las coordenadas especificadas o próximo a ellas.
<code>TextSnapshot.setSelectColor()</code>	Especifica el color que debe utilizarse al resaltar los caracteres que se han seleccionado mediante el comando <code>TextSnapshot.setSelected()</code> .
<code>TextSnapshot.setSelected()</code>	Especifica un rango de caracteres que debe seleccionarse o deseleccionarse.

## TextSnapshot.findText()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
my_snap.findText( startIndex, textToFind, caseSensitive )
```

### Parámetros

*startIndex* Número entero que especifica el punto inicial en *my\_snap* para buscar el texto especificado.

*textToFind* Cadena que especifica el texto que debe buscarse. Si especifica una literal de cadena en lugar de una variable de tipo String, escriba la cadena entre comillas.

*caseSensitive* Valor booleano que especifica si el texto de *my\_snap* debe coincidir en mayúsculas y minúsculas con la cadena de *textToFind*.

### Valor devuelto

La posición del índice basado en cero de la primera aparición del texto especificado o -1.

### Descripción

Método; busca en el objeto TextSnapshot especificado y devuelve la posición de la aparición de *textToFind* que se encuentre en *startIndex* o después del mismo. Si no se encuentra *textToFind* el método devuelve -1.

### Véase también

[TextSnapshot.getText\(\)](#)

## TextSnapshot.getCount()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
my_snap.getCount()
```

### Parámetros

Ninguno.

### Valor devuelto

Número entero que representa el número de caracteres del objeto TextSnapshot especificado.

### Descripción

Método; devuelve el número de caracteres de un objeto TextSnapshot.



Véase también

[TextSnapshot.getText\(\)](#)

## TextSnapshot.getSelected()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
my_snap.getSelected(from, to)
```

### Parámetros

*from* Número entero que indica la posición del primer carácter de *my\_snap* que debe examinarse. Los valores válidos para *from* van de 0 a `String.lengthTextSnapshot.getCount() - 1`. Si *from* es un valor negativo, se utiliza 0.

*to* Número entero que es 1+ el índice del último carácter de *my\_snap* que debe examinarse. Los valores válidos para *to* van de 0 a `TextSnapshot.getCount()`. El carácter indexado por el parámetro *to* no se incluye en la cadena extraída. Si el parámetro se omite, se utiliza `TextSnapshot.getCount()`. Si este valor es inferior o igual al valor de *from*, se utiliza *from*+1.

### Valor devuelto

Un valor booleano `true`, si el comando `TextSnapshot.setSelected()` correspondiente ha seleccionado al menos un carácter del rango especificado, de lo contrario `false`.

### Descripción

Método; devuelve un valor booleano que especifica si un objeto `TextSnapshot` contiene texto seleccionado en el rango especificado.

Para buscar todos los caracteres, pase un valor de 0 para *from* y `TextSnapshot.getCount()` (o cualquier número grande) para *to*. Para buscar un solo carácter, pase un valor de *from*+1 para *to*.

### Véase también

[TextSnapshot.getSelectedText\(\)](#), [TextSnapshot.getText\(\)](#)

## TextSnapshot.getSelectedText()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
mySnapshot.getSelectedText( [ includeLineEndings ] )
```

## Parámetros

*includeLineEndings* Valor booleano opcional que especifica si se insertan caracteres newline en la cadena devuelta donde corresponda. El valor predeterminado es *false*.

## Valor devuelto

Una cadena que contiene todos los caracteres especificados por el comando `TextSnapshot.setSelected()` correspondiente.

## Descripción

Método; devuelve una cadena que contiene todos los caracteres especificados por el comando `TextSnapshot.setSelected()` correspondiente. Si no se ha seleccionado ningún carácter, se devuelve una cadena vacía.

Si pasa un valor de `true` para *includeLineEndings*, se insertan caracteres newline en la cadena devuelta donde corresponda. En este caso, la cadena devuelta puede ser más larga que el rango de entrada. Si *includeLineEndings* es *false* o está omitido, el texto seleccionado se devuelve sin caracteres añadidos.

## Véase también

`TextSnapshot.getSelected()`

# TextSnapshot.getText()

## Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

## Sintaxis

```
mySnapshot.getText(from, to [, includeLineEndings ] )
```

## Parámetros

*from* Número entero que indica la posición del primer carácter de *my\_snap* que debe incluirse en la cadena devuelta. Los valores válidos para *from* van de 0 a `String.lengthTextSnapshot.getCount()` -1. Si *from* es un valor negativo, se utiliza 0.

*to* Número entero que es 1+ el índice del último carácter de *my\_snap* que debe examinarse. Los valores válidos para *to* van de 0 a `TextSnapshot.getCount()`. El carácter indexado por el parámetro *to* no se incluye en la cadena extraída. Si el parámetro se omite, se utiliza `TextSnapshot.getCount()`. Si este valor es inferior o igual al valor de *from*, se utiliza *from*+1.

*includeLineEndings* Valor booleano opcional que especifica si se insertan caracteres newline en la cadena devuelta donde corresponda. El valor predeterminado es *false*.

## Valor devuelto

Una cadena que contiene los caracteres del rango especificado o una cadena vacía en caso de que no haya caracteres en el rango especificado.

## Descripción

Método; devuelve una cadena que contiene todos los caracteres especificados por los parámetros *from* y *to*. Si no se ha seleccionado ningún carácter, se devuelve una cadena vacía.

Para devolver todos los caracteres, pase un valor de 0 para *from* y `TextSnapshot.getCount()` (o cualquier número grande) para *to*. Para devolver un solo carácter, pase un valor de *from*+1 para *to*.

Si pasa un valor de `true` para *includeLineEndings*, se insertan caracteres `newline` en la cadena devuelta donde corresponda. En este caso, la cadena devuelta puede ser más larga que el rango de entrada. Si *includeLineEndings* es `false` o está omitido, el texto seleccionado se devuelve sin caracteres añadidos.

## Véase también

[TextSnapshot.getSelectedText\(\)](#)

# TextSnapshot.hitTestTextNearPos()

## Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

## Sintaxis

```
my_snap.hitTestTextNearPos(x, y [, maxDistance] )
```

## Parámetros

- x* Número que representa la coordenada x del clip de película que contiene el texto de *my\_snap*.
- y* Número que representa la coordenada y del clip de película que contiene el texto de *my\_snap*.
- maxDistance* Número opcional que representa la distancia máxima desde *x*, *y* en el que puede buscarse texto. La distancia se mide desde el centro de cada carácter. El valor predeterminado es 0.

## Valor devuelto

Un número entero que representa el valor de índice del carácter en *my\_snap* que se encuentra más próximo a las coordenadas *x*, *y* especificadas o -1, si no se encuentra ningún carácter.

## Descripción

Método; permite determinar el carácter del objeto `TextSnapshot` que se encuentra en las coordenadas *x*, *y*, o próximo a ellas, del clip de película que contiene el texto de *my\_snap*.

Si omite o pasa un valor de 0 para *maxDistance*, la ubicación especificada por las coordenadas *x*, *y* debe encontrarse dentro del recuadro de delimitación de *my\_snap*.

## Véase también

[MovieClip.getTextSnapshot\(\)](#), [MovieClip.\\_x](#), [MovieClip.\\_y](#)

## TextSnapshot.setSelectedColor()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
mySnapshot.setSelectedColor(hexColor);
```

### Parámetros

*hexColor* Color utilizado para el borde que rodea a los caracteres que ha seleccionado el comando `TextSnapshot.setSelected()` correspondiente, expresado en el formato 0xRRGGBB.

### Valor devuelto

Ninguno.

### Descripción

Método; especifica el color que debe utilizarse al resaltar los caracteres que se han seleccionado mediante el comando `TextSnapshot.setSelected()`. El color es siempre opaco; no puede especificarse un valor de transparencia.

## TextSnapshot.setSelected()

### Disponibilidad

Edición: Flash MX 2004.

Reproducción: los archivos SWF publicados para FlashPlayer 6 o posterior, que se reproducen en Flash Player 7 o posterior.

### Sintaxis

```
mySnapshot.setSelected(from, to, select)
```

### Parámetros

*from* Número entero que indica la posición del primer carácter de *my\_snap* que debe seleccionarse. Los valores válidos para *from* van de 0 a `String.lengthTextSnapshot.getCount()` -1. Si *from* es un valor negativo, se utiliza 0.

*to* Número entero que es 1+ el índice del último carácter de *my\_snap* que debe examinarse. Los valores válidos para *to* van de 0 a `TextSnapshot.getCount()`. El carácter indexado por el parámetro *to* no se incluye en la cadena extraída. Si el parámetro se omite, se utiliza `TextSnapshot.getCount()`. Si este valor es inferior o igual al valor de *from*, se utiliza *from*+1.

*select* Valor booleano que especifica si el texto debe seleccionarse (`true`) o deseleccionarse (`false`).

### Valor devuelto

Ninguno.

## Descripción

Método; especifica un rango de caracteres de un objeto `TextSnapshot` que debe seleccionarse o deseleccionarse. Los caracteres seleccionados se dibujan sobre un rectángulo de color que coincide con el recuadro de delimitación del carácter. El color del recuadro de delimitación se define por `TextSnapshot.setSelectColor()`.

Para seleccionar o deseleccionar todos los caracteres, pase un valor de 0 para *from* y `TextSnapshot.getCount()` (o cualquier número grande) para *to*. Para especificar un solo carácter, pase un valor de *from*+1 para *to*.

Como los caracteres se marcan de forma individual cuando se seleccionan, puede emitir este comando varias veces para seleccionar varios caracteres; es decir, la utilización de este comando no deselectiona otros caracteres que ya haya establecido este comando.

## this

### Disponibilidad

Flash Player 5.

### Sintaxis

`this`

### Descripción

Identificador; hace referencia a una instancia de objeto o de clip de película. Cuando se ejecuta un script, `this` hace referencia a la instancia de clip de película que contiene el script. Cuando se llama a un método, `this` contiene una referencia al objeto que contiene el método llamado.

Dentro de una acción de controlador de eventos *on* asociada a un botón, `this` se refiere a la línea de tiempo que contiene el botón. Dentro de una acción de controlador de eventos `onClipEvent()` asociada a un clip de película, `this` se refiere a la línea de tiempo del propio clip de película.

Como `this` se evalúa en el contexto del script que lo contiene, no puede utilizar `this` en un script para hacer referencia a una variable definida en un archivo de clase:

```
// en el archivo applyThis.as
class applyThis{
    var str:String = "Definido en applyThis.as";
    function concatStr(x:String):String{
        return x+x;
    }

    function addStr():String{
        return str;
    }
}

// Utilice el código siguiente en FLA para probar la película
import applyThis;

var obj:applyThis = new applyThis();
var abj:applyThis = new applyThis();
abj.str = "definido en FLA";

trace(obj.addStr.call(abj,null));    // definido en FLA
```

```
trace(obj.addStr.call(this,null)); // no definido
trace(obj.addStr.call(obj,null)); // Definido en applyThis.as
```

Asimismo, para llamar a una función definida en una clase dynamic, debe utilizar `this` para el ámbito de la función:

```
// versión incorrecta de simple.as
dynamic class simple{
    function callfunc(){
        trace(func());
    }
}

// versión correcta de simple.as
dynamic class simple{
    function callfunc(){
        trace(this.func());
    }
}
// sentencias en el archivo FLA
import simple;
var obj:simple = new simple();
obj.num = 0;
obj.func = function():Boolean{
    return true;
}
obj.callfunc(); // error de sintaxis con la versión incorrecta de simple.as
```

## Ejemplo

En el ejemplo siguiente, la palabra clave `this` hace referencia al objeto `Circle`.

```
function Circle(radius) {
    this.radius = radius;
    this.area = Math.PI * radius * radius;
}
```

En la sentencia siguiente asignada a un fotograma, la palabra clave `this` hace referencia al clip de película actual.

```
// establece la propiedad alfa del clip de película actual en 20
this._alpha = 20;
```

En la sentencia siguiente dentro de un controlador `onClipEvent()`, la palabra clave `this` hace referencia al clip de película actual.

```
// cuando se carga el clip de película, se inicia una operación startDrag()
// para el clip de película actual.

onClipEvent (load) {
    startDrag (this, true);
}
```

## Véase también

`on()`, `onClipEvent()`

# throw

## Disponibilidad

Flash Player 7.

## Sintaxis

`throw expression`

## Descripción

Sentencia; genera (emite) un error que puede controlarse (detectarse) mediante un bloque de código `catch{} o finally{}`. Si un bloque `catch` o `finally` no detecta una excepción, la representación de cadena del valor emitido se envía al panel Salida.

Generalmente, se emiten instancias de la clase `Error` o de sus subclases (consulte los ejemplos que aparecen a continuación).

## Parámetros

*expression*    Expresión u objeto de `ActionScript`.

## Ejemplo

En este ejemplo, una función denominada `checkEmail()` comprueba si la cadena que se le ha pasado es una dirección de correo electrónico con un formato correcto. Si la cadena no contiene el símbolo `@`, la función emite un error.

```
function checkEmail(email:String) {  
    if (email.indexOf("@") == -1) {  
        throw new Error("Dirección de correo electrónico incorrecta");  
    }  
}
```

En el código siguiente se llama a la función `checkEmail()` en el bloque de código `try` y se pasa el texto de un campo de texto (`email_txt`) como parámetro. Si el parámetro de cadena no contiene una dirección de correo electrónico válida, el mensaje de error se muestra en un campo de texto (`error_txt`).

```
try {  
    checkEmail("Joe Smith");  
} catch (e) {  
    error_txt.text = e.toString();  
}
```

en este ejemplo, se emite una subclase de la clase `Error`. La función `checkEmail()` se modifica para emitir una instancia de dicha subclase. Para más información, consulte [“Creación de subclases” en la página 169](#).

```
// Definir la subclase Error denominada InvalidEmailError  
// En InvalidEmailError.as:  
class InvalidEmailAddress extends Error {  
    var message = "Dirección de correo incorrecta.";  
}  
  
function checkEmail(email:String) {  
    if (email.indexOf("@") == -1) {  
        throw new InvalidEmailAddress();  
    }  
}
```

**Véase también**

[Clase Error](#), [try..catch..finally](#)

## toggleHighQuality()

### Disponibilidad

Flash 2; desestimado en favor de [\\_quality](#).

### Sintaxis

```
toggleHighQuality()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función desestimada; activa y desactiva el suavizado en Flash Player. El suavizado suaviza los bordes de los objetos y hace más lenta la reproducción del archivo SWF. Esta acción afecta a todos los archivos SWF de Flash Player.

### Ejemplo

El código siguiente podría aplicarse a un botón que, cuando se hace clic en él, activa o desactiva el suavizado:

```
on(release) {  
    toggleHighQuality();  
}
```

### Véase también

[\\_highquality](#), [\\_quality](#)

## trace()

### Disponibilidad

Flash Player 4.

### Sintaxis

```
trace(expression)
```

### Parámetros

*expression* Expresión cuyo valor debe calcularse. Cuando se abre un archivo SWF en la herramienta de edición de Flash (mediante el comando Probar película), el valor del parámetro *expression* se muestra en el panel Salida.

### Valor devuelto

Ninguno.



## Descripción

Sentencia; calcula el valor de la expresión y muestra el resultado en el panel Salida en modo de prueba.

Utilice esta acción para registrar notas de programación o para visualizar mensajes en el panel Salida cuando esté probando una película. El parámetro *expression* sirve para comprobar si existe una condición o para ver valores en el panel Salida. La acción `trace()` es similar a la función `alert` de JavaScript.

Puede utilizar el comando Omitir acciones de seguimiento de Configuración de publicación para eliminar acciones `trace()` del archivo SWF exportado.

## Ejemplo

Este ejemplo corresponde a un juego en el que una instancia de clip de película que se puede arrastrar llamada `my_mc` debe soltarse en un destino específico. Una sentencia condicional comprueba el valor de la propiedad `_droptarget` y ejecuta diferentes acciones dependiendo de donde se suelta la instancia de clip de película `my_mc`. Al final del script se utiliza la acción `trace()` para calcular la ubicación del clip de película `my_mc` y para mostrar el resultado en el panel Salida. Si el comportamiento de `my_mc` no es el esperado (por ejemplo, si se ajusta en el destino incorrecto) los valores que la acción `trace()` ha enviado al panel Salida ayudarán a determinar el problema del script.

```
on(press){
    my_mc.startDrag();
}

on(release) {
    if(eval(_droptarget) != target) {
        my_mc._x = my_mc.xValue;
        my_mc._y = my_mc.yValue;
    } else {
        var my_mc_xValue = my_mc._x;
        var my_mc_yValue = my_mc._y;
        target = "_root.pasture";
    }
    trace("my_mc_xValue = " + my_mc_xValue);
    trace("my_mc_yValue = " + my_mc_yValue);
    stopDrag();
}
```

## true

### Disponibilidad

Flash Player 5.

### Sintaxis

`true`

### Descripción

Constante; valor booleano único que representa lo contrario de `false`.

### Véase también

`false`

# try..catch..finally

## Disponibilidad

Flash Player 7.

## Sintaxis

```
try {  
    // ... bloque try ...  
} finally {  
    // ... bloque finally ...  
}  
  
try {  
    // ... bloque try ...  
} catch(error[:ErrorType1]) {  
    // ... bloque catch ...  
} [catch(error[:ErrorTypeN]) {  
    // ... bloque catch ...  
}] [finally {  
    // ... bloque finally ...  
}]
```

## Parámetros

*error* Expresión emitida desde una sentencia *throw*, que generalmente es una instancia de la clase *Error* o una subclase de la misma.

*ErrorType* Especificador de tipo opcional para el identificador *error*. La cláusula *catch* sólo detecta errores del tipo especificado.

## Descripción

Palabras clave; incluya un bloque de código en el que pueda producirse un error y, a continuación, responda al error. Si el código del bloque *try* emite un error (utilizando la acción *throw*), el control pasa al bloque *catch*, si existe, y, a continuación, al bloque *finally*, si existe. El bloque *finally* siempre se ejecuta, independientemente de que se emita un error o no. Si el código del bloque *try* no emite un error (es decir, el bloque *try* finaliza correctamente), se ejecuta el código del bloque *finally*. El bloque *finally* se ejecuta aun cuando el bloque *try* salga utilizando una sentencia *return*.

Un bloque *try* debe ir seguido por un bloque *catch*, un bloque *finally*, o ambos. Un mismo bloque *try* puede tener varios bloques *catch*, pero sólo un bloque *finally*. Puede anidar bloques *try* en tantos niveles de profundidad como desee.

El parámetro *error* especificado en un controlador *catch* debe ser un identificador simple como *e*, *theException* o *x*. La variable de un controlador *catch* también puede ser *typed*. Cuando se utilizan con varios bloques *catch*, los errores de tipo permiten detectar varios tipos de errores emitidos desde un mismo bloque *try*.

Si la excepción emitida es un objeto, el tipo coincidirá si el objeto emitido es una subclase del tipo especificado. Si se emite un error de un tipo específico, se ejecuta el bloque *catch* que controla el error correspondiente. Si se emite una excepción que no es del tipo especificado, el bloque *catch* no se ejecuta y la excepción se emite automáticamente desde el bloque *try* a un controlador *catch* del mismo tipo.

Si se emite un error desde una función y ésta no incluye un controlador `catch`, el intérprete de `ActionScript` sale de dicha función y de todas las funciones de llamada hasta que se encuentre un bloque `catch`. Durante este proceso, se llama a los controladores `finally` en todos los niveles.

## Ejemplo

En este ejemplo se muestra cómo crear una sentencia `try..finally`. Puesto que el código del bloque `finally` se ejecutará de todos modos, generalmente se utiliza para ejecutar el código de “limpieza” necesario tras ejecutar un bloque `try`. En este ejemplo, el bloque `finally` se utiliza para eliminar un objeto `ActionScript`, independientemente de si se ha producido un error.

```
var account = new Account()
try {
    var returnVal = account.getAccountInfo();
    if(returnVal != 0) {
        throw new Error("Error al obtener información sobre la cuenta.");
    }
}
finally {
    // Eliminar el objeto 'account' en cualquier circunstancia.
    if(account != null) {
        delete account;
    }
}
```

En este ejemplo se muestra el funcionamiento de una sentencia `try..catch`. Se ejecuta el código del bloque `try`. Si algún código del bloque `try` emite una excepción, el control pasa al bloque `catch`, que muestra el mensaje de error en un campo de texto utilizando el método `Error.toString()`.

```
var account = new Account()
try {
    var returnVal = account.getAccountInfo();
    if(returnVal != 0) {
        throw new Error("Error al obtener información sobre la cuenta.");
    }
} catch (e) {
    status_txt.text = e.toString();
}
```

En este ejemplo se muestra un bloque de código `try` con varios bloques de código `catch` de tipo. En función del tipo de error que se produzca, el bloque de código `try` emitirá un tipo de objeto distinto. En este caso, `myRecordSet` es una instancia de una clase (hipotética) denominada `RecordSet` cuyo método `sortRows()` puede emitir dos tipos de error distintos: `RecordSetException` y `MalformedRecord`.

En este ejemplo, los objetos `RecordSetException` y `MalformedRecord` son subclases de la clase `Error`. Cada una está definida en su propio archivo de clase `AS`. Para más información, consulte [Capítulo 9, “Creación de clases con ActionScript 2.0”, en la página 161](#).

```
// En RecordSetException.as:
class RecordSetException extends Error {
    var message = "Excepción de registro."
}
// In MalformedRecord.as:
class MalformedRecord extends Error {
    var message = "Excepción de registro incorrecto.";
}
```

En el método `sortRows()` de la clase `RecordSet`, se emite uno de los objetos de error definidos anteriormente en función del tipo de excepción que se haya producido. En el fragmento de código siguiente se muestra cuál podría ser el aspecto de este código.

```
// En el archivo de clase RecordSet.as...
function sortRows() {
    ...
    if(recordSetErrorCondition) {
        throw new RecordSetException();
    }
    if(malFormedRecordCondition) {
        throw new MalformedRecord();
    }
    ...
}
```

Por último, en otro archivo AS o script FLA, el código siguiente invoca el método `sortRows()` o una instancia de la clase `RecordSet`. Define bloques `catch` para cada tipo de error que emite `sortRows()`.

```
try {
    myRecordSet.sortRows();
} catch (e:RecordSetException) {
    trace("Se ha detectado una excepción de registro");
} catch (e:MalformedRecord) {
    trace("Se ha detectado una excepción de registro incorrecto");
}
```

#### Véase también

[Clase Error](#), [throw](#), [class](#), [extends](#)

## typeof

### Disponibilidad

Flash Player 5.

### Sintaxis

`typeof(expression)`

### Parámetros

*expression*    Cadena, clip de película, botón, objeto o función.

### Descripción

Operador; operador unario situado antes de un solo parámetro. El operador `typeof` hace que el intérprete de Flash calcule la *expression*; el resultado es una cadena que especifica si la expresión es una cadena, un clip de película, un objeto, una función, un número o un valor booleano. En la tabla siguiente se muestran los resultados del operador `typeof` en cada tipo de expresión.

Parámetro	Resultado
Cadena	<code>string</code>
Clip de película	<code>movieclip</code>
Button	<code>objeto</code>

---

Campo de texto	objeto
Number	number
Booleano	boolean
Object	objeto
Función	function

---

## undefined

### Disponibilidad

Flash Player 5.

### Sintaxis

`undefined`

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Valor especial que normalmente se utiliza para indicar que aún no se ha asignado ningún valor a una variable. Una referencia a un valor sin definir devuelve el valor especial `undefined`. El código de `ActionScript` `typeof(undefined)` devuelve la cadena `"undefined"`. El único valor del tipo `undefined` es `undefined`.

En los archivos publicados para Flash Player 6 o anterior, el valor de `undefined.toString()` es `""` (una cadena vacía). En los archivos publicados por Flash Player 7 o posterior, el valor de `undefined.toString()` es `undefined`.

El valor `undefined` es similar al valor especial `null`. Si se comparan los valores `null` y `undefined` con el operador de igualdad, verá que son iguales.

### Ejemplo

En este ejemplo, la variable `x` no se ha declarado y, por lo tanto, tiene el valor `undefined`. En la primera parte del código, el operador de igualdad (`==`) compara el valor de `x` con el valor `undefined` y el resultado se envía al panel Salida. En la segunda parte del código, el operador de igualdad compara los valores `null` y `undefined`.

```
// x aún no se ha declarado
trace ("El valor de x es " + x);
if (x == undefined) {
    trace ("x es undefined");
} else {
    trace ("x no es undefined");
}

trace ("typeof (x) es " + typeof (x));
if (null == undefined) {
    trace ("null y undefined son iguales");
} else {
```

```
    trace ("null y undefined no son iguales");  
}
```

Se muestra el resultado siguiente en el panel Salida.

```
El valor de x es undefined  
x es undefined  
typeof (x) es undefined  
null y undefined son iguales
```

## unescape

### Disponibilidad

Flash Player 5.

### Sintaxis

```
unescape(x)
```

### Parámetros

*x* Cadena con secuencias hexadecimales de escape.

### Valor devuelto

Una cadena descodificada de un parámetro con codificación URL

### Descripción

Función; obtiene una cadena como valor del argumento *x*, descodifica la cadena a partir de un formato URL codificado (convirtiendo todas las secuencias hexadecimales en caracteres ASCII) y devuelve la cadena.

### Ejemplo

En el ejemplo siguiente se ilustra el proceso de conversión de caracteres de escape a caracteres que no son de escape.

```
escape("Hola{[Mundo]}");
```

El resultado con caracteres de escape es el que se muestra a continuación:

```
("Hola%7B%5BMundo%5D%7D");
```

La función `unescape` sirve para volver al formato original:

```
unescape("Hola%7B%5BMundo%5D%7D");
```

El resultado es el que se muestra a continuación:

```
Hola{[Mundo]}
```

## unloadMovie()

### Disponibilidad

Flash Player 3.

### Sintaxis

```
unloadMovie(target)
```

### Parámetros

*target* Ruta de destino de un clip de película.

### Valor devuelto

Ninguno.

### Descripción

Función; elimina un clip de película que se ha cargado mediante `loadMovie()` desde Flash Player. Para descargar una película que se ha cargado mediante `loadMovieNum()`, utilice `unloadMovieNum()` en lugar de `unloadMovie()`.

### Ejemplo

En el ejemplo siguiente se descarga el clip de película `draggable_mc` en la línea de tiempo principal y se carga `movie.swf` en el nivel 4.

```
on(press){
    unloadMovie ("_root.draggable_mc");
    loadMovieNum ("movie.swf", 4);
}
```

En el ejemplo siguiente se descarga la película cargada en el nivel 4.

```
on(press){
    unloadMovieNum (4);
}
```

### Véase también

`loadMovie()`, `MovieClipLoader.unloadClip()`

## unloadMovieNum()

### Disponibilidad

Flash Player 3.

### Sintaxis

```
unloadMovieNum(level)
```

### Parámetros

*level* Nivel (`_levelN`) de una película cargada.

### Valor devuelto

Ninguno.

### Descripción

Función; elimina una película que se ha cargado mediante `loadMovieNum()` desde Flash Player. Para descargar una película que se ha cargado mediante `loadMovie()`, utilice `unloadMovie()` en lugar de `unloadMovieNum()`.

### Véase también

`loadMovie()`, `loadMovieNum()`, `unloadMovie()`

## updateAfterEvent()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
updateAfterEvent()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Función; actualiza la visualización (independientemente de los fotogramas por segundo establecidos para la película) cuando el usuario la llama en un controlador `onClipEvent()` o como parte de una función o un método que el usuario pasa a `setInterval()`. Flash ignora las llamadas a `updateAfterEvent` que no se encuentran en un controlador `onClipEvent()` o que no forman parte de una función o un método pasado a `setInterval()`.

### Véase también

`onClipEvent()`, `setInterval()`

## var

### Disponibilidad

Flash Player 5.

### Sintaxis

```
var variableName [= value1] [...,variableNameN [=valueN]]
```

### Parámetros

*variableName* Un identificador.

*value* Valor asignado a la variable.

### Valor devuelto

Ninguno.

### Descripción

Sentencia; se utiliza para declarar variables locales o de línea de tiempo.

- Si declara las variables en una función, las variables son locales. Se definen para la función y caducan al final de la llamada a la función.
- Si las variables no se declaran dentro de un bloque (`{}`), pero la lista de acciones se ejecutó con una acción `call()`, las variables son locales y caducan al final de la lista actual.



- Si las variables no se declaran dentro de un bloque y la lista de acciones actual no se ejecutó con la acción `call()`, las variables se interpretarán como variables de Línea de tiempo. Sin embargo, no debe utilizar `var` para declarar las variables de Línea de tiempo.

No puede declarar que el ámbito de una variable es otro objeto como variable local:

```
my_array.length = 25; // correcto
var my_array.length = 25; // error de sintaxis
```

Cuando utiliza `var`, puede limitarse a escribir la variable; consulte [“Strict data typing” en la página 40](#)

**Nota:** las clases definidas en scripts externos también admiten ámbitos de variables públicas, privadas y estáticas. Consulte el [Capítulo 9, “Creación de clases con ActionScript 2.0”](#), en la [página 161](#) y `private`, `public`, y `static`.

## Clase Video

### Disponibilidad

Flash Player 6; la posibilidad de reproducir archivos Flash Video (FLV) se ha añadido en Flash Player 7.

### Descripción

La clase Video permite visualizar flujo de vídeo en directo en el escenario sin incluirlo en el archivo SWF. El vídeo se captura mediante `Camera.get()`. En los archivos publicados por Flash Player 7 o posterior, también puede utilizar la clase Video para reproducir archivos Flash Video (FLV) utilizando el protocolo HTTP o desde el sistema de archivos local. Para más información, consulte [“Reproducción dinámica de archivos FLV externos” en la página 205](#), [Clase NetConnection](#) y [Clase NetStream](#).

Un objeto Video puede utilizarse como un clip de película. Al igual que con otros objetos que coloca en el escenario, puede controlar diversas propiedades de los objetos Video. Por ejemplo, puede mover el objeto Video por el escenario utilizando su propiedad `_x` y su propiedad `_y`; puede cambiar su tamaño utilizando sus propiedades `_height` y `_width`, etc.

Para visualizar un flujo de vídeo, coloque primero un objeto Video en el escenario. A continuación, utilice `Video.attachVideo()` para asociar el flujo de vídeo al objeto Video.

### Para colocar un objeto Video en el escenario:

- 1 Si el panel Biblioteca no está visible, seleccione Ventana > Biblioteca para visualizarlo.
- 2 Añada un objeto Video incluido a la biblioteca; para hacerlo, haga clic en el menú Opciones que se encuentra a la derecha de la barra de título del panel Biblioteca y seleccione Nuevo vídeo.
- 3 Arrastre el objeto Video al escenario y utilice el inspector de propiedades para asignarle un nombre de instancia exclusivo, como por ejemplo, `my_video`. No le asigne el nombre Video.

## Resumen de métodos para la clase Video

Método	Descripción
<code>Video.attachVideo()</code>	Especifica el flujo de vídeo que debe visualizarse entre los límites del objeto Video en el escenario.
<code>Video.clear()</code>	Borra la imagen que se está visualizando en el objeto Video.

## Resumen de propiedades para la clase Video

Propiedad	Descripción
<code>Video.deblocking</code>	Especifica el comportamiento del filtro de desbloqueo que el compresor de vídeo aplica según sea necesario cuando se reproduce el vídeo.
<code>Video.height</code>	Sólo lectura; altura del flujo de vídeo, expresada en píxeles.
<code>Video.smoothing</code>	Especifica si el vídeo debe suavizarse (interpolarse) cuando se modifica su escala.
<code>Video.width</code>	Sólo lectura; anchura del flujo de vídeo, expresada en píxeles.

## Video.attachVideo()

### Disponibilidad

Flash Player 6; la posibilidad de trabajar con archivos Flash Video (FLV) se ha añadido en Flash Player 7.

### Sintaxis

```
my_video.attachVideo(source)
```

### Parámetros

*source* Objeto Camera que captura los datos de vídeo o un objeto NetStream. Para abandonar la conexión con el objeto Video, pase el valor `null` para el parámetro *source*.

### Valor devuelto

Ninguno.

### Descripción

Método; especifica el flujo de vídeo (*source*) que debe visualizarse entre los límites del objeto Video en el escenario. El flujo de vídeo es un archivo FLV que se está visualizando mediante el comando `NetStream.play()`, un objeto Camera o `null`. Si el valor de *source* es `null`, el vídeo deja de visualizarse en el objeto Video.

Este método no es necesario si el archivo FLV sólo contiene audio; la porción de audio de los archivos FLV se reproduce automáticamente cuando se emite el comando `NetStream.play()`.

Si desea controlar el sonido asociado a un archivo FLV file, puede usar `MovieClip.attachAudio()` para dirigir el sonido a un clip de película; a continuación, puede crear un objeto Sound para controlar algunos aspectos del sonido. Para más información, consulte `MovieClip.attachAudio()`.

### Ejemplo

El ejemplo siguiente reproduce el vídeo en vivo localmente.

```
my_cam = Camera.get();
my_video.attachVideo(my_cam); // my_video es un objeto Video del escenario
```

En el ejemplo siguiente se reproduce un archivo grabado con anterioridad que se llama `myVideo.flv` y que está almacenado en el mismo directorio que el archivo SWF.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

```
var ns:NetStream = new NetStream(my_nc);  
my_video.attachVideo(ns); // my_video es un objeto Video del escenario  
ns.play("myVideo.flv");
```

#### Véase también

[Clase Camera](#), [Clase NetStream](#)

## Video.clear()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_video.clear()
```

#### Parámetros

Ninguno.

#### Valor devuelto

Ninguno.

#### Descripción

Método; borra la imagen que se está visualizando en el objeto Video. Esto es útil cuando, por ejemplo, se desea visualizar información de espera sin ocultar el objeto Video.

#### Véase también

[Video.attachVideo\(\)](#)

## Video.deblocking

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
my_video.deblocking
```

```
my_video.deblocking = setting
```

#### Descripción

Propiedad; especifica el comportamiento del filtro de desbloqueo que aplica el compresor cuando es necesario al reproducir el vídeo. Los valores aceptables para *setting* son:

- 0 (valor predeterminado): deja que el compresor de vídeo pueda aplicar el filtro de desbloqueo según sea necesario.
- 1: no debe utilizarse nunca el filtro de desbloqueo.
- 2: debe utilizarse siempre el filtro de desbloqueo.

El filtro de desbloqueo tiene un efecto sobre el rendimiento global de reproducción, y generalmente no es necesario para vídeo con un ancho de banda alto. Si el sistema no es suficientemente potente, es posible que tenga dificultades al reproducir el vídeo con este filtro activado.

# Video.height

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_video.height*

## Descripción

Propiedad de sólo lectura; número entero que especifica la altura del flujo de vídeo, expresada en píxeles. Para los flujos en vivo, este valor es el mismo que la propiedad [Camera.height](#) del objeto Camera que captura el flujo de vídeo. Para los archivos FLV, este valor es la altura del archivo que se ha exportado como FLV.

Puede utilizar esta propiedad, por ejemplo, para garantizar que el usuario ve el vídeo con el mismo tamaño que se captura, independientemente del tamaño real del objeto Video del escenario.

## Ejemplo

Sintaxis 1: en el ejemplo siguiente se establecen los valores de altura y anchura del objeto Video de modo que coincidan con los valores de un archivo FLV. Debe llamar a este código después de que se invoque [NetStream.onStatus](#) con una propiedad code de [NetStream.Buffer.Full](#). Si lo llama cuando la propiedad code es [NetStream.Play.Start](#), los valores de altura y anchura serán de 0, porque el objeto Video no tiene todavía la altura y anchura del archivo FLV cargado.

```
// Clip es el nombre de instancia del clip de película
// que contiene el objeto de vídeo "my_video".
_root.Clip._width = _root.Clip.my_video.width;
_root.Clip._height = _root.Clip.my_video.height;
```

Sintaxis 2: en el ejemplo siguiente se deja que el usuario presione un botón para establecer que la altura y la anchura del flujo de vídeo que se está reproduciendo en Flash Player sea la misma que la altura y la anchura a la que se ha capturado el flujo de vídeo.

```
on(release) {
    _root.my_video._width = _root.my_video.width
    _root.my_video._height = _root.my_video.height
}
```

## Véase también

[MovieClip.\\_height](#), [Video.width](#)

# Video.smoothing

## Disponibilidad

Flash Player 6.

## Sintaxis

*my\_video.smoothing*

## Descripción

Propiedad; valor booleano que especifica si el vídeo debe suavizarse (interpolarse) cuando se modifica su escala. Para que el suavizado funcione, el reproductor debe estar en modo de alta calidad. El valor predeterminado es `false` (sin suavizado).

## Video.width

### Disponibilidad

Flash Player 6.

### Sintaxis

*my\_video.width*

### Descripción

Propiedad de sólo lectura; número entero que especifica la anchura del flujo de vídeo, expresada en píxeles. Para los flujos en vivo, este valor es el mismo que la [Camera.width](#) propiedad del objeto Camera que captura el flujo de vídeo. Para los archivos FLV, este valor es la anchura del archivo que se ha exportado como archivo FLV.

Puede utilizar esta propiedad, por ejemplo, para garantizar que el usuario ve el vídeo con el mismo tamaño que se captura, independientemente del tamaño real del objeto Video del escenario.

### Ejemplo

Consulte los ejemplos de [Video.height](#).

## void

### Disponibilidad

Flash Player 5.

### Sintaxis

*void (expression)*

### Descripción

Operador; operador unario que descarta el valor *expresión* y devuelve un valor sin definir. El operador `void` se utiliza a menudo en comparaciones con el operador `==` para comprobar valores sin definir.

## while

### Disponibilidad

Flash Player 4.

### Sintaxis

```
while(condition) {  
    statement(s);  
}
```

### Parámetros

*condition*    Expresión cuyo valor se vuelve a comprobar cada vez que se ejecuta la acción `while`.  
*statement(s)*    Instrucciones que deben ejecutarse si la condición tiene el valor `true`.

### Valor devuelto

Ninguno.

## Descripción

Sentencia; comprueba el valor de una expresión y ejecuta una sentencia o una serie de sentencias indefinidamente mientras el valor de la expresión sea `true`.

Antes de que se ejecute el bloque de sentencia, se comprueba *condition*; si se obtiene como resultado el valor `true`, se ejecuta el bloque de sentencia. Si la condición es `false`, el bloque de sentencia se omite y se ejecuta la primera sentencia después de ejecutar el bloque de sentencia de la acción `while`.

Las reproducciones indefinidas se utilizan con frecuencia para realizar una acción mientras que una variable de contador es menor que un valor especificado. Al final de cada reproducción, el contador se incrementa hasta que se alcanza el valor especificado. En ese momento, *condition* deja de ser `true` y la reproducción termina.

La sentencia `while` efectúa la siguiente serie de pasos. Cada repetición de los pasos 1 al 4 se denomina una *repetición*. Al principio de cada repetición, se vuelve a comprobar el valor de *condition*, tal como se describe en los pasos siguientes:

- 1 Se comprueba el valor de la expresión *condition*.
- 2 Siga con el paso 3 si el valor de la *condition* es `true` o un valor que se convierte en el valor booleano `true`, como por ejemplo, cualquier número distinto de cero.  
En caso contrario, la sentencia `while` finaliza y la ejecución se reanuda en la siguiente sentencia tras la reproducción indefinida `while`.
- 3 Ejecute el bloque de sentencia *statement(s)*.
- 4 Vaya al paso 1.

## Véase también

`do while`, `continue`, `for`, `for..in`

## with

### Disponibilidad

Flash Player 5.

### Sintaxis

```
with (object) {  
    statement(s);  
}
```

### Parámetros

*object*    Instancia de un objeto o clip de película de ActionScript.

*statement(s)*    Acción o grupo de acciones entre llaves.

### Valor devuelto

Ninguno.

## Descripción

Sentencia; permite especificar un objeto (por ejemplo, un clip de película) con el parámetro *object* y comprobar el valor de expresiones y acciones dentro de ese objeto con el parámetro *statement(s)*. Esto evita que tenga que escribir el nombre del objeto o la ruta al objeto continuamente.

El parámetro *object* pasa a ser el contexto en el que se leen las propiedades, variables y funciones en el parámetro *statement(s)*. Por ejemplo, si el *object* es *my\_array* y dos de las propiedades especificadas son *length* y *concat*, dichas propiedades se leen automáticamente como *my\_array.length* y *my\_array.concat*. En otro ejemplo, si el *object* es *state.california*, se llama a las acciones y sentencias de la acción *with* desde la instancia *california*.

Para averiguar el valor de un identificador del parámetro *statement(s)*, ActionScript se inicia al principio de la cadena de ámbito especificada por el *object* y busca el identificador en cada nivel de la cadena de ámbito, en un orden específico.

La cadena de ámbito utilizada por la acción *with* para resolver identificadores comienza con el primer elemento de la lista siguiente y continúa hasta el último:

- El objeto especificado en el parámetro *object* de la acción *with* más interior.
- El objeto especificado en el parámetro *object* de la acción *with* más exterior.
- El objeto Activation (objeto temporal que se crea automáticamente cuando se llama a una función que contiene las variables locales a las que se ha llamado en la función).
- El clip de película contiene el script que se está ejecutando.
- El objeto Global (objetos incorporados, como *Math* y *String*).

Para establecer una variable dentro de una acción *with*, la variable debe haber sido declarada fuera de la acción *with* o debe introducir la ruta de acceso completa a la línea de tiempo en la que desea que resida la variable. Si establece una variable en una acción *with* sin haberla declarado, la acción *with* buscará el valor según la cadena de ámbito. Si la variable todavía no existe, el nuevo valor se establecerá en la línea de tiempo desde la que se llamó a la acción *with*.

En Flash 5 o posterior, la acción *with* sustituye a la acción *tellTarget* que se eliminó. Se recomienda utilizar *with* en lugar de *tellTarget* debido a que es una extensión estándar de ActionScript del estándar ECMA-262. La diferencia principal entre las acciones *with* y *tellTarget* es que *with* toma una referencia a un clip de película o a otro objeto como parámetro, mientras que *tellTarget* toma una cadena de ruta de destino que identifica un clip de película como su parámetro y no se puede utilizar para especificar el destino de objetos.

## Ejemplo

En el ejemplo siguiente se establecen las propiedades *\_x* e *\_y* de la instancia *someOther\_mc* y se indica a *someOther\_mc* que vaya al fotograma 3 y se detenga.

```
with (someOther_mc) {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```

El siguiente fragmento de código muestra cómo escribir el código anterior sin utilizar una acción *with*.

```
someOther_mc._x = 50;
```

```
someOther_mc._y = 100;
someOther_mc.gotoAndStop(3);
```

También podría escribir este código mediante la acción `tellTarget`. Sin embargo, si `someOther_mc` no fuera un clip de película, sino un objeto, no podría utilizar la acción `with`.

```
tellTarget ("someOther_mc") {
    _x = 50;
    _y = 100;
    gotoAndStop(3);
}
```

La acción `with` es útil para acceder a varios elementos de una lista de cadena de ámbito simultáneamente. En el ejemplo siguiente, el objeto incorporado `Math` se sitúa al frente de la cadena de ámbito. Si se establece `Math` como valor predeterminado, el objeto resuelve los identificadores `cos`, `sin` y `PI` en los valores `Math.cos`, `Math.sin` y `Math.PI`, respectivamente. Los identificadores `a`, `x`, `y` y `r` no son métodos ni propiedades del objeto `Math`, pero como existen en el ámbito de activación de objeto de la función `polar()`, se resuelven en las variables locales correspondientes.

```
function polar(r) {
    var a, x, y;
    with (Math) {
        a = PI * r * r;
        x = r * cos(PI);
        y = r * sin(PI/2);
    }
    trace("área = " + a);
    trace("x = " + x);
    trace("y = " + y);
}
```

Puede utilizar acciones `with` anidadas para acceder a la información en varios ámbitos. En el ejemplo siguiente, la instancia `fresno` y la instancia `salinas` son instancias secundarias de la instancia `california`. La sentencia establece los valores `_alpha` de `fresno` y `salinas` sin cambiar el valor `_alpha` de `california`.

```
with (california){
    with (fresno){
        _alpha = 20;
    }
    with (salinas){
        _alpha = 40;
    }
}
```

**Véase también**

[tellTarget](#)

## Clase XML

**Disponibilidad**

Flash Player 5 (pasó a ser un objeto nativo en Flash Player 6, lo cual mejoró el rendimiento notablemente).



## Descripción

Utilice los métodos y las propiedades del objeto XML para cargar, analizar, enviar, construir y manipular árboles de documentos XML.

Debe utilizar el constructor `new XML()` para crear un objeto XML antes de llamar a los métodos de la clase XML.

## Resumen de métodos para la clase XML

Método	Descripción
<code>XML.setRequestHeader()</code>	Añade o cambia los encabezados HTTP para las operaciones POST.
<code>XML.appendChild()</code>	Anexa un nodo al final de la lista de objetos secundarios del objeto especificado.
<code>XML.cloneNode()</code>	Duplica el nodo especificado y, opcionalmente, puede duplicar repetidamente todos los nodos secundarios.
<code>XML.createElement()</code>	Crea un nuevo elemento XML.
<code>XML.createTextNode()</code>	Crea un nuevo nodo de texto XML.
<code>XML.getBytesLoaded()</code>	Devuelve el número de bytes cargados para el documento XML especificado.
<code>XML.getBytesTotal()</code>	Devuelve el tamaño del documento XML, expresado en bytes.
<code>XML.hasChildNodes()</code>	Devuelve <code>true</code> si el nodo especificado tiene nodos secundarios; de lo contrario, devuelve <code>false</code> .
<code>XML.insertBefore()</code>	Inserta un nodo delante de un nodo existente en la lista de nodos secundarios del nodo especificado.
<code>XML.load()</code>	Carga un documento (especificado por el objeto XML) desde una URL.
<code>XML.parseXML()</code>	Analiza un documento XML en el árbol de objeto XML especificado.
<code>XML.removeNode()</code>	Elimina el nodo especificado de su nodo principal.
<code>XML.send()</code>	Envía el objeto XML especificado a una URL.
<code>XML.sendAndLoad()</code>	Envía el objeto XML especificado a una URL y carga la respuesta del servidor en otro objeto XML.
<code>XML.toString()</code>	Convierte el nodo especificado y cualquiera de sus secundarios en texto XML.

## Resumen de propiedades para la clase XML

Propiedad	Descripción
<code>XML.contentType</code>	Indica el tipo MIME transmitido al servidor.
<code>XML.docTypeDecl</code>	Establece y devuelve información sobre la declaración DOCTYPE de un documento XML.
<code>XML.firstChild</code>	Sólo lectura; hace referencia al primer nodo secundario de la lista del nodo especificado.

Propiedad	Descripción
<code>XML.ignoreWhite</code>	Cuando se establecen en <code>true</code> , los nodos de texto que sólo contienen espacios en blanco se descartan durante el proceso de análisis.
<code>XML.lastChild</code>	Hace referencia al último nodo secundario de la lista del nodo especificado.
<code>XML.loaded</code>	Sólo lectura; comprueba si se ha cargado el objeto XML especificado.
<code>XML.nextSibling</code>	Sólo lectura, hace referencia al siguiente nodo colateral en la lista de nodos secundarios del nodo principal.
<code>XML.nodeName</code>	El nombre de nodo de un objeto XML.
<code>XML.nodeType</code>	Tipo del nodo especificado (nodo de texto o elemento XML).
<code>XML.nodeValue</code>	Texto del nodo especificado si el nodo es un nodo de texto.
<code>XML.parentNode</code>	Sólo lectura; hace referencia al nodo principal del nodo especificado.
<code>XML.previousSibling</code>	Sólo lectura; hace referencia al nodo colateral anterior en la lista de nodos secundarios del nodo principal.
<code>XML.status</code>	Código de estado numérico que indica el éxito o el fracaso de una operación de análisis de un documento XML.
<code>XML.xmlDecl</code>	Especifica información sobre una declaración XML de documento.

## Resumen de colecciones para la clase XML

Método	Descripción
<code>XML.attributes</code>	Devuelve una matriz asociativa que contiene todos los atributos del nodo especificado.
<code>XML.childNodes</code>	Sólo lectura; devuelve una matriz que contiene referencias a los nodos secundarios del nodo especificado.

## Resumen de controladores de eventos para la clase XML

Controlador de eventos	Descripción
<code>XML.onData</code>	Un controlador de eventos que se invoca cuando el texto XML se ha descargado completamente del servidor o cuando se produce un error al descargar texto XML de un servidor.
<code>XML.onLoad()</code>	Un controlador de eventos que devuelve un valor booleano que indica si se ha cargado correctamente el objeto XML con <code>XML.load()</code> o <code>XML.sendAndLoad()</code> .

## Constructor para la clase XML

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new XML([source])
```

### Parámetros

*source* Texto XML que se va a analizar para crear un nuevo objeto XML.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto XML. Debe utilizar el constructor para crear un objeto XML antes de llamar a los métodos de la clase XML.

**Nota:** los métodos `createElement()` y `createTextNode()` son los métodos “constructores” para crear los elementos y los nodos de texto en un árbol de documentos XML.

### Ejemplo

Sintaxis 1: en el ejemplo siguiente se crea un nuevo objeto XML vacío.

```
my_xml = new XML();
```

Sintaxis 2: en el ejemplo siguiente se crea un objeto XML analizando el texto XML especificado en el parámetro *source* y se rellena el objeto XML recién creado con el árbol del documento XML resultante.

```
anyOtherXML = new XML("<state>California<city>San Francisco</city></state>");
```

### Véase también

[XML.createElement\(\)](#), [XML.createTextNode\(\)](#)

## XML.setRequestHeader()

### Disponibilidad

Flash Player 6.

### Sintaxis

```
xml.setRequestHeader(headerName, headerValue)
```

```
xml.setRequestHeader(["headerName_1", "headerValue_1" ... "headerName_n",  
"headerValue_n"])
```

### Parámetros

*headerName* Nombre de encabezado de una solicitud HTTP.

*headerValue* Valor asociado con *headerName*.

### Valor devuelto

Ninguno.

## Descripción

Método; añade o cambia encabezados de solicitud HTTP (tales como Content-Type o SOAPAction) enviados con acciones POST. En la primera sintaxis, se pasan dos cadenas al método: *headerName* y *headerValue*. En la segunda sintaxis, se pasa una matriz de cadenas, en las que se alternan los nombres y los valores de encabezado.

Si se realizan varias llamadas para establecer el mismo nombre de encabezado, cada valor sucesivo sustituirá el valor establecido en la llamada anterior.

Si se utiliza este método no es posible añadir ni cambiar los encabezados HTTP estándar siguientes: Accept-Ranges, Age, Allow, Allowed, Connection, Content-Length, Content-Location, Content-Range, ETag, Host, Last-Modified, Locations, Max-Forwards, Proxy-Authenticate, Proxy-Authorization, Public, Range, Retry-After, Server, TE, Trailer, Transfer-Encoding, Upgrade, URI, Vary, Via, Warning y WWW-Authenticate.

## Ejemplo

En este ejemplo se añade un encabezado HTTP personalizado denominado SOAPAction con un valor de Foo a un objeto XML denominado my\_xml.

```
my_xml.setRequestHeader("SOAPAction", "'Foo'");
```

En el ejemplo siguiente, se crea una matriz denominada headers que contiene dos encabezados HTTP alternos y los valores asociados correspondientes. La matriz se pasa como parámetro al método addRequestHeader().

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];
my_xml.setRequestHeader(headers);
```

## Véase también

[LoadVars.setRequestHeader\(\)](#)

# XML.appendChild()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_xml.appendChild(childNode)
```

## Parámetros

*childNode*   Nodo secundario que debe agregarse a la lista de nodos secundarios del objeto XML que se ha especificado.

## Valor devuelto

Ninguno.

## Descripción

Método; anexa el nodo secundario especificado a la lista de nodos secundarios del objeto XML. El nodo anexo se coloca en la estructura del árbol una vez que se ha eliminado de su nodo principal existente, si lo hay.

### Ejemplo

En el ejemplo siguiente se duplica el último nodo de `doc1` y se adjunta a `doc2`.

```
doc1 = new XML(src1);
doc2 = new XML();
node = doc1.lastChild.cloneNode(true);
doc2.appendChild(node);
```

## XML.attributes

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.attributes
```

### Parámetros

Ninguno.

### Valor devuelto

Una matriz.

### Descripción

Propiedad; una matriz asociativa que contiene todos los atributos del objeto XML especificado.

### Ejemplo

En el ejemplo siguiente se muestran los nombres de los atributos XML en la ventana Salida.

```
str = "<mytag name=\"Val\"> item </mytag>";
doc = new XML(str);
y = doc.firstChild.attributes.name;
    trace(y);
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order
    trace(z);
```

Lo que se muestra a continuación es lo que aparece en el panel Salida.

```
Val
first
```

## XML.childNodes

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.childNodes
```

### Parámetros

Ninguno.

**Valor devuelto**

Una matriz.

**Descripción**

Propiedad (sólo lectura); una matriz de los nodos secundarios del objeto XML especificado. Cada elemento de la matriz es una referencia a un objeto XML que representa un nodo secundario. Esta es una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios. Utilice `XML.appendChild()`, `XML.insertBefore()` y `XML.removeNode()` para manipular nodos secundarios.

Esta propiedad no está definida para nodos de texto (`nodeType == 3`).

**Véase también**

[XML.nodeType](#)

## XML.cloneNode()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
my_xml.cloneNode(deep)
```

**Parámetros**

*deep* Valor booleano que especifica si los valores secundarios del objeto XML especificado se duplican repetidamente.

**Valor devuelto**

Un nodo XML.

**Descripción**

Método; construye y devuelve un nuevo nodo XML del mismo tipo, nombre, valor y atributos que el objeto XML especificado. Si *deep* está establecido en `true`, todos los nodos secundarios se duplican repetidamente, lo que da como resultado una copia exacta del árbol de documento del objeto original.

El duplicado del nodo que se devuelve deja de estar asociado con el árbol del elemento duplicado. En consecuencia, `nextSibling`, `parentNode` y `previousSibling` tienen el valor `null`. Si no se efectúa una copia del `clip`, `firstChild` y `lastChild` también son `null`.

## XML.contentType

**Disponibilidad**

Flash Player 6.

**Sintaxis**

```
my_xml.contentType
```

### Descripción

Propiedad; tipo MIME que se envía al servidor al llamar al método `XML.send()` o `XML.sendAndLoad()`. El valor predeterminado es *application/x-www-form-urlencoded*.

### Véase también

`XML.send()`, `XML.sendAndLoad()`

## XML.createElement()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.createElement( name )
```

### Parámetros

*name* Nombre de etiqueta del elemento XML que se está creando.

### Valor devuelto

Un elemento XML.

### Descripción

Método; crea un nuevo elemento XML con el nombre especificado en el parámetro. Inicialmente, el nuevo elemento no tiene ningún elemento principal, secundario ni colateral. El método devuelve una referencia al objeto XML recién creado que representa el elemento. Este método y `createTextNode()` son los métodos constructor para crear nodos para un objeto XML.

## XML.createTextNode()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.createTextNode( text )
```

### Parámetros

*text* Texto utilizado para crear el nuevo nodo de texto.

### Valor devuelto

Ninguno.

### Descripción

Método; crea un nuevo nodo de texto XML con el texto especificado. Inicialmente, el nuevo nodo no tiene nodo principal y los nodos de texto no pueden tener nodos secundarios ni colaterales. Este método devuelve una referencia al objeto XML que representa el nuevo nodo de texto. Este método y `createElement()` son los métodos constructor para crear nodos para un objeto XML.

# XML.docTypeDecl

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_xml.XMLdocTypeDecl
```

## Descripción

Propiedad; especifica información sobre la declaración DOCTYPE del documento XML. Una vez analizado el texto XML de un objeto XML, la propiedad XML.docTypeDecl del objeto XML se establece en el texto de la declaración DOCTYPE del documento XML. Por ejemplo, <!DOCTYPE greeting SYSTEM "hello.dtd">. Esta propiedad se establece utilizando una representación de cadena de la declaración DOCTYPE, no un objeto de nodo XML.

El analizador XML de ActionScript no es un analizador de validación. El analizador lee la declaración DOCTYPE que, a continuación, se almacena en la propiedad docTypeDecl, pero no se realiza ninguna validación DTD.

Si no se detecta ninguna declaración DOCTYPE durante la operación de análisis, XML.docTypeDecl se establece en undefined. XML.toString() muestra el contenido de XML.docTypeDecl inmediatamente después de la declaración XML almacenada en XML.xmlDecl y antes de cualquier otro texto del objeto XML. Si la propiedad XML.docTypeDecl no está definida, no se muestra ninguna declaración DOCTYPE.

## Ejemplo

En el ejemplo siguiente se utiliza XML.docTypeDecl para establecer la declaración DOCTYPE de un objeto XML:

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

## Véase también

[XML.toString\(\)](#), [XML.xmlDecl](#)

# XML.firstChild

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_xml.firstChild
```

## Descripción

Propiedad (sólo lectura); evalúa el objeto XML especificado y hace referencia al primer nodo secundario en la lista de nodos secundarios del nodo principal. Esta propiedad tiene el valor null si el nodo no tiene nodos secundarios. Esta propiedad está sin definir si el nodo es un nodo de texto. Esta es una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios; utilice appendChild(), insertBefore() y removeNode() para manipular nodos secundarios.



#### Véase también

[XML.appendChild\(\)](#), [XML.insertBefore\(\)](#), [XML.removeNode\(\)](#)

## XML.getBytesLoaded()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
XML.getBytesLoaded()
```

#### Parámetros

Ninguno.

#### Valor devuelto

Entero que indica el número de bytes cargados.

#### Descripción

Método; devuelve el número de bytes cargados (en flujo) para el documento XML. Puede comparar el valor de `getBytesLoaded()` con el valor de `getBytesTotal()` para determinar el porcentaje que se ha cargado de un documento XML.

#### Véase también

[XML.getBytesTotal\(\)](#)

## XML.getBytesTotal()

#### Disponibilidad

Flash Player 6.

#### Sintaxis

```
XML.getBytesTotal()
```

#### Parámetros

Ninguno.

#### Valor devuelto

Un número entero.

#### Descripción

Método; devuelve el tamaño, en bytes, del documento XML.

#### Véase también

[XML.getBytesLoaded\(\)](#)

## XML.hasChildNodes()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.hasChildNodes()
```

### Parámetros

Ninguno.

### Valor devuelto

Valor booleano.

### Descripción

Método; devuelve el valor `true` si el objeto XML especificado tiene nodos secundarios; en caso contrario, devuelve el valor `false`.

### Ejemplo

En el ejemplo siguiente se utiliza la información del objeto XML en una función definida por el usuario.

```
if (rootNode.hasChildNodes()) {  
    myfunc (rootNode.firstChild);  
}
```

## XML.ignoreWhite

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.ignoreWhite = boolean  
XML.prototype.ignoreWhite = boolean
```

### Parámetros

*boolean* Valor booleano (`true` o `false`).

### Descripción

Propiedad; el valor predeterminado es `false`. Cuando se establecen en `true`, los nodos de texto que sólo contienen espacios en blanco se descartan durante el proceso de análisis. Los nodos de texto con espacio en blanco al principio o al final no se ven afectados.

Sintaxis 1: puede establecer la propiedad `ignoreWhite` para objetos XML individuales, como se muestra en el código siguiente:

```
my_xml.ignoreWhite = true
```

Sintaxis 2: puede establecer la propiedad predeterminada `ignoreWhite` para los objetos XML, como se muestra en el código siguiente:

```
XML.prototype.ignoreWhite = true
```

## XML.insertBefore()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.insertBefore(childNode, beforeNode)
```

### Parámetros

*childNode*    Nodo que se va a insertar.

*beforeNode*    Nodo situado delante del punto de inserción de *childNode*.

### Valor devuelto

Ninguno.

### Descripción

Método; inserta un nuevo nodo secundario en la lista de nodos secundarios del objeto XML, antes de *beforeNode*. Si el parámetro *beforeNode* no está definido o tiene el valor `null`, el nodo se agrega con el método `appendChild()`. Si *beforeNode* no es un nodo secundario de *my\_xml*, la inserción no puede realizarse.

## XML.lastChild

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.lastChild
```

### Descripción

Propiedad (sólo lectura); evalúa el objeto XML y hace referencia al último nodo secundario en la lista de nodos secundarios del nodo principal. Este método devuelve el valor `null` si el nodo no tiene nodos secundarios. Esta es una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios; utilice `appendChild()`, `insertBefore()` y `removeNode()` para manipular nodos secundarios.

### Véase también

[XML.appendChild\(\)](#), [XML.insertBefore\(\)](#), [XML.removeNode\(\)](#)

## XML.load()

### Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

### Sintaxis

```
my_xml.load(url)
```

## Parámetros

*url* URL en la que se encuentra el documento XML que debe cargarse. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

## Valor devuelto

Ninguno.

## Descripción

Método; carga un documento XML desde la URL especificada y reemplaza el contenido del objeto XML especificado con los datos XML descargados. La URL es relativa, y se llama a través de HTTP. El proceso de carga es asíncrono; no finaliza inmediatamente después de que se ejecute el método `load()`.

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables desde archivos SWF que también estén en `www.someDomain.com`. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Cuando se ejecuta el método `load()`, la propiedad del objeto XML `loaded` se establece en `false`. Cuando finaliza la descarga de los datos XML, la propiedad `loaded` se establece en `true` y se invoca el método `onLoad()`. Los datos XML no se analizan hasta que no se han descargado por completo. Si el objeto XML contenía anteriormente algún árbol XML, se pasa por alto.

Puede especificar su propio controlador de eventos en lugar del método `onLoad()`.

## Ejemplo

A continuación, se muestra un ejemplo simple de uso de `XML.load()`:

```
doc = new XML();  
doc.load ("theFile.xml");
```

## Véase también

[XML.loaded](#), [XML.onLoad\(\)](#)

# XML.loaded

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_xml.loaded
```

## Descripción

Propiedad (sólo lectura); determina si el proceso de carga de documentos iniciado por la llamada a `XML.load()` ha finalizado. Si el proceso se completa correctamente, el método devuelve el valor `true`; en caso contrario, devuelve el valor `false`.

## Ejemplo

En el ejemplo siguiente, se utiliza `XML.loaded` en un script sencillo.

```
if (doc.loaded) {  
    gotoAndPlay(4);  
}
```

# XML.nextSibling

## Disponibilidad

Flash Player 5.

## Sintaxis

`my_xml.nextSibling`

## Descripción

Propiedad (sólo lectura); evalúa el objeto XML y hace referencia al siguiente colateral en la lista de nodos secundarios del nodo principal. Este método devuelve el valor `null` si el nodo no tiene un nodo colateral a continuación. Esta es una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios. Utilice `appendChild()`, `insertBefore()` y `removeNode()` para manipular los nodos secundarios.

## Véase también

[XML.appendChild\(\)](#), [XML.insertBefore\(\)](#), [XML.removeNode\(\)](#)

# XML.nodeName

## Disponibilidad

Flash Player 5.

## Sintaxis

`my_xml.nodeName`

## Descripción

Propiedad; el nombre de nodo del objeto XML. Si el objeto XML es un elemento XML (`nodeType == 1`), `nodeName` es el nombre de la etiqueta que representa el nodo en el archivo XML. Por ejemplo, `TITLE` es el `nodeName` de una etiqueta `TITLE` HTML. Si el objeto XML es un nodo de texto (`nodeType == 3`), el valor de `nodeName` es `null`.

## Véase también

[XML.nodeType](#)

## XML.nodeType

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.nodeType
```

### Descripción

Propiedad (sólo lectura); toma o devuelve un valor `nodeType`, donde 1 es un elemento XML y 3 es un nodo de texto.

### Véase también

[XML.nodeValue](#)

## XML.nodeValue

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.nodeValue
```

### Descripción

Propiedad; devuelve el valor de nodo del objeto XML. Si el objeto XML es un nodo de texto, `nodeType` es 3 y `nodeValue` es el texto del nodo. Si el objeto XML es un elemento XML (el tipo de nodo es 1), `nodeValue` tendrá el valor `null` y será de sólo lectura.

### Véase también

[XML.nodeType](#)

## XML.onData

### Disponibilidad

Flash Player 5

### Sintaxis

```
my_xml.onData = function(src) {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*src* Datos sin procesar, normalmente en formato XML, enviados por el servidor.

### Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando el texto XML se ha descargado completamente del servidor o cuando se produce un error al descargar texto XML de un servidor. Este controlador se invoca antes de que se analice el texto XML y, por lo tanto, puede utilizarse para llamar a una rutina de análisis personalizada en lugar de utilizar el analizador XML de Flash. El método `XML.onData` devuelve el valor `undefined` o bien una cadena que contiene texto XML descargado del servidor. Si el valor devuelto es `undefined`, significa que se ha producido un error al descargar el texto XML del servidor.

De forma predeterminada, el método `XML.onData` invoca `XML.onLoad()`. Puede sustituir el método `XML.onData` por el comportamiento que desee, pero ya no se llamará al método `XML.onLoad()` a menos que lo haga en su implementación de `XML.onData`.

## Ejemplo

En el ejemplo siguiente se muestra el aspecto que tiene el método `onData` de forma predeterminada:

```
XML.prototype.onData = function (src) {  
    if (src == undefined) {  
        this.onLoad(false);  
    } else {  
        this.parseXML(src);  
        this.loaded = true;  
        this.onLoad(true);  
    }  
}
```

El método `XML.onData` puede sustituirse para interceptar el texto XML sin analizarlo.

# XML.onLoad()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
my_xml.onLoad = function (success) {  
    //las sentencias se escriben aquí  
}
```

## Parámetros

*success* Valor booleano que indica si el objeto XML se ha cargado correctamente con una operación `XML.load()` o `XML.sendAndLoad()`.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; invocado por Flash Player cuando se recibe un documento XML del servidor. Si el documento XML se recibe correctamente, el parámetro *success* tiene el valor *true*. Si el documento no se ha recibido o se ha producido un error al recibir la respuesta del servidor, el valor del parámetro *success* es *false*. La implementación predeterminada de este método no está activa. Para sustituir la implementación predeterminada, debe asignar una función que contenga las acciones que desea.

## Ejemplo

En el ejemplo siguiente se crea un archivo SWF sencillo para una aplicación de escaparate de tienda de comercio electrónico sencilla. El método `sendAndLoad()` transmite un elemento XML que contiene el nombre y la contraseña del usuario e instala un controlador `onLoad` para controlar la respuesta del servidor.

```
function myOnLoad(success) {
    if (success){
        if (e.firstChild.nodeName == "LOGINREPLY_xml" &&
            e.firstChild.attributes.status == "OK") {
            gotoAndPlay("loggedIn")
        } else {
            gotoAndStop("loginFailed")
        }
    } else {
        gotoAndStop("connectionFailed")
    }
}
var myLoginReply_xml = new XML();
myLoginReply_xml.onLoad = myOnLoad;
my_xml.sendAndLoad("http://www.samplestore.com/login.cgi",
    myLoginReply_xml);
```

## Véase también

[function](#), [XML.load\(\)](#), [XML.sendAndLoad\(\)](#)

# XML.parentNode

## Disponibilidad

Flash Player 5.

## Sintaxis

*my\_xml*.parentNode

## Descripción

Propiedad (sólo lectura); hace referencia al nodo principal del objeto XML especificado o devuelve el valor *null* si el nodo no tiene nodo principal. Esta es una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios; utilice `appendChild()`, `insertBefore()` y `removeNode()` para manipular dichos nodos.



## XML.parseXML()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.parseXML(source)
```

### Parámetros

*source* Texto XML que debe analizarse y pasarse al objeto XML especificado.

### Valor devuelto

Ninguno.

### Descripción

Método; analiza el texto XML especificado en el parámetro *source* y rellena el objeto XML especificado con el árbol XML resultante. Se descartan cualquiera de los árboles existentes del objeto XML.

## XML.previousSibling

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.previousSibling
```

### Descripción

Propiedad (sólo lectura); devuelve una referencia al nodo colateral anterior de la lista de nodos secundarios del nodo principal. La propiedad tiene el valor `null` si el nodo no tiene un nodo colateral anterior. Se trata de una propiedad de sólo lectura y no puede utilizarse para manipular nodos secundarios; para ello, utilice `XML.appendChild()`, `XML.insertBefore()` y `XML.removeNode()`.

## XML.removeNode()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.removeNode()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; elimina el objeto XML especificado de su principal. Todos los descendientes del nodo también se eliminan.

## XML.send()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.send(url, [window])
```

### Parámetros

*url* URL de destino para el objeto XML especificado.

*window* Ventana del navegador en la que se visualizarán los datos devueltos por el servidor: *\_self* especifica el marco actual en la ventana actual, *\_blank* especifica una ventana nueva, *\_parent* especifica el marco principal del marco actual, y *\_top* especifica el marco de nivel superior de la ventana actual. Este parámetro es opcional. No especificar ningún parámetro *window* equivale a especificar *\_self*.

### Valor devuelto

Ninguno.

### Descripción

Método; codifica el objeto XML especificado en un documento XML y lo envía a la URL especificada mediante el método POST.

## XML.sendAndLoad()

### Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

### Sintaxis

```
my_xml.sendAndLoad(url, targetXMLObject)
```

### Parámetros

*url* URL de destino para el objeto XML especificado. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*targetXMLObject* Objeto XML creado con el método constructor XML que recibirá la información devuelta del servidor.

### Valor devuelto

Ninguno.

## Descripción

Método; codifica el objeto XML especificado en un documento XML, lo envía a la URL especificada utilizando el método `POST`, descarga la respuesta del servidor y después la carga en el *targetXMLObject* especificado en los parámetros. La respuesta del servidor se carga del mismo modo utilizado por el método `load()`.

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, *url* debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, *url* debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables desde archivos SWF que también estén en `www.someDomain.com`. Si desea cargar variables desde un dominio diferente, puede colocar un *archivo de política para distintos dominios* en el servidor que alberga el archivo SWF al que se está accediendo. Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Cuando se ejecuta el método `load()`, la propiedad del objeto XML `loaded` se establece en `false`. Cuando finaliza la descarga de los datos XML, la propiedad `loaded` se establece en `true` y se invoca el método `onLoad()`. Los datos XML no se analizan hasta que no se han descargado por completo. Si el objeto XML contenía anteriormente algún árbol XML, se pasa por alto.

## Véase también

[XML.load\(\)](#)

## XML.status

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.status
```

### Descripción

Propiedad; establece y devuelve automáticamente un valor numérico que indica si se ha analizado correctamente un documento XML en un objeto XML. Se muestra la siguiente lista con los códigos de estado numéricos y una descripción de cada uno:

- 0 Sin errores; el análisis se completó correctamente.
- -2 Una sección de CDATA no se finalizó correctamente.
- -3 La declaración XML no se finalizó correctamente.
- -4 La declaración DOCTYPE no se finalizó correctamente.
- -5 Un comentario no se finalizó correctamente.
- -6 Un elemento XML estaba mal formado.
- -7 Memoria insuficiente.
- -8 Un valor de atributo no se finalizó correctamente.

- -9 Una etiqueta de inicio no coincidía con una etiqueta final.
- -10 Se ha encontrado una etiqueta final sin la etiqueta de inicio correspondiente.

## XML.toString()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.toString()
```

### Parámetros

Ninguno.

### Valor devuelto

Una cadena.

### Descripción

Método; comprueba el valor del objeto XML especificado, construye una representación textual de la estructura XML que incluye el nodo principal, los nodos secundarios y los atributos, y devuelve el resultado en forma de cadena.

Para objetos XML de nivel superior (los creados con el constructor), `XML.toString()` presenta la declaración XML del documento (almacenada en `XML.xmlDecl`), seguida de la declaración `DOCTYPE` del documento (almacenada en `XML.docTypeDecl`), seguida de la representación textual de todos los nodos XML del objeto. Si `XML.xmlDecl` está sin definir, no se realiza la salida de declaración XML. Si `XML.docTypeDecl` está sin definir, no se realiza la salida de la declaración `DOCTYPE`.

### Ejemplo

El código siguiente es un ejemplo de `XML.toString()` que envía `<h1>test</h1>` al panel Salida.

```
node = new XML("<h1>prueba</h1>");  
trace(node.toString());
```

### Véase también

[XML.docTypeDecl](#), [XML.xmlDecl](#)

## XML.xmlDecl

### Disponibilidad

Flash Player 5.

### Sintaxis

```
my_xml.xmlDecl
```

Descripción

Propiedad; especifica información sobre la declaración XML de un documento. Una vez analizado el documento XML en un objeto XML, esta propiedad se establece en el texto de la declaración XML del documento. Esta propiedad se establece utilizando una representación de cadena de la declaración XML, no un objeto de nodo XML. Si no se ha encontrado ninguna declaración XML durante una operación de análisis, la propiedad se establece en `undefined.XML`. El método `toString()` presenta del contenido de `XML.xmlDecl` antes que cualquier otro texto del objeto XML. Si `XML.xmlDecl` contiene el tipo `undefined`, no se genera la salida de ninguna declaración XML.

Ejemplo

En el ejemplo siguiente se utiliza `XML.xmlDecl` para establecer la declaración de documento XML de un objeto XML.

```
my_xml.xmlDecl = "<?xml version=\"1.0\" ?>";
```

A continuación, se muestra un ejemplo de declaración XML:

```
<?xml version="1.0" ?>
```

Véase también

```
XML.docTypeDecl, XML.toString()
```

Clase XMLNode

Disponibilidad

Flash Player 5.

Descripción

La clase XMLNode admite las propiedades, los métodos y las colecciones siguientes; para más información sobre su sintaxis, consulte las entradas de la clase XML correspondientes.

Propiedad, método o colección	Entrada de la clase XML correspondiente
<code>appendChild()</code>	<code>XML.appendChild()</code>
<code>attributes</code>	<code>XML.attributes</code>
<code>childNodes</code>	<code>XML.childNodes</code>
<code>cloneNode()</code>	<code>XML.cloneNode()</code>
<code>firstChild</code>	<code>XML.firstChild</code>
<code>hasChildNodes()</code>	<code>XML.hasChildNodes()</code>
<code>insertBefore()</code>	<code>XML.insertBefore()</code>
<code>lastChild</code>	<code>XML.lastChild</code>
<code>nextSibling</code>	<code>XML.nextSibling</code>
<code>nodeName</code>	<code>XML.nodeName</code>
<code>nodeType</code>	<code>XML.nodeType</code>
<code>nodeValue</code>	<code>XML.nodeValue</code>
<code>parentNode</code>	<code>XML.parentNode</code>

Propiedad, método o colección	Entrada de la clase XML correspondiente
<code>previousSibling</code>	<code>XML.previousSibling</code>
<code>removeNode()</code>	<code>XML.removeNode()</code>
<code>toString()</code>	<code>XML.toString()</code>

**Véase también**

[Clase XML](#)

## Clase XMLSocket

### Disponibilidad

Flash Player 5.

### Descripción

La clase XMLSocket implementa sockets cliente que permiten que el equipo que ejecuta Flash Player se comunice con el equipo servidor, que se identifica mediante una dirección IP o un nombre de dominio. La clase XMLSocket es útil para las aplicaciones cliente-servidor que requieren una latencia baja, como las aplicaciones de chat en tiempo real. Una solución de chat basada en HTTP tradicional consulta frecuentemente al servidor y descarga los nuevos mensajes utilizando una solicitud HTTP. Por el contrario, una solución de chat XMLSocket mantiene una conexión abierta con el servidor, lo que permite a éste enviar inmediatamente los mensajes entrantes sin una solicitud del cliente.

Para utilizar la clase XMLSocket, el equipo servidor debe ejecutar un daemon que comprenda el protocolo utilizado por la clase XMLSocket. El protocolo es el que se muestra a continuación:

- Los mensajes XML se envían por una conexión socket en flujo TCP/IP de dúplex completo.
- Cada mensaje XML es un documento XML completo, que finaliza con un byte cero.
- Se pueden enviar y recibir un número ilimitado de mensajes XML por una sola conexión XMLSocket.

Se aplican las restricciones siguientes al modo y al lugar en que un objeto XMLSocket se puede conectar al servidor:

- El método `XMLSocket.connect()` puede conectarse solamente a los números de puerto TCP 1024 o superiores. Una consecuencia de esta restricción es que los daemons del servidor que se comunican con el objeto XMLSocket también deben estar asignados a números de puerto 1024 o superiores. Los números de puerto por debajo de 1024 los suelen utilizar los servicios del sistema, como FTP, Telnet y HTTP, por lo que los objetos XMLSocket no están autorizados a acceder a ellos por razones de seguridad. La restricción de número de puerto limita la posibilidad de que se acceda y se usen estos recursos de modo inapropiado.
- El método `XMLSocket.connect()` puede conectarse solamente con equipos ubicados en el mismo dominio donde reside el archivo SWF. Esta restricción no se aplica a los archivos SWF que se ejecutan fuera de un disco local. Esta restricción es idéntica a las normas de seguridad de `loadVariables()`, `XML.sendAndLoad()` y `XML.load()`. Para conectarse a un daemon del servidor que se ejecuta en un dominio distinto del dominio en el que reside el archivo SWF, puede crear un archivo de política de seguridad en el servidor que permita el acceso desde dominios específicos. Para más información sobre la creación de políticas para las conexiones de XMLSocket, consulte [“Carga de datos de varios dominios” en la página 198](#).

Puede suponer un reto establecer un servidor para que se comunique con el objeto XMLSocket. Si su aplicación no requiere interactividad en tiempo real, utilice la acción `loadVariables()` o las funciones de conectividad de servidor XML basada en HTTP de Flash (`XML.load()`, `XML.sendAndLoad()`, `XML.send()`), en lugar de la clase XMLSocket.

Para utilizar los métodos de la clase XMLSocket, primero debe utilizar el constructor, `newXMLSocket`, para crear un nuevo objeto XMLSocket.

## Resumen de métodos para la clase XMLSocket

Método	Descripción
<code>XMLSocket.close()</code>	Cierra una conexión de socket abierta.
<code>XMLSocket.connect()</code>	Establece una conexión con el servidor especificado.
<code>XMLSocket.send()</code>	Envía un objeto XML al servidor.

## Resumen de controladores de eventos para la clase XMLSocket

Controlador de eventos	Descripción
<code>XMLSocket.onClose()</code>	Un controlador de eventos que se invoca al cerrarse una conexión XMLSocket.
<code>XMLSocket.onConnect()</code>	Un controlador de eventos que Flash Player invoca cuando una solicitud de conexión iniciada mediante <code>XMLSocket.connect()</code> se ha realizado correctamente o no.
<code>XMLSocket.onData()</code>	Un controlador de eventos que se invoca cuando se ha descargado un mensaje XML del servidor.
<code>XMLSocket.onXML()</code>	Un controlador de eventos que se invoca cuando llega un objeto XML del servidor.

## Constructor para la clase XMLSocket

### Disponibilidad

Flash Player 5.

### Sintaxis

```
new XMLSocket()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Constructor; crea un nuevo objeto XMLSocket. El objeto XMLSocket no está conectado inicialmente con ningún servidor. Debe llamar a `XMLSocket.connect()` para conectar el objeto a un servidor.

## XMLSocket.close()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myXMLSocket.close()
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Método; cierra la conexión especificada por el objeto XMLSocket.

### Véase también

[XMLSocket.connect\(\)](#)

## XMLSocket.connect()

### Disponibilidad

Flash Player 5; comportamiento modificado en Flash Player 7.

### Sintaxis

```
myXMLSocket.connect(host, port)
```

### Parámetros

*host* Nombre de dominio DNS calificado o dirección IP con el formato *aaa.bbb.ccc.ddd*.

También puede especificar el valor `null` para conectarse al servidor host en el que reside el archivo SWF. Si el archivo SWF que emite esta llamada se está ejecutando en un navegador web, *url* debe estar en el mismo dominio que el archivo SWF. Para más información, consulte la descripción que aparece a continuación.

*port* Número de puerto TCP en el host utilizado para establecer una conexión. El número de puerto debe ser 1024 o superior.

### Valor devuelto

Valor booleano.

### Descripción

Método; establece una conexión con el host de Internet especificado utilizando el puerto TCP indicado (debe ser 1024 o superior) y devuelve `true` o `false` dependiendo de si la conexión se ha establecido correctamente. Si no conoce el número de puerto de su equipo de Internet anfitrión, póngase en contacto con su administrador de red.



Si se especifica el valor `null` para el parámetro `host`, el host con el que se establecerá contacto será el host en el que reside el archivo SWF que llama a `XMLSocket.connect()`. Por ejemplo, si el archivo SWF se descargó de `http://www.yoursite.com`, especificar `null` para el parámetro `host` es lo mismo que introducir la dirección IP de `www.yoursite.com`.

En los archivos SWF que se ejecuten en una versión del reproductor anterior a Flash Player 7, `url` debe encontrarse en el mismo superdominio que el archivo SWF que emite esta llamada. Por ejemplo, un archivo SWF de `www.someDomain.com` puede cargar variables desde un archivo SWF de `store.someDomain.com` porque ambos archivos se encuentran en el mismo superdominio que `someDomain.com`.

En los archivos SWF de cualquier versión que se ejecuten en Flash Player 7 o posterior, `url` debe encontrarse exactamente en el mismo dominio (véase [“Funciones de seguridad de Flash Player” en la página 196](#)). Por ejemplo, un archivo SWF de `www.someDomain.com` sólo puede cargar variables de archivos SWF que también se encuentren en `www.someDomain.com`. Si desea cargar variables de un dominio diferente puede situar un *archivo de política para distintos dominios* en el servidor que aloja el archivo SWF al que se accede (debe situarse en el servidor HTTP que se ejecuta en el puerto 80 en el mismo dominio que el servidor de socket). Para más información, consulte [“Carga de datos de varios dominios” en la página 198](#).

Cuando se ejecuta el método `load()`, la propiedad del objeto XML `loaded` se establece en `false`. Cuando finaliza la descarga de los datos XML, la propiedad `loaded` se establece en `true` y se invoca el método `onLoad()`. Los datos XML no se analizan hasta que no se han descargado por completo. Si el objeto XML contenía anteriormente algún árbol XML, se pasa por alto.

Si `XMLSocket.connect()` devuelve el valor `true`, la fase inicial del proceso de conexión se realiza correctamente; más tarde, se invoca el método `XMLSocket.onConnect` para determinar si la conexión final se realizó correctamente o no. Si `XMLSocket.connect()` devuelve el valor `false`, significa que no se ha podido establecer la conexión.

## Ejemplo

En el ejemplo siguiente se utiliza `XMLSocket.connect()` para conectar con el host donde reside el archivo SWF y se utiliza `trace` para mostrar el valor devuelto que indica si la conexión se ha realizado correctamente o ha generado un error.

```
function myOnConnect(success) {
    if (success){
        trace ("Conexión establecida")
    } else {
        trace ("Error de conexión")
    }
}
socket = new XMLSocket()
socket.onConnect = myOnConnect
if (!socket.connect(null, 2000)) {
    trace ("Error de conexión")
}
```

## Véase también

[function, XMLSocket.onConnect\(\)](#)

## XMLSocket.onClose()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myXMLSocket.onClose() = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetros

Ninguno.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; se invoca solamente cuando el servidor cierra una conexión abierta. La implementación predeterminada de este método no realiza acciones. Para sustituir la implementación predeterminada, debe asignar una función que contenga las acciones que desea.

### Véase también

[function](#), [XMLSocket.onConnect\(\)](#)

## XMLSocket.onConnect()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myXMLSocket.onConnect(success)  
    // las sentencias se escriben aquí  
}
```

### Parámetros

*success* Valor booleano que indica si una conexión de socket se ha establecido correctamente (true o false).

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; Flash Player lo invoca cuando una solicitud iniciada mediante [XMLSocket.connect\(\)](#) se ha ejecutado correctamente o ha generado un error. Si la conexión se ha realizado correctamente, el parámetro *success* es true; de lo contrario, el parámetro *success* es false.

La implementación predeterminada de este método no realiza acciones. Para sustituir la implementación predeterminada, debe asignar una función que contenga las acciones que desea.

## Ejemplo

En el ejemplo siguiente se muestra cómo especificar una función de sustitución para el método `onConnect` en una aplicación de chat sencilla.

La función controla a qué pantalla se lleva a los usuarios, dependiendo de si se ha establecido correctamente una conexión. Si la conexión se ha establecido correctamente, se lleva a los usuarios a la pantalla principal de chat en el fotograma con la etiqueta `startChat`. Si la conexión no se establece, los usuarios van a una pantalla con información para solucionar problemas en el fotograma con la etiqueta `connectionFailed`.

```
function myOnConnect(success) {
    if (success) {
        gotoAndPlay("startChat")
    } else {
        gotoAndStop("connectionFailed")
    }
}
```

Después de crear el objeto `XMLSocket` utilizando el método constructor, el script instala el método `onConnect` utilizando el operador de asignación:

```
socket = new XMLSocket();
socket.onConnect = myOnConnect;
```

Finalmente, la conexión se inicia. Si `connect()` devuelve el valor `false`, el archivo SWF se envía directamente al fotograma con la etiqueta `connectionFailed` y `onConnect` no se llega a invocar. Si `connect()` devuelve el valor `true`, el archivo SWF salta al fotograma con la etiqueta `waitForConnection`, que es la pantalla “Espere, por favor”. El archivo SWF permanece en el fotograma `waitForConnection` hasta que se invoca el controlador `onConnect` (la velocidad a la que se invoca depende de la latencia de la red).

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed")
} else {
    gotoAndStop("waitForConnection")
}
```

## Véase también

[function, XMLSocket.connect\(\)](#)

# XMLSocket.onData()

## Disponibilidad

Flash Player 5.

## Sintaxis

```
XMLSocket.onData = function(src) {
    // las sentencias se escriben aquí
}
```

## Parámetros

*src* Cadena que contiene los datos enviados por el servidor.

## Valor devuelto

Ninguno.

## Descripción

Controlador de eventos; se invoca cuando se ha descargado un mensaje del servidor, terminado en un byte cero. Puede ignorar `XMLSocket.onData` para interceptar los datos enviados por el servidor sin analizarlos como XML. Esto resulta de utilidad si se transmiten arbitrariamente paquetes de datos con formato y preferiría manipular los datos directamente cuando lleguen en lugar de que Flash Player los analice como XML.

De forma predeterminada, el método `XMLSocket.onData` invoca el método `XMLSocket.onXML`. Si sustituye el método `XMLSocket.onData` por el comportamiento que desea, ya no se llamará al método `XMLSocket.onXML` a menos que lo haga en su implementación de `XMLSocket.onData`.

```
XMLSocket.prototype.onData = function (src) {  
    this.onXML(new XML(src));  
}
```

En el ejemplo anterior, el parámetro *src* es una cadena que contiene el texto XML descargado del servidor. El byte cero de terminación no se incluye en la cadena.

## XMLSocket.onXML()

### Disponibilidad

Flash Player 5.

### Sintaxis

```
myXMLSocket.onXML(object) = function() {  
    // las sentencias se escriben aquí  
}
```

### Parámetro

*object* Objeto XML que contiene un documento XML analizado que se ha recibido de un servidor.

### Valor devuelto

Ninguno.

### Descripción

Controlador de eventos; Flash Player lo invoca cuando el objeto XML especificado que contiene un documento XML llega por una conexión `XMLSocket` abierta. Una conexión `XMLSocket` puede utilizarse para transferir un número ilimitado de documentos XML entre el cliente y el servidor. Cada documento termina en un byte 0 (cero). Cuando Flash Player recibe el byte 0, analiza todos los XML recibidos desde el byte cero anterior o desde que se estableció la conexión si éste es el primer mensaje que se recibe. Cada lote de XML analizado se trata como un solo documento XML y se pasa al método `onXML`.

La implementación predeterminada de este método no realiza acciones. Para sustituir la implementación predeterminada, debe asignar una función que contenga las acciones que desea.

### Ejemplo

La función siguiente sustituye la implementación predeterminada del método `onXML` en una aplicación de chat sencilla. La función `myOnXML` da instrucciones a la aplicación de chat para que reconozca un solo elemento XML, `MESSAGE`, con el formato siguiente:

```
<MESSAGE USER="Juan" TEXT="Hola, me llamo Juan." />.
```

El controlador `onXML` debe instalarse primero en el objeto `XMLSocket` como se muestra a continuación:

```
socket.onXML = myOnXML;
```

Se presupone que la función `displayMessage()` es una función definida por el usuario que muestra el mensaje recibido por el usuario.

```
function myOnXML(doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MENSAJE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

**Véase también**

[function](#)

## XMLSocket.send()

**Disponibilidad**

Flash Player 5.

**Sintaxis**

```
myXMLSocket.send(object)
```

**Parámetros**

*object* Objeto XML u otros datos que se van a transmitir al servidor.

**Valor devuelto**

Ninguno.

**Descripción**

Método; convierte el objeto XML o los datos especificados en el parámetro *object* en una cadena y la transmite al servidor, seguida de un byte cero. Si *object* es un objeto XML, la cadena es la representación textual XML del objeto XML. La operación de envío es asíncrona, vuelve inmediatamente, pero los datos pueden transmitirse más tarde. El método `XMLSocket.send()` no devuelve un valor que indica si los datos se han transmitido correctamente.

Si el objeto *myXMLSocket* no está conectado al servidor (utilizando [XMLSocket.connect\(\)](#)), la operación `XMLSocket.send()` generará un error.

**Ejemplo**

En el ejemplo siguiente se muestra como podría especificar un nombre de usuario y una contraseña para enviar el objeto XML `my_xml` al servidor:

```
var my_xml = new XML();  
var myLogin = my_xml.createElement("login");  
myLogin.attributes.username = usernameTextField;  
myLogin.attributes.password = passwordTextField;  
my_xml.appendChild(myLogin);  
myXMLSocket.send(my_xml);
```

**Véase también**

[XMLSocket.connect\(\)](#)



# APÉNDICE A

## Mensajes de error

Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004 ofrecen una función mejorada para crear informes de error durante la compilación si se especifica ActionScript 2.0 (valor predeterminado) al publicar el archivo. La tabla siguiente contiene una lista de mensajes de error que el compilador de Flash puede generar.

Número de error	Texto del mensaje
1093	Se esperaba un nombre de clase.
1094	Se espera un nombre de clase base después de la palabra clave 'extends'.
1095	Se ha utilizado un atributo de miembro incorrectamente.
1096	No se puede repetir el mismo nombre de miembro más de una vez.
1097	Todas las funciones de miembro de clase requieren un nombre.
1099	Las definiciones de clase no admiten este tipo de declaración.
1100	Ya se ha definido una clase o interfaz con este nombre.
1101	El tipo no coincide.
1102	No hay ninguna clase que lleve por nombre '«NombreClase»'.
1103	No hay ninguna propiedad que lleve por nombre '«nombrePropiedad»'.
1104	Se ha intentado llamar a una función cuando no se trataba de una función.
1105	El tipo de la declaración de asignación no coincide. Se ha encontrado [lhs-type] donde es necesario [rhs-type].
1106	El miembro de clase es privado y no permite el acceso.
1107	Las interfaces no admiten declaraciones de variables.
1108	Las interfaces no admiten declaraciones de eventos.
1109	Las interfaces no admiten declaraciones de captador/definidor.
1110	Las interfaces no admiten miembros de clase privados.
1111	Las interfaces no admiten cuerpos de función.
1112	Una clase no se puede ampliar.

Número de error	Texto del mensaje
1113	Una interfaz no se puede ampliar.
1114	No hay ninguna interfaz definida con este nombre.
1115	Una clase no puede ampliar una interfaz.
1116	Una interfaz no puede ampliar una clase.
1117	Se espera un nombre de interfaz después de la palabra clave 'implements'.
1118	Una clase no puede implementar otra clase, sólo interfaces.
1119	La clase debe implementar el método 'nombreMétodo' de la interfaz 'nombreInterfaz'.
1120	La implementación de un método de interfaz tiene que ser un método, no una propiedad.
1121	Una clase no puede ampliar la misma interfaz más de una vez.
1122	La implementación del método de interfaz no coincide con su definición.
1123	Esta construcción sólo está disponible en ActionScript 1.0.
1124	Esta construcción sólo está disponible en ActionScript 2.0.
1125	Las interfaces no admiten miembros de clase estáticos.
1126	La expresión devuelta tiene que coincidir con el tipo de devolución de la función.
1127	Esta función requiere una acción 'return'.
1128	Atributo utilizado fuera de una clase.
1129	Las funciones cuyo tipo de devolución sea Void no podrán devolver un valor.
1130	La cláusula 'extends' debe aparecer antes de la cláusula 'implements'.
1131	Después de ':' se espera un identificador de tipo.
1132	Las interfaces deben utilizar la palabra clave 'extends' y no 'implements'.
1133	Una clase no podrá ampliar más de una clase.
1134	Una interfaz no podrá ampliar más de una interfaz.
1135	No hay ningún método que lleve por nombre '<nombreMétodo>'.
1136	Las definiciones de interfaz no admiten este tipo de declaración.
1137	Las funciones set requieren exactamente un parámetro.
1138	Las funciones get no requieren ningún parámetro.
1139	Sólo se pueden definir clases en scripts de clase ActionScript 2.0 externos.
1140	Los scripts de clase ActionScript 2.0 sólo pueden definir construcciones de clase o interfaz.
1141	Existe un conflicto entre el nombre de esta clase, '<A.B.C>', y el nombre de otra clase que se ha cargado, '<A.B>'.
1142	No se ha podido cargar la clase '<NombreClase>'.



Número de error	Texto del mensaje
1143	Las interfaces sólo pueden definirse en scripts de clase ActionScript 2.0 externos.
1144	No se puede acceder a variables de instancia en funciones estáticas.
1145	No se pueden anidar las definiciones de clases e interfaces.
1146	La propiedad a la que se hace referencia no cuenta con el atributo estático.
1147	Esta llamada a un operador de superclase no coincide con el superconstructor.
1148	En los métodos de interfaz sólo se admite el atributo público.
1149	No se puede utilizar la palabra clave 'import' como directiva.
1150	Para poder utilizar esta acción, deberá exportar su película como Flash 7.
1151	Para poder utilizar esta expresión, deberá exportar su película como Flash 7.
1152	Esta cláusula de excepción no está colocada correctamente.
1153	Una clase sólo puede tener un constructor.
1154	Los constructores no pueden devolver un valor.
1155	Los constructores no pueden especificar un tipo de devolución.
1156	Las variables no pueden ser del tipo Void.
1157	Los parámetros de función no pueden ser del tipo Void.
1158	Sólo puede accederse directamente a los miembros estáticos a través de las clases.
1159	Varias de las interfaces implementadas contienen el mismo método con tipos diferentes.
1160	Ya existe una clase o interfaz definida con ese nombre.
1161	No es posible borrar clases, interfaces o tipos integrados.
1162	No hay ninguna clase que lleve este nombre.
1163	La palabra clave 'palabraclave' está reservada para ActionScript 2.0 y, por lo tanto, no se podrá utilizar aquí.
1164	No se ha concluido la definición del atributo personalizado.
1165	Sólo se podrá definir una clase o interfaz por cada archivo .as de ActionScript 2.0.
1166	La clase que se está compilando, 'A.b', no coincide con la clase que se importó, 'A.B'.
1167	Debe especificar un nombre de clase.
1168	El nombre de clase que ha introducido contiene un error de sintaxis.
1169	El nombre de interfaz que ha introducido contiene un error de sintaxis.
1170	El nombre de clase base que ha introducido contiene un error de sintaxis.
1171	El nombre de interfaz base que ha introducido contiene un error de sintaxis.

Número de error	Texto del mensaje
1172	Tiene que escribir un nombre de interfaz.
1173	Tiene que escribir un nombre de clase o de interfaz.
1174	El nombre de clase o de interfaz que ha introducido incluye un error de sintaxis.
1175	No se puede acceder a 'variable' desde este ámbito.
1176	El atributo 'get/set/private/public/static' aparece en varias ocasiones.
1177	Se ha utilizado un atributo de clase incorrectamente.
1178	No se pueden utilizar variables de instancia y funciones para inicializar variables estáticas.
1179	Se han descubierto circularidades de tiempo de ejecución entre las siguientes clases:%1
1180	El Flash Player que se desea utilizar como destino no admite depuración.
1181	El Flash Player que se desea utilizar como destino no admite el evento releaseOutside.
1182	El Flash Player que se desea utilizar como destino no admite el evento dragOver.
1183	El Flash Player que se desea utilizar como destino no admite el evento dragOver.
1184	El Flash Player que se desea utilizar como destino no admite acciones de arrastre.
1185	El Flash Player que se desea utilizar como destino no admite la acción loadMovie.
1186	El Flash Player que se desea utilizar como destino no admite la acción getURL.
1187	El Flash Player que se desea utilizar como destino no admite la acción FSCommand.
1188	No se admiten declaraciones de importación en definiciones de clase o interfaz.
1189	No se puede importar la clase 'A.B' porque su nombre de hoja ya está dirigido a la clase que se está definiendo, 'C.B'.
1190	No se puede importar la clase 'A.B' porque su nombre de hoja ya está dirigido a la clase importada 'C.B'.
1191	Sólo se pueden inicializar las variables de instancia de una clase para compilar/cronometrar expresiones de constantes.
1192	Las funciones de miembro de clase no pueden llamarse igual que una función de constructor de una superclase.
1193	Existe un conflicto entre el nombre de esta clase, 'NombreClase', y el nombre de otra clase que se ha cargado.
1194	Hay que llamar primero al superconstructor en el cuerpo del constructor.

Número de error	Texto del mensaje
1195	El identificador '‹nombreClase›' no se dirigirá al objeto integrado '‹NombreClase›' en tiempo de ejecución.
1196	Hay que definir la clase '‹A.B.NombreClase›' en un archivo cuya ruta relativa sea '‹A.B›'.
1197	En el nombre de clase '‹NombreClase›' no se utiliza el carácter comodín '*' correctamente.
1198	Las mayúsculas/minúsculas de la función de miembro '‹nombreclase›' no coinciden con las del nombre de clase que se está definiendo, '‹NombreClase›', y no se interpretarán como constructor de clase en tiempo de ejecución.
1199	El único tipo permitido para repetidores de reproducción indefinida for-in es Cadena.
1200	Las funciones de definidor no pueden devolver un valor.
1201	Los únicos atributos permitidos para las funciones constructoras son public y private.



# APÉNDICE B

## Precedencia y asociatividad de los operadores

En esta tabla se enumeran todos los operadores de ActionScript y su asociatividad, ordenados de la precedencia más alta a la más baja.

Operador	Descripción	Asociatividad
<b>Precedencia más alta</b>		
+	Más unario	De derecha a izquierda
-	Menos unario	De derecha a izquierda
~	NOT en modo bit	De derecha a izquierda
!	NOT lógico	De derecha a izquierda
not	NOT lógico (estilo de Flash 4)	De derecha a izquierda
++	Incremento posterior	De izquierda a derecha
--	Decremento posterior	De izquierda a derecha
()	Llamada a función	De izquierda a derecha
[]	Elemento de matriz	De izquierda a derecha
.	Miembro de una estructura	De izquierda a derecha
++	Incremento previo	De derecha a izquierda
--	Decremento previo	De derecha a izquierda
new	Asignar objeto	De derecha a izquierda
delete	Anular la asignación de objeto	De derecha a izquierda
typeof	Tipo de objeto	De derecha a izquierda
void	Devuelve un valor no definido	De derecha a izquierda
*	Multiplicar	De izquierda a derecha
/	Dividir	De izquierda a derecha
%	Módulo	De izquierda a derecha
+	Sumar	De izquierda a derecha

Operador	Descripción	Asociatividad
add	Concatenación de cadenas (anteriormente &)	De izquierda a derecha
-	Restar	De izquierda a derecha
<<	Desplazamiento a la izquierda en modo bit	De izquierda a derecha
>>	Desplazamiento a la derecha en modo bit	De izquierda a derecha
>>>	Desplazamiento a la derecha en modo bit (sin signo)	De izquierda a derecha
<	Menor que	De izquierda a derecha
<=	Menor o igual que	De izquierda a derecha
>	Mayor que	De izquierda a derecha
>=	Mayor o igual que	De izquierda a derecha
instanceof	Instancia de	De izquierda a derecha
lt	Menor que (versión para cadenas)	De izquierda a derecha
le	Menor o igual que (versión para cadenas)	De izquierda a derecha
gt	Mayor que (versión para cadenas)	De izquierda a derecha
ge	Mayor o igual que (versión para cadenas)	De izquierda a derecha
==	Igual	De izquierda a derecha
!=	Distinto	De izquierda a derecha
eq	Igual (versión para cadenas)	De izquierda a derecha
ne	Distinto (versión para cadenas)	De izquierda a derecha
&	AND en modo bit	De izquierda a derecha
^	XOR en modo bit	De izquierda a derecha
	OR en modo bit	De izquierda a derecha
&&	AND lógico	De izquierda a derecha
and	AND lógico (estilo de Flash 4)	De izquierda a derecha
	OR lógico	De izquierda a derecha
or	OR lógico (estilo de Flash 4)	De izquierda a derecha
?:	Condicional	De derecha a izquierda
=	Asignación	De derecha a izquierda
*=, /=, %=, +=, -=, &=,  =, ^=, <<=, >>=, >>>=	Asignación compuesta	De derecha a izquierda
,	Coma	De izquierda a derecha
<b>Precedencia más baja</b>		

# APÉNDICE C

## Teclas del teclado y valores de códigos de tecla

En las tablas siguientes se muestran todas las teclas de un teclado estándar y los valores correspondientes del código de tecla ASCII utilizados para identificar las teclas en ActionScript. Para más información, consulte la entrada [Clase Key](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la página 213.

### Letras de la A a la Z y números estándar del 0 al 9

En la tabla siguiente se enumeran las teclas de un teclado estándar para las letras de la A a la Z y los números del 0 al 9, con los valores correspondientes del código de tecla ASCII usados para identificar las teclas en ActionScript.

Tecla de letra o número	Código de tecla
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80

Tecla de letra o número	Código de tecla
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

## Teclas del teclado numérico

En la tabla siguiente se enumeran las teclas de un teclado numérico con los valores correspondientes del código de tecla ASCII usados para identificar las teclas en ActionScript.

Tecla del teclado numérico	Código de tecla
0 del teclado numérico	96
1 del teclado numérico	97
2 del teclado numérico	98
3 del teclado numérico	99
4 del teclado numérico	100
5 del teclado numérico	101
6 del teclado numérico	102
7 del teclado numérico	103



Tecla del teclado numérico	Código de tecla
8 del teclado numérico	104
9 del teclado numérico	105
Multiplicar	106
Sumar	107
Intro	108
Restar	109
Decimal	110
Dividir	111

## Teclas de función

En la tabla siguiente se enumeran las teclas de función de un teclado estándar con los valores correspondientes del código de tecla ASCII usados para identificar las teclas en ActionScript.

Tecla de función	Código de tecla
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
F13	124
F14	125
F15	126

## Otras teclas

En la tabla siguiente se enumeran las teclas de un teclado estándar que no son letras, números, teclas del teclado numérico ni teclas de función, con los valores correspondientes del código de tecla ASCII usados para identificar las teclas en ActionScript.

Tecla	Código de tecla
Retroceso	8
Tabulador	9
Borrar	12
Intro	13
Mayús	16
Control	17
Alt	18
Bloq Mayús	20
Esc	27
Barra espaciadora	32
Re Pág	33
Av Pág	34
Fin	35
Inicio	36
Flecha izquierda	37
Flecha arriba	38
Flecha derecha	39
Flecha abajo	40
Insert	45
Supr	46
Ayuda	47
Bloq Num	144
::	186
= +	187
- _	189
/ ?	191
` ~	192
[ {	219
\	220

Tecla	Código de tecla
}}	221
" '	222



## APÉNDICE D

# Escritura de scripts para versiones anteriores de Flash Player

ActionScript ha cambiado considerablemente con la versión Macromedia Flash MX 2004 y Macromedia Flash MX Professional 2004. Al crear contenido para Flash Player 7, se beneficiará de todas las prestaciones de ActionScript. Puede utilizar Flash MX 2004 para crear contenido para versiones anteriores de Flash Player, pero no podrá utilizar todos los elementos de ActionScript.

En este capítulo se proporcionan las directrices que le ayudarán a escribir scripts con una sintaxis correcta para la versión de reproductor que desea utilizar.

### Utilización de versiones anteriores de Flash Player

Cuando escriba los scripts, utilice la información de disponibilidad para cada elemento del diccionario de ActionScript (véase el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#)) para determinar si el elemento que desea usar es compatible con la versión de Flash Player que desea utilizar. También hallará información sobre los elementos que puede utilizar en la caja de herramientas Acciones; los elementos no compatibles con la versión que desea utilizar aparecerán resaltados en amarillo.

Si crea contenido para Flash Player 6 o Flash Player 7, debe utilizar ActionScript 2.0, que ofrece una serie de funciones importantes que no se encuentran disponibles en ActionScript 1, como por ejemplo una gestión de errores de compilador mejorada y capacidades de programación orientada a objetos más fiables.

Para conocer más a fondo las diferencias de implementación de ciertas funciones al publicar archivos para Flash Player 7 o para versiones anteriores del reproductor, consulte [“Transferencia de scripts existentes a Flash Player 7” en la página 17](#).

Para especificar el reproductor y la versión de ActionScript que desea utilizar al publicar un documento, seleccione Archivo > Configuración de publicación y, a continuación, elija las opciones correspondientes en la ficha Flash. Si necesita usar Flash Player 4, consulte la siguiente sección.

### Utilización de Flash MX 2004 para crear contenido para Flash Player 4

Si desea utilizar Flash MX 2004 para crear contenido para Flash Player 4, especifique Flash Player 4 en la ficha Flash del cuadro de diálogo Configuración de publicación (Archivo > Configuración de publicación).

ActionScript de Flash Player 4 solamente tiene un tipo de datos primitivo básico que se utiliza tanto para la manipulación numérica como para la manipulación de cadenas. Al crear una aplicación para Flash Player 4, debe utilizar los operadores de cadena desfasados que se encuentran en la categoría Eliminado en nuevas versiones > Operadores de la caja de herramientas Acciones.

Puede utilizar las funciones siguientes de Flash MX 2004 al publicar para Flash Player 4:

- El operador de acceso a matriz y objeto (`[]`)
- El operador de punto (`.`)
- Los operadores lógicos, los operadores de asignación y los operadores de incremento previo e incremento/decremento posterior
- El operador de módulo (`%`) y todos los métodos y propiedades de la clase `Math`

Originariamente, Flash Player 4 no es compatible con los siguientes elementos de lenguaje. Flash MX 2004 los exporta como aproximaciones de series, lo que da lugar a resultados con menor precisión numérica. Además, debido a la inclusión de aproximaciones de series en el archivo SWF, estos elementos ocupan más espacio en los archivos SWF de Flash Player 4 que en los archivos SWF de Flash Player 5 o posterior.

- Las acciones `for`, `while`, `do..while`, `break` y `continue`
- Las acciones `print()` y `printAsBitmap()`
- La acción `switch`

Para más información, consulte [“Utilización de versiones anteriores de Flash Player” en la página 797](#).

## Utilización de Flash MX 2004 para abrir archivos de Flash 4

ActionScript de Flash 4 sólo tenía un tipo de datos verdadero: `string`. Utilizaba diferentes tipos de operadores en expresiones para indicar si el valor debería ser tratado como una cadena o como un número. En las versiones posteriores de Flash, puede utilizar un conjunto de operadores para todos los tipos de datos.

Cuando se utiliza Flash 5 o una versión posterior para abrir un archivo creado en Flash 4, Flash convierte automáticamente las expresiones de ActionScript para hacerlas compatibles con la nueva sintaxis. En el código de ActionScript, verá las conversiones de los siguientes tipos de datos y operadores:

- El operador `=` en Flash 4 se utilizó para la igualdad numérica. En Flash 5 y versiones posteriores, `==` es el operador de igualdad y `=` es el operador de asignación. Cualquiera de los operadores `=` en los archivos de Flash 4 se convierte automáticamente en `==`.
- Flash realiza automáticamente conversiones de tipo para garantizar que los operadores se comportan del modo esperado. Debido a la introducción de múltiples tipos de datos, los siguientes operadores tienen nuevos significados:

`+`, `==`, `!=`, `<>`, `<`, `>`, `>=`, `<=`

En ActionScript de Flash 4, estos operadores siempre eran operadores numéricos. En Flash 5 y versiones posteriores, se comportan de manera diferente según el tipo de datos de los operandos. Para evitar cualquier diferencia semántica en los archivos importados, la función `Number()` se inserta alrededor de todos los operandos de estos operadores. Los números constantes son números obvios, de modo que no se incluyen en `Number()`.

- En Flash 4, la secuencia de escape `\n` generaba un carácter de retorno de carro (ASCII 13). En Flash 5 y versiones posteriores, para cumplir el estándar ECMA-262, `\n` genera un carácter de avance de línea (ASCII 10). Una secuencia `\n` en los archivos FLA de Flash 4 se convierte automáticamente en `\r`.
- El operador `&` en Flash 4 se utilizaba para la suma de cadenas. En Flash 5 y versiones posteriores, `&` es el operador AND en modo bit. El operador de suma de cadenas ahora se denomina `add`. Cualquiera de los operadores `&` en los archivos de Flash 4 se convierten automáticamente en operadores `add`.
- Muchas de las funciones de Flash 4 no necesitan paréntesis de cierre, por ejemplo, `Get Timer`, `Set Variable`, `Stop y Play`. Para crear una sintaxis coherente, la función `getTimer` y todas las acciones requieren ahora paréntesis de cierre. Estos paréntesis se agregan automáticamente durante la conversión.
- En Flash 5 y versiones posteriores, cuando se ejecuta la función `getProperty` en un clip de película que no existe, devuelve el valor `undefined`, no 0. La sentencia `undefined == 0` es `false` en ActionScript en versiones posteriores a Flash 4 (en Flash 4, `undefined == 1`). En Flash 5 y versiones posteriores, este problema se soluciona al convertir los archivos de Flash 4 introduciendo las funciones `Number()` en las comparaciones de igualdad. En el siguiente ejemplo, `Number()` hace que `undefined` se convierta en 0 para que la comparación sea efectiva:  

```
getProperty("clip", _width) == 0
Number(getProperty("clip", _width)) == Number(0)
```

**Nota:** si ha utilizado palabras clave de Flash 5 o de versiones anteriores como nombres de variable en ActionScript de Flash 4, la sintaxis devolverá un error al realizar la compilación en Flash MX 2004. Para solucionar este problema, debe cambiar el nombre de sus variables en todas las ubicaciones. Véase [“Palabras clave” en la página 36](#) y [“Asignación de un nombre a una variable” en la página 43](#).

## Utilización de sintaxis con barras

La sintaxis con barras se utilizó en Flash 3 y 4 para indicar la ruta de destino de un clip de película o de una variable. En ella se utilizan barras en lugar de puntos; además, para indicar una variable, deben utilizarse dos puntos antes del nombre de la misma:

```
myMovieClip/childMovieClip:myVariable
```

Para escribir la misma ruta de destino en la sintaxis con punto (véase [“Sintaxis con punto” en la página 32](#)), admitida por Flash 5 y versiones posteriores, debe utilizar el código siguiente:

```
myMovieClip.childMovieClip.myVariable
```

La sintaxis con barra inversa se utilizaba comúnmente con la acción `tellTarget`, pero su utilización ya no se recomienda. Ahora es recomendable emplear la acción `with` en lugar de `tellTarget` debido a que es más compatible con la sintaxis con punto. Para más información, consulte `tellTarget` y `with` en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).





# APÉNDICE E

## Programación orientada a objetos con ActionScript 1

La información de este apéndice se ha extraído de la documentación de Macromedia Flash MX y ofrece información sobre la utilización del modelo de objetos ActionScript 1 para crear scripts. Se incluye aquí por las siguientes razones:

- Si desea escribir scripts orientados a objetos compatibles con Flash Player 5, debe utilizar ActionScript 1.
- Si ya utiliza ActionScript 1 para escribir scripts orientados a objetos y no está listo para cambiar a ActionScript 2.0, puede utilizar este apéndice para encontrar o revisar la información necesaria para escribir scripts ActionScript 1.

Si nunca ha utilizado ActionScript para escribir scripts orientados a objetos y no necesita utilizar Flash Player 5, no debe utilizar la información de este apéndice, ya que la escritura de scripts orientados a objetos con ActionScript 1 está desfasada; en su lugar, consulte el [Capítulo 9, “Creación de clases con ActionScript 2.0”](#), en la [página 161](#) para más información sobre la utilización de ActionScript 2.0.

**Nota:** en algunos ejemplos de este apéndice se utiliza el método `Object.registerClass()`. Este método sólo funciona con Flash Player 6 y versiones posteriores; no utilice este método si va a utilizar Flash Player 5.

### ActionScript 1

ActionScript es un lenguaje de programación orientado a objetos. La programación orientada a objetos utiliza *objetos*, o estructuras de datos, para agrupar propiedades y métodos que controlan el comportamiento o el aspecto del objeto. Los objetos permiten organizar y reutilizar el código. Después de definir un objeto, puede referirse al objeto por el nombre sin tener que redefinirlo cada vez que lo utiliza.

Una *clase* es una categoría genérica de objetos. Una clase define una serie de objetos que tienen propiedades comunes y pueden controlarse de la misma forma. Las propiedades son atributos que definen un objeto, como su tamaño, posición, color, transparencia, etc. Las propiedades se definen para una clase y los valores de las propiedades se definen para objetos individuales de la clase. Los métodos son funciones que pueden establecer o recuperar propiedades de un objeto. Por ejemplo, puede definir un método para calcular el tamaño de un objeto. Al igual que las propiedades, los métodos se definen para una clase de objeto y se invocan para objetos individuales de la clase.

ActionScript incluye varias clases incorporadas, como la clase `MovieClip`, entre otras. También puede crear clases para definir categorías de objetos para las aplicaciones.

Los objetos en ActionScript pueden ser meros contenedores de datos, o estar representados gráficamente en el escenario como clips de película, botones o campos de texto. Todos los clips de película son instancias de la clase incorporada `MovieClip`, y todos los botones son instancias de la clase incorporada `Button`. Cada instancia de clip de película contiene todas las propiedades (por ejemplo, `_height`, `_rotation`, `_totalframes`) y todos los métodos (por ejemplo, `gotoAndPlay()`, `loadMovie()` y `startDrag()`) de la clase `MovieClip`.

Para definir una clase, debe crear una función especial denominada *función constructora* (constructor). Las clases incorporadas tienen funciones constructoras incorporadas. Por ejemplo, si desea información sobre un ciclista que aparece en la aplicación, puede crear una función constructora, `Biker()`, con las propiedades `time` y `distance` y el método `getSpeed()`, que indique la velocidad a la que se desplaza el ciclista:

```
function Biker(t, d) {  
    this.time = t;  
    this.distance = d;  
    this.getSpeed = function() {return this.time / this.distance;};  
}
```

En este ejemplo, se crea una función que necesita dos elementos de información, o parámetros, para realizar su tarea: `t` y `d`. Al llamar a la función para crear nuevas instancias del objeto, debe pasar los parámetros a dicha función. El código siguiente crea instancias del objeto `Biker` denominadas `emma` y `hamish`.

```
emma = new Biker(30, 5);  
hamish = new Biker(40, 5);
```

En la creación de scripts orientados a objetos, las clases pueden recibir propiedades y métodos de ellas mismas, de acuerdo a un orden determinado; esta función se denomina *herencia*. La herencia puede utilizarse para ampliar o redefinir las propiedades y métodos de una clase. Una clase que hereda propiedades y métodos de otra clase recibe el nombre de *subclase*. Una clase que pasa propiedades y métodos a otra clase recibe el nombre de *superclase*. Una clase puede ser a la vez subclase y superclase.

Un objeto es un tipo de datos complejo que contiene cero o más propiedades y métodos. Cada propiedad tiene un nombre y un valor, como los tiene una variable. Las propiedades están asociadas al objeto y contienen los valores que pueden cambiarse y recuperarse. Estos valores pueden ser de cualquier tipo de datos: cadena, número, booleano, objeto, clip de película o no definido. Las siguientes propiedades tienen varios tipos de datos:

```
customer.name = "Jane Doe";  
customer.age = 30;  
customer.member = true;  
customer.account.currentRecord = 000609;  
customer.mcInstanceName._visible = true;
```

La propiedad de un objeto también puede ser un objeto. En la línea 4 del ejemplo anterior, `account` es una propiedad del objeto `customer` y `currentRecord` es una propiedad del objeto `account`. El tipo de datos de la propiedad `currentRecord` es numérico.

## Creación de un objeto personalizado en ActionScript 1

Para crear un objeto personalizado, debe definir una función constructora. A una función constructora siempre se le asigna el mismo nombre que al tipo de objeto que crea. Puede utilizar la palabra clave `this` en el cuerpo de la función constructora para hacer referencia al objeto que crea el constructor; al llamar a una función constructora, Flash pasa la palabra clave `this` como parámetro oculto. La función constructora que se muestra a continuación, por ejemplo, crea un círculo con la propiedad `radius`:

```
function Circle(radius) {  
    this.radius = radius;  
}
```

Tras definir la función constructora, debe crear una instancia del objeto. Anteponga el operador `new` al nombre de la función constructora y asigne un nombre de variable a la nueva instancia. En el código siguiente, por ejemplo, se utiliza el operador `new` para crear un objeto `Circle` con un radio de 5 y se asigna a la variable `myCircle`:

```
myCircle = new Circle(5);
```

**Nota:** un objeto tiene el mismo ámbito que la variable a la que se ha asignado.

## Asignación de métodos a un objeto personalizado en ActionScript 1

Los métodos de un objeto pueden definirse en la función constructora del objeto. No obstante, no se recomienda utilizar esta técnica porque define el método cada vez que se utiliza la función constructora, como en el ejemplo siguiente, que crea los métodos `area()` y `diameter()`:

```
function Circle(radius) {  
    this.radius = radius;  
    this.area = Math.PI * radius * radius;  
    this.diameter = function() {return 2 * this.radius;}  
}
```

Cada función constructora tiene una propiedad `prototype` que se crea automáticamente al definir la función. La propiedad `prototype` indica los valores predeterminados de la propiedad para los objetos creados con esa función. Cada nueva instancia de un objeto tiene una propiedad `__proto__` que hace referencia a la propiedad `prototype` de la función constructora que la ha creado. Por consiguiente, si se asignan métodos a la propiedad `prototype` de un objeto, dichos métodos sólo estarán disponibles para las instancias nuevas creadas del objeto. Lo mejor es asignar un método a la propiedad `prototype` de la función constructora, ya que de este modo sólo existe en un lugar y las nuevas instancias del objeto (o clase) hacen referencia al mismo. Pueden utilizarse las propiedades `prototype` y `__proto__` para ampliar los objetos de modo que pueda reutilizarse el código de forma orientada a los objetos. Para más información, consulte [“Creación de herencia en ActionScript 1” en la página 805](#).

En el procedimiento siguiente se muestra cómo asignar un método `area()` a un objeto `Circle` personalizado.

**Para asignar un método a un objeto personalizado:**

- 1 Defina la función constructora `Circle()`, como se detalla a continuación.

```
function Circle(radius) {  
    this.radius = radius;  
}
```

- 2 Defina el método `area()` del objeto `Circle`. El método `area()` calcula el área del círculo. Puede utilizar un literal de función para definir el método `area()` y asignar la propiedad `area` al objeto prototipo del círculo, como se detalla a continuación:

```
Circle.prototype.area = function () {  
    return Math.PI * this.radius * this.radius;  
};
```

- 3 Cree una instancia del objeto `Circle`, como se detalla a continuación:

```
var myCircle = new Circle(4);
```

- 4 Llame al método `area()` del nuevo objeto `myCircle`, como se muestra a continuación:

```
var myCircleArea = myCircle.area();
```

ActionScript busca el método `area()` en el objeto `myCircle`. Como el objeto no tiene un método `area()`, se busca el método `area()` en su objeto prototipo `Circle.prototype`. ActionScript lo encuentra y lo llama.

## Definición de métodos de controlador de eventos en ActionScript 1

Puede crear una clase de ActionScript para clips de película y definir los métodos de controlador de eventos del objeto prototipo de esta nueva clase. La definición de los métodos del objeto prototipo hace que todas las instancias de este símbolo respondan a estos eventos del mismo modo.

También puede agregar una acción de controlador de eventos `onClipEvent()` u `on()` a una sola instancia para proporcionar instrucciones exclusivas que sólo se ejecutarán cuando se produzca el evento de dicha instancia. Las acciones `onClipEvent()` y `on()` no prevalecen sobre el método de controlador de eventos; ambos eventos hacen que se ejecuten sus scripts. Sin embargo, si define los métodos de controlador de eventos del objeto prototipo y también define un método de controlador de eventos para una instancia determinada, la definición de la instancia prevalece sobre la definición del objeto prototipo.

### Para definir un método de controlador de eventos en un objeto prototipo de un objeto:

- 1 Coloque un símbolo de clip de película con el ID de vínculo `theID` en la biblioteca.
- 2 En el panel Acciones (Ventana > Paneles de desarrollo > Acciones), utilice la acción `function` para definir una clase nueva, tal como se muestra a continuación:

```
// defina una clase  
function myClipClass() {}
```

La nueva clase se asignará a todas las instancias de clip de película agregadas a la aplicación mediante la línea de tiempo o con el método `attachMovie()` o `duplicateMovieClip()`. Si desea que estos clips de película puedan acceder a los métodos y propiedades del objeto incorporado `MovieClip`, deberá hacer que la nueva clase herede de la clase `MovieClip`.

- 3 Introduzca código como el del siguiente ejemplo:

```
// heredar de la clase MovieClip  
myClipClass.prototype = new MovieClip();
```

Ahora, la clase `myClipClass` hereda todas las propiedades y métodos de la clase `MovieClip`.

- 4 Introduzca el código siguiente para definir los métodos de controlador de eventos para la nueva clase:

```
// defina los métodos de controlador de eventos para la clase myClipClass
```

```
myClipClass.prototype.onload = function() {trace ("se ha cargado el clip de
película");}
myClipClass.prototype.onEnterFrame = function() {trace ("el clip de película
ha entrado en el fotograma");}
```

- 5 Seleccione Ventana > Biblioteca para abrir el panel Biblioteca si aún no está abierto.
- 6 Seleccione los símbolos que desea asociar con la nueva clase y seleccione la opción Vinculación en el menú emergente situado en la parte superior derecha del panel Biblioteca.
- 7 En el cuadro de diálogo Propiedades de vinculación, seleccione Exportar para ActionScript.
- 8 Introduzca un identificador en el cuadro Identificador.  
El identificador debe ser el mismo para todos los símbolos que desea asociar con la nueva clase. En el ejemplo myClipClass, el identificador es theID.
- 9 En el panel Script, introduzca el código siguiente:

```
// registre la clase
Object.registerClass("theID", myClipClass);
_root.attachMovie("theID", "myName", 1);
```

Este código registra cualquier símbolo cuyo identificador de vínculo sea theID con la clase myClipClass. Todas las instancias de myClipClass tienen métodos de controlador de eventos que se comportan tal como se definió en el paso 4. También se comportan como todas las instancias de la clase MovieClip, puesto que en el paso 3 se indicó que la nueva clase debe heredar de la clase MovieClip.

```
function myClipClass() {}

myClipClass.prototype = new MovieClip();
myClipClass.prototype.onload = function(){
    trace("se ha cargado el clip de película");
}
myClipClass.prototype.onPress = function(){
    trace("presionado");
}

myClipClass.prototype.onEnterFrame = function(){
    trace("el clip de película ha entrado en el fotograma");
}

myClipClass.prototype.myfunction = function(){
    trace("se ha llamado al método myfunction");
}

Object.registerClass("myclipID", myClipClass);
_root.attachMovie("myclipID", "ablue2", 3);
```

## Creación de herencia en ActionScript 1

La herencia es una forma de organizar, ampliar y reutilizar funciones. Las subclases heredan las propiedades y los métodos de las superclases y agregan sus propios métodos y propiedades especializados. Para poner un ejemplo real, podemos decir que Bicicleta sería una superclase y que Bicicleta de montaña (BTT) y Triciclo serían subclases de dicha superclase. Ambas subclases contienen, o *heredan*, los métodos y las propiedades de la superclase (por ejemplo, wheels). Cada subclase tiene, asimismo, sus propias propiedades y métodos que se amplían a la superclase (la subclase BTT, por ejemplo, tendría una propiedad gears). Para crear herencia en ActionScript pueden utilizarse los elementos prototype y \_\_proto\_\_.

Todas las funciones constructoras tienen una propiedad `prototype` que se crea automáticamente cuando se define la función. La propiedad `prototype` indica los valores predeterminados de la propiedad para los objetos creados con esa función. La propiedad `prototype` puede utilizarse para asignar propiedades y métodos a una clase. Para más información, consulte [“Asignación de métodos a un objeto personalizado en ActionScript 1” en la página 803](#).

Todas las instancias de una clase tienen una propiedad `__proto__` que indica de qué objeto heredan métodos y propiedades. Al utilizar una función constructora para crear un objeto, se define la propiedad `__proto__` para hacer referencia a la propiedad `prototype` de su función constructora.

La herencia se comporta de acuerdo a una jerarquía determinada. Cuando se llama a la propiedad o al método de un objeto, ActionScript busca en el objeto para ver si existe tal elemento. Si no existe, ActionScript busca la información (`myObject.__proto__`) en la propiedad `__proto__` del objeto. Si la propiedad no pertenece al objeto `__proto__` del objeto, ActionScript busca en `myObject.__proto__.__proto__`, y así sucesivamente.

En el ejemplo siguiente se define la función constructora `Bike()`:

```
function Bike (length, color) {  
    this.length = length;  
    this.color = color;  
}
```

En el código siguiente se agrega el método `roll()` a la clase `Bike`:

```
Bike.prototype.roll = function() {this._x = _x + 20;;};
```

En lugar de agregar `roll()` a la clase `MountainBike` y a la clase `Tricycle`, puede crear la clase `MountainBike` con la clase `Bike` como superclase:

```
MountainBike.prototype = new Bike();
```

A continuación, puede llamar al método `roll()` de `MountainBike`, como se indica a continuación:

```
MountainBike.roll();
```

Los clips de película no heredan entre sí. Para crear herencia en los clips de película, puede utilizar el método `Object.registerClass()` para asignar una clase distinta de `MovieClip` a los clips de película. Véase `Object.registerClass()` en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

Para más información sobre la herencia, consulte las entradas `Object.__proto__`, `#initclip`, `#endinitclip` y `super` en el [Capítulo 12, “Diccionario de ActionScript”, en la página 213](#).

## Adición de propiedades de captador/definidor a objetos en ActionScript 1

Puede crear propiedades de captador/definidor para un objeto utilizando el método `Object.addProperty()`.

La función captador es una función que carece de parámetros. El valor devuelto puede ser de cualquier tipo. El tipo de valor puede cambiar según la invocación. El valor devuelto se trata como el valor actual de la propiedad. La función definidor es una función que acepta un parámetro: el nuevo valor de la propiedad. Por ejemplo, si la propiedad `x` se asigna mediante la sentencia `x = 1`, la función definidor recibirá el parámetro 1 del tipo número. El valor devuelto de la función definidor se ignora.

Cuando Flash lee una propiedad de captador/definidor, invoca la función captador y el valor devuelto por la función se convierte en un valor de `prop`. Cuando Flash escribe una propiedad de captador/definidor, invoca la función definidor y le pasa el nuevo valor como parámetro. Si existe una propiedad con ese nombre concreto, la nueva propiedad lo sobrescribe.

Puede agregar propiedades de captador/definidor a los objetos prototipo. Si agrega una propiedad de captador/definidor a un objeto prototipo, todas las instancias del objeto que heredan el objeto prototipo heredarán la propiedad de captador/definidor. Esto hace posible agregar una propiedad de captador/definidor a una ubicación, el objeto prototipo, y aplicarla a todas las instancias de una clase (como si se agregaran métodos a objetos prototipo). Si se invoca una función captador/definidor para una propiedad de captador/definidor de un objeto prototipo heredado, la referencia que se pasa a la función captador/definidor será el objeto referenciado originalmente, no el objeto prototipo.

Para más información, consulte `Object.addProperty()` en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

El comando Depurar > Mostrar variables en el modo de prueba es compatible con las propiedades de captador/definidor que se añaden a los objetos mediante el método `Object.addProperty()`. Las propiedades que se añaden así a un objeto se muestran junto con otras propiedades del mismo en el panel Salida. Las propiedades de captador/definidor se identifican en el panel Salida con el prefijo `[getter/setter]`. Para más información sobre el comando Mostrar variables, consulte [“Utilización del panel Salida” en la página 80](#).

## Utilización de las propiedades del objeto Function en ActionScript 1

Puede especificar el objeto al que se aplica una función y los valores de los parámetros que se pasan a la función mediante los métodos `call()` y `apply()` del objeto `Function`. Cada función de ActionScript se representa mediante un objeto `Function`, de modo que todas las funciones admiten los métodos `call()` y `apply()`. Al crear una clase personalizada mediante una función constructora, o al definir métodos para una clase personalizada utilizando una función, puede invocar los métodos `call()` y `apply()` para la función.

### Invocación de una función mediante el método `Function.call()` en ActionScript 1

El método `Function.call()` invoca la función representada por un objeto `Function`.

En casi todos los casos puede utilizarse el operador de llamada de función `()` en lugar del método `call()`. El operador de llamada de función hace que el código sea conciso y legible. El método `call()` es de gran utilidad cuando debe controlarse explícitamente el parámetro `this` de la llamada de función. Normalmente, si se invoca una función como método de un objeto, el parámetro `this` se establece en `myObject` dentro del cuerpo de la función como en el caso siguiente:

```
myObject.myMethod(1, 2, 3);
```

En algunos casos, es posible que desee que `this` haga referencia a otro elemento; por ejemplo, si debe invocarse una función como un método de un objeto, pero en realidad no se almacena como método de dicho objeto.

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

Puede pasar el valor `null` para el parámetro *thisObject* para invocar una función como función regular y no como un método de un objeto. Por ejemplo, las llamadas de función siguientes son equivalentes:

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

Para más información, consulte [Function.call\(\)](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

**Para invocar una función mediante el método Function.call:**

- Utilice la siguiente sintaxis.

```
myFunction.call(thisObject, parameter1, ..., parameterN)
```

Este método utiliza los siguientes parámetros:

- El parámetro *thisObject* especifica el valor de *this* en el cuerpo de la función.
- Los parámetros *parameter1...*, *parameterN* especifican parámetros que se pasan a *myFunction*. Puede especificar cero o más parámetros.

## Especificación de un objeto al que se aplica una función mediante **Function.apply()** en ActionScript 1

El método `Function.apply()` especifica el valor de *this* que debe utilizarse en una función que llama `ActionScript`. Este método también especifica parámetros que deben pasarse a la función llamada.

Los parámetros se especifican como un objeto `Array`. Suele ser útil cuando no se conoce el número de parámetros hasta que se ejecuta el script.

Para más información, consulte [Function.apply\(\)](#) en el [Capítulo 12, “Diccionario de ActionScript”](#), en la [página 213](#).

**Para especificar el objeto al que se aplica una función mediante el método Function.apply():**

- Utilice la siguiente sintaxis.

```
myFunction.apply(thisObject, argumentsObject)
```

Este método utiliza los siguientes parámetros:

- El parámetro *thisObject* especifica el objeto al que se aplica *myFunction*.
- El parámetro *argumentsObject* define una matriz cuyos elementos se pasan a *myFunction* como parámetros.



# ÍNDICE ALFABÉTICO

## A

acceso a propiedades de objeto 52

acciones

definición 28

repetir 58

acciones asíncronas 186

Acciones, panel 60

ActionScript

asignar la clase ActionScript 2.0 a los clips de película 135

información general sobre ActionScript 2.0 25, 161

mensajes de error del compilador de

ActionScript 2.0 783

strict data typing no compatible con

ActionScript 1 41

agrupar sentencias 33

ajuste de texto en el código, activar 70

añadir notas a los scripts 35

animación, símbolos 39

aplicaciones Web, conexión continua 192

aplicar formato al código 70

aplicar reproducción indefinida 58, 59

acciones 59

archivo SWD, definición 72

archivos de clase externos

crear 164

utilizar rutas de clases para localizar 175

archivos de clase, crear 164

archivos de política 198

deben denominarse crossdomain.xml 198

política, archivos

*Véase también* seguridad

archivos JPEG

cargar en clips de película 127, 202

incorporar en campos de texto 156

precargar 207

archivos MP3

cargar en clips de película 203

precargar 208

y etiquetas ID3 204

archivos remotos, comunicación 185

archivos SWF

cargar en clips de película 202

cargar y descargar 125

colocar en página Web 95

controlar en Flash Player 195

crear controles de sonido 103

escalar respecto a Flash Player 194

incorporar en campos de texto 156

mantener el tamaño original 194

pasar información 186

precargar 207

saltar a fotogramas o a escenas 94

*Véase también* clips de película

archivos SWF cargados

eliminar 125

identificar 53

argumentos. *Véase* Parámetros

arquitectura basada en componentes, definición 123

arrastrar clips de película 128

ASCII, códigos de tecla

obtener 98

otras teclas 794

teclado numérico 792

teclas de función 793

teclas de letra o número 791

asignación de nombre a variables 43, 64

asociatividad, de operadores 48, 789

atributo private para miembros de la clase 170

atributo public para miembros de la clase 170

## B

balance (sonido), controlar 105  
bucles for y miembros de instancia 171

## C

cadenas de concatenación 37  
caja de herramientas Acciones 60  
elementos en amarillo 63  
campos de texto 139  
    aplicar hojas de estilos en cascada 146  
    crear y eliminar en tiempo de ejecución 141  
    determinar el tamaño necesario 143  
    evitar conflictos de nombres de variables 141  
    formato 141  
    formato con hojas de estilos en cascada, aplicar 143  
    hacer que el texto fluya alrededor  
        de las imágenes 151, 153  
    mostrar propiedades para depuración 81  
    nombres de instancia y de variable comparados 140  
    propiedades predeterminadas 142  
    y texto HTML 147  
    *Véase también* clase TextField, clase TextFormat,  
        clase TextField.StyleSheet  
capturar teclas presionadas 98  
caracteres especiales 37  
clase TextField 140  
    crear texto desplazable 158  
clase TextField.StyleSheet 143  
    crear estilos de texto 146  
    y hojas de estilos en cascada 145  
    y propiedad TextField.styleSheet 143, 146  
clase TextFormat 141  
clases  
    ampliar 169  
    ampliar en tiempo de ejecución 180  
    asignación de nombres 167  
    asignar a clips de película 135  
    compilar y exportar 180  
    crear archivos de clase externos 164  
    crear propiedades y métodos 167  
    crear subclases 169  
    crear y utilizar 167  
    definición 29, 115  
    definidas sólo en archivos externos 164, 167  
    dinámicas 180  
    ejemplo de creación 163  
    especificar fotograma de exportación 180  
    get/set, métodos 179  
    importar 178  
    inicializar propiedades en línea 168  
    inicializar propiedades en tiempo de ejecución 136  
    interfaces 173–175  
    miembros de instancia y miembros de clase 171  
    organizar en paquetes 177  
    public y private, atributos de miembros 170  
    rutas de clases 175  
    sobrecarga no admitida 170  
    solucionar referencias a clases 175  
    y programación orientada a objetos 162  
    *Véase también* clases incorporadas  
clases dinámicas 180  
clases, incorporadas 115–122  
    ampliar 169  
clips de película  
    activar con teclado 100  
    ajustar color 102  
    añadir parámetros 131  
    anidados, definición 123  
    arrastrar 128  
    asignar estados de botón 89  
    asignar nombre de instancia 53  
    asociar a símbolo en el escenario 129  
    asociar controladores on() y onClipEvent() 89  
    cambiar propiedades del depurador 76  
    cambiar propiedades durante la reproducción 127  
    cargar archivos MP3 203  
    cargar archivos SWF y JPEG 202  
    compartir 129  
    control 123  
    crear en tiempo de ejecución 128  
    crear subclases 135  
    crear una instancia vacía 129  
    detectar colisiones 106  
    determinar la profundidad disponible siguiente 132  
    determinar profundidad de 133  
    duplicar 129  
    eliminar 129  
    funciones 124  
    gestionar profundidad 132  
    incorporar en campos de texto 156  
    inicializar propiedades en tiempo de ejecución 136  
    iniciar y detener 94  
    invocar métodos 124  
    llamar a varios métodos 124  
    métodos 124  
    métodos y funciones comparados 123  
    métodos, utilizar para dibujar formas 133  
    mostrar objetos 80  
    mostrar variables 81

- nombre de instancia, definición 123
  - principal, definición 123
  - propiedades 127
  - propiedades, inicializar en tiempo de ejecución 136
  - reproducir indefinidamente a través
    - de los subniveles 59
  - secundario, definición 123
  - tipo de datos 39
  - utilizar como máscaras 134
  - y propiedad `_root` 126
  - y sentencia `with` 124
  - Véase también* archivos SWF
  - clips de película anidados, definición 123
  - clips de película principales, definición 123
  - clips de película secundarios, definición 123
  - clips de película, detener 94
  - clips de película, reproducir 94
  - codificación predeterminada 28
  - código
    - ajuste de texto 70
    - desplazarse por las líneas 78
    - formato 70
    - mostrar números de línea 70
    - seleccionar una línea 78
  - colisiones, detectar 106
    - entre clips de película 108
    - entre un clip de película y un punto del escenario 108
  - colores
    - en caja de herramientas Acciones 63
    - Script, panel 63
    - valores, definir 102
  - combinar operaciones 51
  - comentarios 35
  - comillas, incluir en cadenas 37
  - comprobar
    - datos cargados 186
    - sintaxis y puntuación 69
  - comunicación con Flash Player 193
  - condiciones, comprobar 58
  - conexión TCP/IP
    - con objeto `XMLSocket` 192
    - enviar información 186
  - conexiones de socket
    - información 192
    - script de muestra 193
  - constantes 29, 36
  - constructores
    - definición 29
    - información general 170
  - contadores, repetir acción 58, 59
  - contraseñas y depuración remota 72
  - controladores de eventos
    - ámbito 90
    - asignar funciones 86
    - asociar a botones o a clips de película 88
    - comprobar datos XML 186
    - definición 29, 85
    - definidos por clases de `ActionScript` 86
    - `on()` y `onClipEvent()` 88
  - controladores `on()` `onClipEvent()` 88
    - ámbito 90
    - asociar a clips de película 89
  - controladores. *Véase* controladores de eventos
  - controles `ActiveX` 195
  - controles de teclado
    - para activar clips de película 100
    - y Probar película 71
  - convenciones de asignación de nombre
    - para clases 167
    - para paquetes 177
  - convenciones tipográficas 10
  - convertir tipos de datos 37, 41
  - corchetes. *Véase* operadores de acceso a matriz
  - crear instancias de objetos 115
  - crear objetos 115
  - CSS. *Véase* hojas de estilos en cascada
  - cuadro de mensaje, visualizar 194
  - cursores, crear personalizados 96
- D**
- datos cargados, comprobar 186
  - datos externos 185
    - acceso entre archivos SWF de varios dominios 197
    - comprobar si están cargados 186
    - enviar y cargar 185
    - funciones de seguridad 196
    - y mensajes 193
    - y objeto `LoadVars` 188
    - y objeto `XMLSocket` 192
    - y scripts de servidor 187
    - y XML 189
  - depuración remota 72
  - depurador
    - botones 79
    - establecer puntos de corte 77
    - lista Observación 75
    - Propiedades, ficha 76
    - Reproductor de depuración de Flash 71
    - seleccionar en el menú contextual 74

- utilizar 71
- variables 74
- depurar 71
  - con la sentencia trace 82
  - desde ubicación remota 72
  - gestión de excepciones 15
  - mensajes de error del compilador 783
  - mostrar objetos 80
  - mostrar variables 81
  - propiedades de campo de texto 81
  - Reproductor de depuración 71
  - utilizar el panel Salida 80
- desplazar texto 158
- desplazarse por las líneas de código 78
- detectar colisiones 106
- detectores de eventos 87
  - ámbito 90
  - definidos por clases de ActionScript 88
- determinar el tipo de variables 39
- dibujar
  - formas 133
  - líneas y rellenos 109
- direcciones IP
  - y archivos de política 199
  - y seguridad 196
- DOM (modelo de objetos de documento), XML 189
- duplicar, clips de película 129

## E

- ECMA-262
  - conformidad 18
  - especificación 27
- editor de ActionScript 60, 63
- elementos multimedia externos 201–209
  - cargar archivos MP3 203
  - cargar archivos SWF y JPEG 202
  - información general sobre la carga 201
  - precargar 206, 207, 208
  - razones para utilizar 201
  - reproducir archivos FLV 205
- eliminar
  - archivos SWF cargados 125
  - clips de película 129
- emparejamiento de puntuación, comprobar 69
- enviar información
  - a archivos remotos 185
  - en formato XML 186
  - formato URL codificado 186
  - mediante TCP/IP 186

- errores
  - lista de mensajes de error 783
  - nombre, conflicto 45
  - sintaxis 63
- escenario, asociar símbolos a los clips de película 129
- etiquetas ID3 204
- evento de usuario, definición 85
- evento del sistema, definición 85
- eventos, definición 29, 85
- exportar scripts, codificación de lenguaje 28
- expresiones
  - asignar varias variables 51
  - comparar valores 49
  - definición 29
  - manipular valores 47

## F

- fase de compilación, definición 10
- ficha Observación, depurador 75
- ficha Propiedades, depurador 76
- ficha Variables, depurador 74
- fixar scripts en un lugar 62
- Flash Player
  - atenuar menú contextual 194
  - comunicación 193
  - conseguir la última versión 82
  - depurar 71
  - escalar archivos SWF a 194
  - métodos 195
  - vista de menú normal 194
  - visualización a pantalla completa 194
  - visualizar menú contextual 194
- Flash Player 7
  - elementos de lenguaje nuevos y modificados 15
  - nuevo modelo de seguridad 17, 19, 21, 23
  - transferir scripts existentes 17, 199
  - y conformidad con ECMA-262 18
- FLV (vídeo externo), archivos 205
  - precargar 208
- formato MIME, estándar 187
- formato URL codificado, enviar información 186
- fscommand(), función
  - comandos y argumentos 194
  - comunicación con Director 195
  - utilizar 193
- fuentes de dispositivo, enmascarar 135
- fuentes externas, conectar Flash 185
- funciones 29
  - asíncronas 186
  - constructor 802

- conversión 37
- definición 54
- devolver valores 56
- incorporadas 54
- llamar 56
- muestra 31
- para controlar clips de película 124
- pasar parámetros 55
- personalizadas 54
- variables locales 55
- y métodos 30

funciones constructoras, muestra 802

funciones de conversión 37

funciones incorporadas 54

funciones personalizadas 54

## G

gestión de excepciones 15

get/set, métodos de clases 179

getAscii(), método 98

## H

hacer referencia a variables 44

herencia 162

- permitida sólo desde una clase 169
- y subclases 169

herencia múltiple, no permitida 169

hitTest(), método 106

hojas de estilos en cascada

- aplicar a campos de texto 146
- aplicar clases de estilos 147
- asignar estilos a etiquetas HTML incorporadas 147
- cargar 145
- combinar estilos 147
- definir estilos en ActionScript 146
- ejemplo de utilización con etiquetas HTML 148
- ejemplo de utilización con etiquetas XML 150
- formato al texto, aplicar 143
- propiedades admitidas 144
- utilizar para definir etiquetas nuevas 150
- y clase TextField.StyleSheet 145

hojas de estilos. *Véase* hojas de estilos en cascada

HTML

- aplicar estilos a etiquetas incorporadas 147
- ejemplo de utilización con estilos 148
- etiquetas admitidas 152
- etiquetas entre comillas 152
- utilizar en campos de texto 152

- utilizar hojas de estilos en cascada para definir etiquetas 150
- utilizar la etiqueta <img> para que el texto fluya 151, 153, 156

## I

iconos

- depurador 79
- encima del panel Script 61

identificador de vínculo 129, 135

identificadores, definición 30

idiomas, utilizar varios en scripts 28

imágenes

- cargar en clips de película 127
- incorporar en campos de texto 156
- Véase también* elementos multimedia externos

importar

- clases 178
- scripts, codificación de lenguaje 28

importar clases 178

información, pasar entre archivos SWF 186

inicializar propiedades de clips de película 136

instancias

- definición 30, 115
- ejemplo de creación 166

interactividad, en archivos SWF

- crear 93
- técnicas 96

interfaces 163

- crear y utilizar 173–175

## J

JavaScript

- documentación de Netscape Navigator 27
- enviar mensajes 194
- estándar internacional 27
- sentencia alert 82
- y ActionScript 27
- y Netscape 195

## L

Lenguaje extensible de marcado *Véase* XML

llamar a métodos 39

llaves 33

- comprobación de pares emparejados 69

llaves. *Véase* llaves

loadMovie(), función 186

loadVariables(), función 186

LoadVars, objeto 188

## M

- Macromedia Director, comunicación 195
- manipular números 38
- máscaras 134
  - trazos ignorados 133, 134
  - y fuentes de dispositivo 135
- matrices multidimensionales 53
- matrices, multidimensionales 53
- menú emergente Ver opciones 69, 70
- método `getURL()` 95
- métodos
  - asíncronas 186
  - de objetos, llamar 116
  - declarar 167
  - definición 30
  - para controlar clips de película 124
- métodos abreviados de teclado
  - para scripts fijados 62
- métodos de controlador de eventos 85
- miembros de clase 116
  - creados una vez por cada clase 171
  - crear 171
  - ejemplo de utilización 172
  - y subclases 173
- miembros de instancia 171
- miembros estáticos. *Véase* miembros de clase
- modelo de eventos
  - para controladores `on()` y `onClipEvent()` 88
  - para detectores de eventos 87
  - para métodos de controlador de eventos 85
- Mostrar objetos, comando 80
- Mostrar variables, comando 81
- `movienamename_DoFSCCommand`, función 194

## N

- navegación
  - control 93
  - saltar a fotogramas o a escenas 94
- Navegador de scripts 61
- Netscape DevEdge Online 27
- Netscape, métodos admitidos JavaScript 195
- niveles 53
  - cargar 125
- nodo secundario 189
- nodos 189
- nombre, conflictos 45
- nombres de dominio y seguridad 196

- nombres de instancia
  - asignar 53
  - comparados con nombres de variable 140
  - definición 30, 123
  - establecer dinámicamente 52
- números
  - convertir a números enteros de 32 bits 50
  - manipular 38
- números de línea en el código, mostrar 70

## O

- objects
  - tipo de datos 38
- objeto detector 87
  - no registrar 88
- objeto difusor 87
- objetos
  - acceder a propiedades 116
  - crear 115
  - definición 30
  - llamar a métodos 116
  - reproducir indefinidamente a través de los subniveles 59
  - y programación orientada a objetos 162
- obtener información de archivos remotos 185
- obtener posición del puntero del ratón 97
- `onClipEvent()`, controladores 111
- Opciones, menú emergente
  - Acciones, panel 62, 63
  - depurador 73
  - panel Salida 80
- operadores 30
  - acceso a matriz 52
  - asignación 51
  - asociatividad 48, 789
  - cadena 49
  - combinar con valores 47
  - comparación 49
  - en modo bit 50
  - igualdad 50
  - lógicos 50
  - numéricos 48
  - precedencia 789
  - punto 52
- operadores de acceso a matriz 52
  - comprobación de pares emparejados 69
- operadores de asignación
  - compuestos 51
  - diferentes de operadores de igualdad 50
  - información 51

- operadores de cadena 49
- operadores de comparación 49
- operadores de igualdad 50
  - diferentes de operadores de asignación 50
  - estricta 51
- operadores de igualdad estricta 51
- operadores de punto 52
- operadores en modo bit 50
- operadores lógicos 50
- operadores numéricos 48
- orden de ejecución
  - asociatividad de operadores 48
  - precedencia de operadores 48
  - scripts 57

## P

- palabra clave this 110
- palabras clave 30
  - enumeradas 36
- palabras reservadas. *Véase* palabras clave
- panel Script
  - botones 61
  - información 60
  - trabajo con scripts 61
- paquetes 177
  - asignación de nombres 177
- parámetros
  - definición 31
  - entre paréntesis 34
  - pasar a funciones 55
- paréntesis 34
  - comprobación de pares emparejados 69
- pasar valores
  - por contenido 46
  - por referencia 47
- pausa en el (desplazarse por el) código 78
- posición del ratón, obtener 97
- precedencia de operadores 789
- Probar película
  - y controles de teclado 71
  - y Unicode 71
- probar. *Véase* depurar
- profundidad
  - definición 132
  - determinar instancia en 133
  - determinar para clips de película 133
  - determinar valor disponible siguiente 132
  - gestionar 132

- programación orientada a objetos 162
  - Véase también* clases
- propiedades
  - acceso 52
  - constante 36
  - de clips de película 127
  - de objetos, acceder 116
  - declarar 167
  - definición 31
  - inicializar en tiempo de ejecución 136
- propiedades de objeto
  - acceso 52
- Propiedades de vinculación, cuadro
  - de diálogo 129, 135
- propiedades del objeto
  - asignar valores a 116
- protocolo HTTP 186
  - comunicación con scripts de servidor 187
- protocolo HTTPS 186
- proyectores, ejecutar aplicaciones 194
- puntero de ratón. *Véase* cursores
- puntero. *Véase* cursores
- punto de registro e imágenes cargadas 127
- punto y coma 34
- puntos de corte
  - establecer en el depurador 77
  - información 77
  - y archivos externos 77

## R

- propiedad \_root y clips de película cargados 126
- recursos, adicionales 10
- repetir acciones 58
- Reproductor de depuración 71
- requisitos del sistema 9
- resaltar sintaxis 63
- rutas de clases
  - buscar orden de 175
  - definición 175
  - globales y de documento 175
  - modificar 176
- rutas de clases de documento 175
- rutas de clases globales 175
- rutas de destino
  - definición 31
  - especificar 53
  - introducir 53

## S

- Salida, panel 80
    - Mostrar objetos, comando 80
    - Mostrar variables, comando 81
    - opciones 80
    - y la sentencia trace 82
  - saltar a una URL 95
  - sangría en el código, activar 70
  - script de muestra 110
  - Script, ventana (sólo para Flash Professional) 60
  - scripts
    - comentar 35
    - controlar ejecución 57
    - controlar el flujo 57
    - declarar variables 46
    - depurar 71
    - escritura y depuración 57
    - fijar en un lugar 62
    - importar y exportar 28
    - métodos abreviados de teclado para
      - scripts fijados 62
    - muestra 110
    - probar 71
    - problemas de visualización de texto, corregir 28
    - transferir a Flash Player 7 17, 199
  - scripts de servidor
    - formato XML 190
    - lenguajes 185
  - secuencias de caracteres. *Véase* strings
  - secuencias de escape 37
  - seguridad 196–200
    - acceso de datos en varios dominios 197, 198
    - y archivos de política 198
    - y transferir scripts a Flash Player 7 19, 21, 23
  - sentencias
    - agrupar 33
    - terminación 34
    - trace, sentencias 82
  - sentencias de terminación 34
  - servidores, abrir conexión continua 192
  - setRGB, método 102
  - sintaxis
    - barra 33
    - comprobar 69
    - distinción entre mayúsculas y minúsculas 31–32
    - llaves 33
    - paréntesis 34
    - punto 32
    - punto y coma 34
    - reglas 31
    - resaltar 63
  - sintaxis con barras 33
    - no admitida en ActionScript 2.0 33
  - sintaxis con punto 32
  - sitios remotos, conexión continua 192
  - solucionar problemas *Véase* depurar
  - sonidos
    - asociar a línea de tiempo 104
    - control 103
    - control de balance 105
    - Véase también* elementos multimedia externos
  - sonidos, asociar 104
  - strict data typing 40
    - no compatible con ActionScript 1 41
    - y variables globales 40
    - y variables locales 45
  - strings 37
  - subclases
    - crear 169
    - crear para clips de película 135
    - y miembros de clase 173
  - sufijos 64
  - sugerencias para el código 64
    - activar 64, 66
    - especificar configuración 66
    - no aparecen 67
    - utilizar 66
    - visualización manual 68
  - sugerencias. *Véase* sugerencias para el código
- ## T
- tecla Tabulador, y Probar película 71
  - teclado numérico, valores del código
    - de tecla ASCII 792
  - teclado, valores de código de tecla ASCII 791
  - teclas de función, valores del código
    - de tecla ASCII 793
  - teclas de método abreviado de Esc 68
  - teclas presionadas, capturar 98
  - terminología 28
  - texto
    - asignar a un campo de texto en tiempo
      - de ejecución 140
    - codificar 28
    - desplazar 158
    - determinar el tamaño necesario del objeto TextField 143
    - obtener información sobre medidas 143



- utilizar la etiqueta <img> para que el texto fluya alrededor de las imágenes 153
- Véase también* campos de texto
- texto, codificar 28
- tiempo de ejecución, definición 10
- tipo de datos null 39
- tipo de datos undefined 39
- tipos de datos 36
  - asignar a elementos 39
  - asignar automáticamente 40
  - Boolean 38
  - convertir 37, 41
  - declarar 40
  - definición 29
  - determinar 39
  - MovieClip 39
  - null 39
  - Number 38
  - Object 38
  - strictly typing 40
  - String 37
  - undefined 39
- tipos de datos de referencia 36
- tipos de datos primitivos 36
- transferir variables entre una película y un servidor 188

## U

- Unicode
  - soporte 28
  - y el comando Probar película 28, 71
- UTF-8 (Unicode) 28

## V

- valores ASCII 98
  - otras teclas 794
  - teclas de función 793
  - teclas de teclado numérico 792
  - teclas del teclado 791
- valores booleanos 38
  - comparar 50
  - definición 29
- valores, manipular en expresiones 47
- variables
  - ámbito 44
  - asignación de nombre 64
  - asignar varias 51
  - comprobar y establecer valores 43
  - convertir a XML 190
  - definición 31

- determinar el tipo de datos 39
- enviar a una URL 95
- establecer dinámicamente 52
- evitar conflictos de nombres 141
- hacer referencia al valor 47
- información 43
- modificar en el depurador 75
- pasar contenido 46
- probar 43
- reglas de asignación de nombres 43
- sufijos 64
- transferir entre una película y un servidor 188
- utilizar en scripts 46
  - y ficha Variables del depurador 74
  - y lista Observación del depurador 75
- variables de línea de tiempo 45
- variables globales 45
  - y strict data typing 40
- variables locales 44
  - en funciones 55
  - muestra 45
  - y strict data typing 45
- varios idiomas, utilizar en scripts 28
- vídeo, alternativa a la importación 205
- vincular, clips de película 129
- volumen, crear control deslizable 105

## X

- XML 189
  - conversión de variables de muestra 189
  - DOM 189
  - ejemplo de utilización con estilos 150
  - en scripts de servidor 190
  - enviar información con métodos XML 186
  - enviar información mediante socket TCP/IP 186
  - jerarquía 189
- XML, métodos de clase 189
- XMLSocket, objeto
  - comprobar datos 186
  - métodos 192
  - utilizar 192

