Guía del desarrollador de ACTIONSCRIPT® 3.0

Adobe

Avisos legales

Para ver los avisos legales, consulte http://help.adobe.com/es_ES/legalnotices/index.html.

Última modificación 12/12/2014

Contenidos

Capítulo 1: Trabajo con fechas y horas
Administración de fechas de calendario y horas 1
Control de intervalos de tiempo 4
Ejemplo de fecha y hora: Un sencillo reloj analógico 6

Capítulo 2: Trabajo con cadenas
Fundamentos de la utilización de cadenas 10
Creación de cadenas 11
La propiedad length 12
Trabajo con caracteres en cadenas 12
Comparación de cadenas 13
Obtención de representaciones de cadena de otros objetos 14
Concatenación de cadenas 14
Búsqueda de subcadenas y patrones en cadenas 15
Conversión de cadenas de mayúsculas a minúsculas y viceversa 19
Ejemplo de cadenas: Arte ASCII 19

Capítulo 3: Trabajo con conjuntos
Fundamentos de la utilización de conjuntos 25
Conjuntos indexados 27
Conjuntos asociativos 38
Conjuntos multidimensionales 41
Clonación de conjuntos 43
Ampliación de la clase Array 44
Ejemplo de conjuntos: PlayList 48

Capítulo 4: Gestión de errores
Fundamentos de la gestión de errores 53
Tipos de errores 54
Gestión de errores en ActionScript 3.0 57
Trabajo con las versiones de depuración de los motores de ejecución de Flash 58
Gestión de errores sincrónicos en una aplicación 59
Creación de clases de error personalizadas 63
Respuesta al estado y a los eventos de error 64
Comparación de las clases Error 68
Ejemplo de gestión de errores: Aplicación CustomErrors 71

Capítulo 5: Uso de expresiones regulares
Fundamentos de la utilización de expresiones regulares 77
Sintaxis de las expresiones regulares 78
Métodos para utilizar expresiones regulares con cadenas 91
Ejemplo de expresiones regulares: Analizador Wiki 92

Capítulo 6: Trabajo con XML
Fundamentos de la utilización de XML 97
El enfoque E4X del procesamiento de XML 100
Objetos XML 101
Objetos XMLList 104
Inicialización de variables XML 105
Construcción y transformación de objetos XML 106
Navegación de estructuras XML 108
Uso de espacios de nombres XML 112
Conversión de tipo XML 113
Lectura de documentos XML externos 114
Ejemplo de XML en ActionScript: Carga de datos RSS desde Internet 115

Capítulo 7: Uso de la funcionalidad JSON nativa
Descripción de la API JSON 119
Definición de comportamiento JSON personalizado 120

Capítulo 8: Gestión de eventos
Fundamentos de la gestión de eventos 127
Diferencias entre la gestión de eventos en ActionScript 3.0 y en las versiones anteriores 129
Flujo de eventos 131
Objetos de evento 133
Detectores de eventos 137
Ejemplo de gestión de eventos: Reloj con alarma 143

Capítulo 9: Trabajo con dominios de aplicación

Capítulo 10: Programación de la visualización
Fundamentos de la programación de la visualización 155
Clases principales de visualización 158
Ventajas de la utilización de la lista de visualización 160
Trabajo con objetos de visualización 162
Manipulación de objetos de visualización 178
Animación de objetos 198
Orientación del escenario 201
Carga dinámica de contenido de visualización 204
Ejemplo de objeto de visualización: SpriteArranger 210

Capítulo 11: Trabajo con la geometría
Fundamentos de geometría 217
Uso de objetos Point 218
Uso de objetos Rectangle 220
Uso de objetos Matrix 223
Ejemplo de geometría: Aplicación de una transformación de matriz a un objeto de visualización 225

Capítulo 12: Uso de la API de dibujo
Aspectos básicos de la API de dibujo 229
Clase Graphics 230

Dibujo de líneas y curvas 230
Dibujo de formas mediante los métodos incorporados 233
Creación de líneas y rellenos degradados 234
Uso de la clase Math con los métodos de dibujo 238
Animación con la API de dibujo 239
Ejemplo de API de dibujo: Generador visual algorítmico 240
Utilización avanzada de la API de dibujo 243

Capítulo 13: Trabajo con mapas de bits
Fundamentos de la utilización de mapas de bits 251
Las clases Bitmap y BitmapData 253
Manipulación de píxeles 255
Copia de datos de mapas de bits 258
Compresión de datos de mapa de bits 259
Creación de texturas con funciones de ruido 260
Desplazamiento por mapas de bits 262
Aprovechamiento de la técnica de mipmapping 263
Ejemplo de mapa de bits: Luna giratoria animada 264
Descodificación asíncrona de imágenes de mapa de bits 275

Capítulo 14: Aplicación de filtros a objetos de visualización
Fundamentos de la aplicación de filtros a los objetos de visualización 279
Creación y aplicación de filtros 280
Filtros de visualización disponibles 287
Ejemplo de filtrado de objetos de visualización: Filter Workbench 305

Capítulo 15: Trabajo con sombreados de Pixel Bender
Fundamentos de los sombreados de Pixel Bender 314
Carga e incorporación de un sombreado 316
Acceso a los metadatos de sombreado 318
Especificación de valores de entrada y parámetro de sombreado 319
Uso de un sombreado 324

Capítulo 16: Trabajo con clips de película
Fundamentos de la utilización de película 337
Trabajo con objetos MovieClip 338
Control de la reproducción de clips de película 338
Creación de objetos MovieClip con ActionScript 341
Carga de un archivo SWF externo 344
Ejemplo de clip de película: RuntimeAssetsExplorer 346

Capítulo 17: Trabajo con interpolaciones de movimiento
Fundamentos de interpolaciones de movimiento 351
Copia de scripts de interpolación de movimiento en Flash 352
Incorporación de scripts de interpolación de movimiento 353
Descripción de la animación 354
Añadir filtros 357
Asociación de una interpolación de movimiento con sus objetos de visualización 359

Capítulo 18: Trabajo con cinemática inversa

Fundamentos de cinemática inversa 361
Información general sobre animación de esqueletos IK 362
Obtener información sobre un esqueleto IK 364
Creación de una instancia de la clase IKMover y limitación del movimiento 364
Desplazamiento de un esqueleto IK 365
Uso de muelles 366
Uso de eventos IK 366

Capítulo 19: Trabajo en tres dimensiones (3D)

Fundamentos de objetos de visualización 3D 368
Información sobre los objetos de visualización 3D en el motor de ejecución de Flash Player y AIR 369
Creación y movimiento de objetos de visualización 3D 371
Proyección de objetos 3D en una vista bidimensional 373
Ejemplo: Proyección en perspectiva 375
Transformaciones 3D complejas 378
Uso de triángulos para efectos 3D 381

Capítulo 20: Fundamentos de la utilización de texto

Capítulo 21: Uso de la clase TextField

Visualización de texto 391
Selección y manipulación de texto 395
Captura de entradas de texto 397
Restricción de entradas de texto 398
Formato de texto 399
Representación avanzada de texto 403
Trabajo con texto estático 405
Ejemplo de TextField: Formato de texto con estilo periodístico 407

Capítulo 22: Uso de Flash Text Engine

Creación y visualización de texto 416
Gestión de eventos en FTE 421
Formato de texto 424
Trabajo con fuentes 429
Control del texto 431
Ejemplo de Flash Text Engine: News Layout 437

Capítulo 23: Uso de Text Layout Framework

Información general sobre Text Layout Framework 446
Uso de Text Layout Framework 447
Estructuración del texto con TLF 452
Formato de texto con TLF 456
Importación y exportación de texto con TLF 457
Administración de contenedores de texto con TLF 458
Activación de las operaciones para deshacer, editar y selección de texto con TLF 459
Control de eventos con TLF 459
Posicionamiento de imágenes en el texto 460

Capítulo 24: Trabajo con sonido

Fundamentos de la utilización de sonido 461
Aspectos básicos de la arquitectura de sonido 462
Carga de archivos de sonido externos 464
Trabajo con sonidos incorporados 466
Trabajo con archivos de flujo de sonido 468
Trabajo con sonido generado dinámicamente 469
Reproducción de sonidos 471
Consideraciones de seguridad al cargar y reproducir sonidos 474
Control de desplazamiento y volumen de sonido 475
Trabajo con metadatos de sonido 477
Acceso a datos de sonido sin formato 477
Captura de entradas de sonido 481
Ejemplo de sonido: Podcast Player 487

Capítulo 25: Trabajo con vídeo

Fundamentos de la utilización de vídeo 494
Aspectos básicos de los formatos de vídeo 495
Aspectos básicos de la clase Video 499
Carga de archivos de vídeo 499
Control de la reproducción de vídeo 500
Reproducción de vídeo en modo de pantalla completa 502
Transmisión de archivos de vídeo 506
Aspectos básicos de los puntos de referencia 506
Escritura de métodos callback para metadatos y puntos de referencia 507
Uso de puntos de referencia y metadatos 513
Control de la actividad de NetStream 523
Temas avanzados para archivos FLV 526
Ejemplo de vídeo: Video Jukebox 528
Uso de la clase StageVideo para la presentación con aceleración por hardware 534

Capítulo 26: Trabajo con cámaras

Aspectos básicos de la clase Camera 543
Visualización del contenido de la cámara en pantalla 544
Diseño de la aplicación de cámara 544
Conexión a la cámara de un usuario 545
Comprobación de que las cámaras están instaladas 545
Detección de permisos para el acceso a una cámara 546
Aumento de la calidad de vídeo de la cámara 548
Supervisión del estado de la cámara 549

Capítulo 27: Uso de la administración de derechos digitales

Aspectos básicos del flujo de trabajo de contenido protegido 552
Eventos y miembros relacionados con DRM de la clase NetStream 558
Uso de la clase DRMStatusEvent 560
Uso de la clase DRMAuthenticateEvent 561
Uso de la clase DRMErrorEvent 565

Uso de la clase DRMManager 565
Uso de la clase DRMContentData 567
Actualización de Flash Player para admitir Adobe Access 567
Licencias fuera de banda 569
Soporte de dominios 570
Reproducción de contenido cifrado con soporte de dominios 571
Previsualización de licencias 571
Entrega de contenido 572
Open Source Media Framework 572

Capítulo 28: Cómo añadir contenido PDF en AIR
Detección de la capacidad de PDF 575
Carga de contenido PDF 576
Uso de scripts con el contenido PDF 576
Limitaciones conocidas del contenido PDF en AIR 578

Capítulo 29: Aspectos básicos de la interacción con el usuario
Captura de entradas del usuario 580
Administración de la selección 581
Detección de tipos de entrada 582

Capítulo 30: Entradas de teclado
Captura de entradas de teclado 584
Uso de la clase IME 586
Teclados virtuales 591

Capítulo 31: Entradas de ratón
Captura de entradas de ratón 599
Ejemplo de entrada de ratón: WordSearch 602

Capítulo 32: Entrada táctil, multitáctil y de gestos
Fundamentos de la entrada táctil 607
Detección con soporte táctil 609
Gestión de eventos táctiles 610
Tocar y arrastrar 613
Gestión de eventos de gestos 615
Solución de problemas 618

Capítulo 33: Copiar y pegar
Aspectos básicos de la operación de copiar y pegar 621
Lectura y escritura en el portapapeles del sistema 622
Operaciones de pegar y copiar HTML en AIR 622
Formatos de datos para Clipboard 624

Capítulo 34: Entrada de acelerómetro
Comprobación de la compatibilidad del acelerómetro 630
Detección de cambios del acelerómetro 631

Capítulo 35: Operación de arrastrar y colocar en AIR
Aspectos básicos de la operación de arrastrar y colocar en AIR 633
Compatibilidad con el gesto de arrastrar hacia fuera 635
Compatibilidad con el gesto de arrastrar hacia dentro 638
Operación de arrastrar y colocar en HTML 641
Arrastrar datos hacia fuera de un elemento en HTML 645
Arrastrar datos dentro de un elemento HTML 645
Ejemplo: Supresión del comportamiento predeterminado de arrastrar hacia dentro en HTML 646
Gestión de la colocación de archivos en entornos limitados de HTML ajenos a la aplicación 648
Colocación de promesas de archivo 649

Capítulo 36: Trabajo con menús
Menú: conceptos básicos 658
Creación de menús nativos (AIR) 665
Menús contextuales en HTML (AIR) 667
Visualización de menús nativos desplegables (AIR) 668
Gestión de eventos de menú 669
Ejemplo de menú nativo: Menú de aplicación y de ventana (AIR) 670

Capítulo 37: Iconos de la barra de tareas en AIR
Iconos de la barra de tareas 674
Iconos del Dock 675
Iconos de la bandeja del sistema 675
Botones e iconos de la barra de tareas de la ventana 677

Capítulo 38: Trabajo con el sistema de archivos
Uso de la clase FileReference 679
Uso de la API del sistema de archivos de AIR 694

Capítulo 39: Almacenamiento de datos locales
Objetos compartidos 731
Almacenamiento local cifrado 740

Capítulo 40: Trabajo con bases de datos SQL locales en AIR
Bases de datos SQL locales 744
Creación y modificación de una base de datos 749
Manipulación de los datos de una base de datos SQL 756
Uso de operaciones sincrónicas y asíncronas de base de datos 785
Uso del cifrado con bases de datos SQL 790
Estrategias para la utilización de bases de datos SQL 807

Capítulo 41: Trabajo con conjuntos de bytes
Lectura y escritura de un ByteArray 810
Ejemplo de ByteArray: Lectura de un archivo .zip 815

Capítulo 42: Fundamentos de redes y comunicación
Interfaces de red 823
Cambios de conectividad de la red 824
Registros del sistema de nombres de dominio (DNS) 827

Capítulo 43: Sockets
Sockets TCP 829
Sockets UDP (AIR) 840
Direcciones IPv6 842

Capítulo 44: Comunicaciones HTTP
Carga de datos externos 844
Peticiones de servicios web 853
Apertura de una URL en otra aplicación 861

Capítulo 45: Conexión con otras instancias de Flash Player y AIR
Información sobre la clase LocalConnection 863
Envío de mensajes entre dos aplicaciones 865
Conexión al contenido en diferentes dominios y a otras aplicaciones de AIR 867

Capítulo 46: Comunicación con procesos nativos en AIR
Información general sobre las comunicaciones de proceso nativo 870
Inicio y cierre de un proceso nativo 871
Comunicación con un proceso nativo 872
Seguridad en la comunicación del proceso nativo 873

Capítulo 47: Uso de la API externa
Fundamentos de la utilización de la API externa 875
Requisitos y ventajas de la API externa 878
Uso de la clase ExternalInterface 879
Ejemplo de API externa: Comunicación entre ActionScript y JavaScript en un navegador web 883

Capítulo 48: Validación de la firma XML en AIR
Aspectos básicos de la validación de firmas XML 889
Firmas XML 894
Implementación de la interfaz IURIDereferencer 896

Capítulo 49: Entorno del sistema del cliente
Fundamentos del entorno del sistema del cliente 904
Uso de la clase System 905
Uso de la clase Capabilities 906
Ejemplo de Capabilities: Detección de las características del sistema 907

Capítulo 50: Cierre e invocación de una aplicación de AIR
Invocación de aplicaciones 911
Captura de argumentos de la línea de comandos 913
Invocación de una aplicación de AIR en el inicio de sesión de usuario 916
Invocación de una aplicación de AIR desde el navegador 917
Cierre de una aplicación 919

Capítulo 51: Trabajo con información sobre el motor de ejecución de AIR y el sistema operativo
Gestión de asociaciones con archivos 921
Obtención de la versión y el nivel de revisión del motor de ejecución 922

Detección de las capacidades de AIR 922
Seguimiento de la presencia de usuarios 923

Capítulo 52: Trabajo con ventanas nativas de AIR
Aspectos básicos de las ventanas nativas en AIR 924
Creación de ventanas 932
Gestión de ventanas 941
Detección de eventos de ventanas 951
Visualización de ventanas a pantalla completa 952

Capítulo 53: Pantallas en AIR
Aspectos básicos de las pantallas en AIR 954
Enumeración de las pantallas 955

Capítulo 54: Impresión
Fundamentos de impresión 958
Impresión de una página 959
Interfaz de impresión del sistema y tareas del motor de ejecución de Flash 960
Configuración del tamaño, la escala y la orientación 963
Técnicas de impresión avanzada 965
Ejemplo de impresión: Impresión de varias páginas 967
Ejemplo de impresión: Ajuste de escala, recorte y respuesta 970
Ejemplo de impresión: Configuración de página y opciones de impresión 971

Capítulo 55: Localización geográfica
Detección de cambios de geolocalización 974

Capítulo 56: Internacionalización de aplicaciones
Conceptos básicos de internacionalización de aplicaciones 978
Información general sobre el paquete flash.globalization 979
Determinación de la configuración regional 981
Formato de números 982
Formato de valores de divisa 984
Formato de fechas y horas 987
Ordenación y comparación de cadenas 988
Conversión tipográfica 990
Ejemplo: Internacionalización de una aplicación de bolsa 991

Capítulo 57: Localización de aplicaciones
Elección de una configuración regional 996
Localización de contenido de Flex 997
Localización de contenido de Flash 997
Localización de aplicaciones de AIR 997
Localización de fechas, horas y monedas 998

Capítulo 58: Entorno HTML
Información general sobre el entorno HTML 1000
AIR y WebKit 1003

Capítulo 59: Programación con HTML y JavaScript en AIR

Información sobre la clase HTMLLoader 1019
Cómo evitar errores de JavaScript relacionados con la seguridad 1021
Acceso a las clases de API de AIR desde JavaScript 1026
Información sobre URL en AIR 1028
Disponibilidad de objetos ActionScript en JavaScript 1029
Acceso a objetos JavaScript y DOM HTML desde ActionScript 1030
Incorporación de contenido SWF en HTML 1032
Uso de las bibliotecas de ActionScript en una página HTML 1034
Conversión de los objetos Date y RegExp 1036
Manipulación de una hoja de estilo HTML de ActionScript 1036
Uso de scripts entre contenidos en diferentes entornos limitados de seguridad 1038

Capítulo 60: Uso de scripts en el contenedor HTML de AIR

Propiedades de visualización de objetos HTMLLoader 1043
Desplazamiento de contenido HTML 1046
Acceso a la lista del historial HTML 1047
Configuración del agente de usuario que se utiliza al cargar contenido HTML 1048
Configuración de la codificación de caracteres para utilizar con el contenido HTML 1048
Definición de interfaces de usuario del navegador para el contenido HTML 1049
Creación de subclases de la clase HTMLLoader 1060

Capítulo 61: Gestión de eventos relacionados con HTML en AIR

Eventos HTMLLoader 1062
Gestión de eventos DOM con ActionScript 1063
Respuesta a excepciones en JavaScript sin capturar 1063
Gestión de eventos del motor de ejecución con JavaScript 1065

Capítulo 62: Visualización de contenido HTML en aplicaciones móviles

Objetos StageWebView 1068
Contenido 1068
Eventos de navegación 1070
Historial 1071
Selección 1072
Captura de mapas de bits 1074

Capítulo 63: Uso de programas de trabajo para simultaneidad

Descripción de los programas de trabajo y la simultaneidad 1076
Creación y administración de programas de trabajo 1077
Comunicación entre programas de trabajo 1080

Capítulo 64: Seguridad

Información general sobre la seguridad de la plataforma Flash 1085
Entornos limitados de seguridad 1087
Controles de permiso 1092
Restricción de las API de red 1100
Seguridad del modo de pantalla completa 1103
Seguridad del modo de pantalla completa 1104

Carga de contenido 1105
Reutilización de scripts 1108
Acceso a medios cargados como datos 1111
Carga de datos 1114
Carga de contenido incorporado de archivos SWF importados en un dominio de seguridad 1117
Trabajo con contenido heredado 1117
Configuración de permisos de LocalConnection 1118
Control del acceso URL saliente 1119
Objetos compartidos 1120
Acceso a la cámara, el micrófono, el portapapeles, el ratón y el teclado 1122
Seguridad en AIR 1122

Capítulo 65: Cómo utilizar ejemplos de ActionScript
Tipos de ejemplos 1145
Ejecución de ejemplos de ActionScript 3.0 en Flash Professional 1147
Ejecución de ejemplos de ActionScript 3.0 en Flash Builder 1148
Ejecución de ejemplos de ActionScript 3.0 en dispositivos móviles 1150

Capítulo 66: Compatibilidad de SQL en bases de datos locales
Sintaxis SQL admitida 1155
Compatibilidad de tipos de datos 1177

Capítulo 67: Mensajes detallados de errores SQL, identificadores y argumentos

Capítulo 68: AGAL (Adobe Graphics Assembly Language, Lenguaje de ensamblaje de gráficos de Adobe)
Formato de código de bytes de AGAL 1187

Capítulo 1: Trabajo con fechas y horas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede ser que el tiempo no sea lo más importante, pero suele ser un factor clave en las aplicaciones de software. ActionScript 3.0 proporciona formas eficaces de administrar fechas de calendario, horas e intervalos de tiempo. Dos clases principales proporcionan la mayor parte de esta funcionalidad de tiempo: la cases Date y Timer del paquete flash.utils.

Las fechas y las horas son un tipo de información utilizado frecuentemente en programas escritos en ActionScript. Por ejemplo, puede ser necesario averiguar el día de la semana o medir cuánto tiempo pasa un usuario en una pantalla determinada, entre otras muchas cosas. En ActionScript se puede utilizar la clase Date para representar un momento puntual en el tiempo, incluida la información de fecha y hora. En una instancia de Date hay valores para las unidades individuales de fecha y hora, incluidas año, mes, fecha, día de la semana, hora, minutos, segundos, milisegundos y zona horaria. Para usos más avanzados, ActionScript también incluye la clase Timer, que se puede utilizar para realizar acciones después de un retardo o a intervalos repetidos.

Más temas de ayuda

Date

flash.utils.Timer

Administración de fechas de calendario y horas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todas las funciones de administración de fechas de calendario y horas de ActionScript 3.0 se concentran en la clase Date de nivel superior. La clase Date contiene métodos y propiedades que permiten gestionar fechas y horas en la hora universal (UTC) o en la hora local específica de una zona horaria. La hora universal (UTC) es una definición de hora estándar que básicamente coincide con la hora del meridiano de Greenwich (GMT).

Creación de objetos Date

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método constructor de la clase Date es uno de los métodos constructores de clase principal más versátiles. Se puede llamar de cuatro formas distintas.

En primer lugar, si no se proporcionan parámetros, el constructor `Date()` devuelve un objeto Date que contiene la fecha y hora actuales, en la hora local correspondiente a la zona horaria. A continuación se muestra un ejemplo:

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

En segundo lugar, si se proporciona un solo parámetro numérico, el constructor `Date()` trata dicho parámetro como el número de milisegundos transcurridos desde el 1 de enero de 1970 y devuelve un objeto Date correspondiente. Hay que tener en cuenta que el valor de milisegundos que se pasa se trata como milisegundos desde el 1 de enero de 1970, en UTC. Sin embargo, el objeto Date muestra los valores en la zona horaria local, a menos que se utilicen métodos específicos de UTC para recuperarlos y mostrarlos. Si se crea un nuevo objeto Date con un solo parámetro de milisegundos, hay tener en cuenta la diferencia de zona horaria entre la hora local y la hora universal UTC. Las siguientes sentencias crean un objeto Date establecido en la medianoche del día 1 de enero de 1970, en UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

En tercer lugar, se pueden pasar varios parámetros numéricos al constructor `Date()`. Estos parámetros se tratan como el año, mes, día, hora, minuto, segundo y milisegundo, respectivamente, y se devuelve un objeto Date correspondiente. Se supone que estos parámetros de entrada están en hora local y no en UTC. Las siguientes sentencias obtienen un objeto Date establecido en la medianoche del inicio del día 1 de enero de 2000, en hora local:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

En cuarto lugar, se puede pasar un solo parámetro de cadena al constructor `Date()`. Este intentará analizar dicha cadena en componentes de fecha u hora, y devolver a continuación un objeto Date correspondiente. Si utiliza este enfoque, es una buena idea incluir el constructor `Date()` en un bloque `try..catch` para localizar cualquier error de análisis. El constructor `Date()` acepta una serie de formatos de cadena diferentes (que se incluyen en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash). La siguiente sentencia inicializa un nuevo objeto Date con un valor de cadena:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Si el constructor `Date()` no puede analizar correctamente el parámetro de cadena, no emitirá una excepción. Sin embargo, el objeto Date resultante contendrá un valor de fecha no válido.

Obtención de valores de unidad de tiempo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Es posible extraer los valores de diversas unidades de tiempo de un objeto Date a través de las propiedades y métodos de la clase Date. Cada una de las siguientes propiedades asigna el valor de una unidad de tiempo en el objeto Date:

- La propiedad `fullYear`
- La propiedad `month`, que tiene un formato numérico con un valor entre 0 (enero) y 11 (diciembre)
- La propiedad `date`, que es el número de calendario del día del mes, en el rango de 1 a 31
- La propiedad `day`, que es el día de la semana en formato numérico, siendo 0 el valor correspondiente al domingo
- La propiedad `hours`, en el rango de 0 a 23
- La propiedad `minutes`
- La propiedad `seconds`
- La propiedad `milliseconds`

  De hecho, la clase Date proporciona varias formas de obtener cada uno de estos valores. Por ejemplo, se puede obtener el valor de mes de un objeto Date de cuatro formas distintas:

- La propiedad `month`
- El método `getMonth()`

- La propiedad `monthUTC`
- El método `getMonthUTC()`

  Los cuatro métodos son igualmente eficaces, de forma que puede elegirse el que mejor convenga a cada aplicación.

  Todas las propiedades anteriormente enumeradas representan componentes del valor de fecha total. Por ejemplo, el valor de la propiedad milliseconds nunca será mayor que 999, ya que cuando llega a 1000, el valor de los segundos aumenta en 1 y el valor de la propiedad milliseconds se restablece en 0.

  Si se desea obtener el valor del objeto Date en milisegundos transcurridos desde el 1 de enero de 1970 (UTC), se puede utilizar el método `getTime()`. El método opuesto, `setTime()`, permite cambiar el valor de un objeto Date existente utilizando los milisegundos transcurridos desde el 1 de enero de 1970 (UTC).

Operaciones aritméticas de fecha y hora

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Date permite realizar sumas y restas en fechas y horas. Los valores de fecha se almacenan internamente como milisegundos, de forma que es necesario convertir los demás valores a milisegundos para poder realizar sumas o restas en objetos Date.

Si la aplicación va a realizar muchas operaciones aritméticas de fecha y hora, quizás resulte útil crear constantes que conserven valores comunes de unidad de tiempo en milisegundos, como se muestra a continuación:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

Ahora resulta sencillo realizar operaciones aritméticas de fecha con unidades de tiempo estándar. El código siguiente establece un valor de fecha de una hora a partir de la hora actual, mediante los métodos `getTime()` y `setTime()`:

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Otra forma de establecer un valor de fecha consiste en crear un nuevo objeto Date con un solo parámetro de milisegundos. Por ejemplo, el siguiente código añade 30 días a una fecha para calcular otra:

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

A continuación, se multiplica por 30 la constante `millisecondsPerDay` para representar un tiempo de 30 días y el resultado se añade al valor `invoiceDate` y se utiliza para establecer el valor `dueDate`.

Conversión entre zonas horarias

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las operaciones aritméticas de fecha y hora son útiles para convertir fechas de una zona horaria a otra. Esta conversión se realiza a través del método `getTimezoneOffset()`, que devuelve el valor en minutos de diferencia entre la zona horaria del objeto Date y la hora universal UTC. Devuelve un valor en minutos porque no todas las zonas horarias se establecen mediante incrementos de horas completas, sino que en algunos casos existen diferencias de media hora entre zonas vecinas.

En el siguiente ejemplo se utiliza el desplazamiento de zona horaria para convertir una fecha de la hora local a la hora universal UTC. Para realizar la conversión, primero se calcula el valor de zona horaria en milisegundos y luego se ajusta el valor de Date en dicha cantidad:

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

Control de intervalos de tiempo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se desarrollan aplicaciones con Adobe Flash CS4 Professional, se tiene acceso a la línea de tiempo, que proporciona una progresión uniforme, fotograma a fotograma, a través de la aplicación. Sin embargo, en proyectos solo de ActionScript, hay que confiar en otros mecanismos de tiempo.

Bucles frente a temporizadores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En algunos lenguajes de programación, es necesario crear esquemas temporales propios mediante sentencias de bucle como `for` o `do..while`.

Las sentencias de bucle suelen ejecutarse con la máxima rapidez permitida en el equipo, lo que significa que la aplicación se ejecuta más rápidamente en unos equipos que en otros. Si una aplicación debe tener un intervalo de tiempo coherente, es necesario asociarla a un calendario o reloj real. Muchas aplicaciones como juegos, animaciones y controladores de tiempo real necesitan mecanismos de tictac regulares que sean coherentes de un equipo a otro.

La clase Timer de ActionScript 3.0 proporciona una solución eficaz. A través del modelo de eventos de ActionScript 3.0, la clase Timer distribuye eventos del temporizador cuando se alcanza un intervalo de tiempo especificado.

La clase Timer

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La forma preferida de gestionar funciones de tiempo en ActionScript 3.0 consiste en utilizar la clase Timer (flash.utils.Timer) para distribuir eventos cuando se alcanza un intervalo.

Para iniciar un temporizador, primero es necesario crear una instancia de la clase Timer e indicarle la frecuencia con la que debe generar un evento del temporizador y la cantidad de veces que debe hacerlo antes de detenerse.

Por ejemplo, el código siguiente crea una instancia de Timer que distribuye un evento cada segundo y se prolonga durante 60 segundos:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

El objeto Timer distribuye un objeto TimerEvent cada vez que se alcanza el intervalo especificado. El tipo de evento de un objeto TimerEvent es `timer` (definido por la constante `TimerEvent.TIMER`). Un objeto TimerEvent contiene las mismas propiedades que un objeto Event estándar.

Si la instancia de Timer se establece en un número fijo de intervalos, también distribuirá un evento `timerComplete` (definido por la constante `TimerEvent.TIMER_COMPLETE`) cuando alcance el intervalo final.

A continuación se muestra una aplicación de ejemplo donde se utiliza la clase Timer:

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

Al crear la clase ShortTimer, se crea una instancia de Timer que hará tictac una vez por segundo durante cinco segundos. Posteriormente agrega dos detectores al temporizador: uno que detecta todos los tictac y otro que detecta el evento `timerComplete` .

A continuación, inicia el tictac del temporizador y, a partir de ese punto, el método `onTick()` se ejecuta en intervalos de un segundo.

El método `onTick()` simplemente muestra el recuento de tictac actual. Después de cinco segundos, se ejecuta el método `onTimerComplete()`, que indica que se ha acabado el tiempo.

Cuando se ejecuta este ejemplo, aparecen las siguientes líneas en la consola o ventana de traza a una velocidad de una línea por segundo:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

Funciones de tiempo en el paquete flash.utils

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye una serie de funciones de tiempo similares a las que estaban disponibles en ActionScript 2.0. Estas funciones se proporcionan como funciones de nivel de paquete en flash.utils y funcionan del mismo modo que en ActionScript 2.0.

Función	Descripción
`clearInterval(id:uint):void`	Cancela una llamada a `setInterval()` especificada.
`clearTimeout(id:uint):void`	Cancela una llamada `setTimeout()` especificada.
`getTimer():int`	Devuelve el número de milisegundos transcurridos desde que se inició Adobe® Flash® Player o Adobe® AIR™.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Ejecuta una función en un intervalo especificado (en milisegundos).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Ejecuta una función especificada tras una demora especificada (en milisegundos).

Estas funciones se conservan en ActionScript 3.0 para permitir la compatibilidad con versiones anteriores. Adobe no recomienda utilizarlas en nuevas aplicaciones de ActionScript 3.0. En general, resulta más sencillo y eficaz utilizar la clase Timer en las aplicaciones.

Ejemplo de fecha y hora: Un sencillo reloj analógico

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En este ejemplo de un sencillo reloj analógico se ilustran estos dos conceptos de fecha y hora:

- Obtención de la fecha y hora actuales, y extracción de valores para las horas, minutos y segundos
- Uso de un objeto Timer para establecer el ritmo de una aplicación

  Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación SimpleClock se encuentran en la carpeta Samples/SimpleClock. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SimpleClockApp.mxml o SimpleClockApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/simpleclock/SimpleClock.as	El archivo de la aplicación principal.
com/example/programmingas3/simpleclock/AnalogClockFace.as	Dibuja una esfera de reloj redonda y las manecillas de hora, minutos y segundos, en función de la hora.

Definición de la clase SimpleClock

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo del reloj es sencillo, pero es recomendable organizar bien incluso las aplicaciones sencillas para poder ampliarlas fácilmente en el futuro. En ese sentido, la aplicación SimpleClock utiliza la clase SimpleClock para gestionar las tareas de inicio y mantenimiento de hora, y luego utiliza otra clase denominada AnalogClockFace para mostrar realmente la hora.

A continuación se muestra el código que define e inicializa la clase SimpleClock (hay que tener en cuenta que, en la versión de Flash, SimpleClock amplía la clase Sprite):

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

La clase tiene dos propiedades importantes:

- La propiedad `face`, que es una instancia de la clase AnalogClockFace
- La propiedad `ticker`, que es una instancia de la clase Timer

  La clase SimpleClock utiliza un constructor predeterminado. El método `initClock()` realiza el trabajo de configuración real, pues crea la esfera del reloj e inicia el tictac de la instancia de Timer.

Creación de la esfera del reloj

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las siguientes líneas del código de SimpleClock crean la esfera del reloj que se utiliza para mostrar la hora:

```
/**
 * Sets up a SimpleClock instance.
 */
public function initClock(faceSize:Number = 200)
{
    // creates the clock face and adds it to the display list
    face = new AnalogClockFace(Math.max(20, faceSize));
    face.init();
    addChild(face);

    // draws the initial clock display
    face.draw();
```

El tamaño de la esfera puede pasarse al método `initClock()`. Si no se pasa el valor `faceSize`, se utiliza un tamaño predeterminado de 200 píxeles.

A continuación, la aplicación inicializa la esfera y la añade a la lista de visualización a través del método `addChild()` heredado de la clase DisplayObjectContainer. Luego llama al método `AnalogClockFace.draw()` para mostrar la esfera del reloj una vez, con la hora actual.

Inicio del temporizador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Después de crear la esfera del reloj, el método `initClock()` configura un temporizador:

```
    // creates a Timer that fires an event once per second
    ticker = new Timer(1000);

    // designates the onTick() method to handle Timer events
    ticker.addEventListener(TimerEvent.TIMER, onTick);

    // starts the clock ticking
    ticker.start();
```

En primer lugar, este método crea una instancia de Timer que distribuirá un evento una vez por segundo (cada 1000 milisegundos). Como no se pasa ningún parámetro `repeatCount` al constructor `Timer()`, el objeto Timer se repetirá indefinidamente.

El método `SimpleClock.onTick()` se ejecutará una vez por segundo cuando se reciba el evento `timer`:

```
public function onTick(event:TimerEvent):void
{
    // updates the clock display
    face.draw();
}
```

El método `AnalogClockFace.draw()` simplemente dibuja la esfera del reloj y las manecillas.

Visualización de la hora actual

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La mayoría del código de la clase AnalogClockFace se utiliza para configurar los elementos que se visualizan en la esfera del reloj. Cuando se inicializa AnalogClockFace, dibuja un contorno circular, coloca una etiqueta de texto numérico en cada marca de hora y luego crea tres objetos Shape: uno para la manecilla de las horas, otro para la de los minutos y un tercero para la de los segundos.

Cuando se ejecuta la aplicación SimpleClock, llama al método `AnalogClockFace.draw()` cada segundo, del siguiente modo:

```
/**
 * Called by the parent container when the display is being drawn.
 */
public override function draw():void
{
    // stores the current date and time in an instance variable
    currentTime = new Date();
    showTime(currentTime);
}
```

Este método guarda la hora actual en una variable, de forma que la hora no puede cambiar mientras se dibujan las manecillas del reloj. Luego llama al método `showTime()` para mostrar las manecillas, como se muestra a continuación:

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

En primer lugar, este método extrae los valores de las horas, minutos y segundos de la hora actual. Luego utiliza estos valores para calcular el ángulo de cada manecilla. Como la manecilla de los segundos realiza una rotación completa en 60 segundos, gira 6 grados cada segundo (360/60). La manecilla de los minutos gira en la misma medida cada minuto.

La manecilla de las horas se actualiza también cada minuto, de forma que va progresando a medida que pasan los minutos. Gira 30 grados cada hora (360/12), pero también gira medio grado cada minuto (30 grados divididos entre 60 minutos).

Capítulo 2: Trabajo con cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase String contiene métodos que le permiten trabajar con cadenas de texto. Las cadenas son importantes para trabajar con muchos de los objetos. Los métodos descritos en este capítulo son útiles para trabajar con cadenas utilizadas en objetos como TextField, StaticText, XML, ContextMenu y FileReference.

Las cadenas son secuencias de caracteres. ActionScript 3.0 admite caracteres ASCII y Unicode.

Más temas de ayuda

String

RegExp

parseFloat()

parseInt()

Fundamentos de la utilización de cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En jerga de programación, una cadena es un valor de texto: una secuencia de letras, números u otros caracteres agrupados en un solo valor. Por ejemplo, esta línea de código crea una variable con el tipo de datos String y asigna un valor de literal de cadena a esa variable:

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

Como se ilustra en este ejemplo, en ActionScript se puede denotar un valor de cadena escribiendo texto entre comillas dobles o simples. A continuación se muestran varios ejemplos de cadenas:

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Siempre que se manipule un fragmento de texto en ActionScript, se trabaja con un valor de cadena. La clase String de ActionScript es el tipo de datos que se utiliza para trabajar con valores de texto. Se suelen utilizar instancias de String para propiedades, parámetros de métodos, etc. en muchas otras clases de ActionScript.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes utilizados:

ASCII Sistema para representar caracteres de texto y símbolos en programas informáticos. El sistema ASCII incluye el alfabeto inglés de 26 letras y un conjunto limitado de caracteres adicionales.

Carácter Unidad más pequeña de los datos de texto (una letra o un símbolo).

Concatenación Unión de varios valores de cadena añadiendo uno a continuación del otro para crear un nuevo valor de cadena.

Cadena vacía Cadena que no contiene texto, espacio en blanco ni otros caracteres; se escribe como `""`. Un valor de cadena vacía no es lo mismo que una variable de tipo String con valor NULL. Una variable de tipo String con valor NULL es una variable que no tiene una instancia de String asignada, mientras que una cadena vacía tiene una instancia con un valor que no contiene ningún carácter.

String Valor de texto (secuencia de caracteres).

Literal de cadena (o "cadena literal") Literal de cadena (o "cadena literal"): valor de cadena escrito explícitamente en el código como un valor de texto entre comillas dobles o simples.

Subcadena Cadena que forma parte de otra cadena.

Unicode Sistema estándar para representar caracteres de texto y símbolos en programas informáticos. El sistema Unicode permite utilizar cualquier carácter de un sistema de escritura.

Creación de cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase String se utiliza para representar datos de cadena (textuales) en ActionScript 3.0. Las cadenas de ActionScript admiten caracteres tanto ASCII como Unicode. La manera más sencilla de crear una cadena es utilizar un literal de cadena. Para declarar un literal de cadena, hay que utilizar comilla dobles rectas (`"`) o comillas simples (`'`). Por ejemplo, las dos cadenas siguientes son equivalentes:

```
var str1:String = "hello";
var str2:String = 'hello';
```

También se puede declarar una cadena utilizando el operador `new`, de la manera siguiente:

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

Las dos cadenas siguientes son equivalentes:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Para utilizar comillas simples (`'`) dentro de un literal de cadena definido con comillas simples (`'`) hay que utilizar el carácter de escape barra diagonal inversa (`\`). De manera similar, para utilizar comillas dobles (`"`) dentro de un literal de cadena definido con comillas dobles (`"`) hay que utilizar el carácter de escape barra diagonal inversa (`\`). Las dos cadenas siguientes son equivalentes:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Se puede elegir utilizar comillas simples o comillas dobles en función de las comillas simples o dobles que existan en un literal de cadena, como se muestra a continuación:

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Hay que tener en cuenta que ActionScript distingue entre una comilla simple recta (`'`) y una comilla simple izquierda o derecha (`' o ' `). Lo mismo ocurre para las comillas dobles. Hay que utilizar comillas rectas para delimitar los literales de cadena. Al pegar texto de otro origen en ActionScript hay que asegurarse de utilizar los caracteres correctos.

Como se muestra en la tabla siguiente, se puede utilizar el carácter de escape de barra diagonal inversa (`\`) para definir otros caracteres en los literales de cadena:

Secuencia de escape	Carácter
`\b`	Retroceso
`\f`	Salto de página
`\n`	Nueva línea
`\r`	Retorno de carro
`\t`	Tabulador
`\u`***nnnn***	Carácter Unicode con el código de carácter especificado por el número hexadecimal *nnnn*; por ejemplo, `\u263a` es el carácter smiley.
`\\x`***nn***	Carácter ASCII con el código de carácter especificado por el número hexadecimal *nn*.
`\'`	Comilla simple
`\"`	Comilla doble
`\\`	Barra diagonal inversa simple

La propiedad length

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cada cadena tiene una propiedad `length` que es igual al número de caracteres de la cadena:

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

Tanto la cadena vacía como la cadena NULL tienen longitud 0, como se muestra en el siguiente ejemplo:

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

Trabajo con caracteres en cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cada carácter de una cadena tiene una posición de índice en la cadena (un entero). La posición de índice del primer carácter es 0. Por ejemplo, en la siguiente cadena, el carácter `y` está en la posición 0 y el carácter `w` en la posición 5:

```
"yellow"
```

Se pueden examinar caracteres individuales en diversas posiciones de una cadena mediante los métodos `charAt()` y `charCodeAt()`, como en este ejemplo:

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

Cuando se ejecuta este código, se produce el siguiente resultado:

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

También se pueden utilizar códigos de caracteres para definir una cadena con el método `fromCharCode()`, como se muestra en el siguiente ejemplo:

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

Comparación de cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar los siguientes operadores para comparar cadenas: `<`, `<=`, `!=`, `==`, `=>` y `>`. Estos operadores se pueden utilizar con sentencias condicionales, como `if` y `while`, como se indica en el siguiente ejemplo:

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

Al usar estos operadores con cadenas, ActionScript considera el valor del código de carácter de cada carácter de la cadena, comparando los caracteres de izquierda a derecha, como se muestra a continuación:

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Los operadores `==` y `!=` se utilizan para comparar cadenas entre sí y para comparar cadenas con otros tipos de objetos, como se muestra en el siguiente ejemplo:

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

Obtención de representaciones de cadena de otros objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede obtener una representación de cadena para cualquier tipo de objeto. Todos los objetos tienen un método `toString()` para este fin:

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

Al utilizar el operador de concatenación `+` con una combinación de objetos String y objetos que no son cadenas, no es necesario utilizar el método `toString()`. Para ver más detalles sobre la concatenación, consulte la siguiente sección.

La función global `String()` devuelve el mismo valor para un objeto determinado como el valor devuelto por el objeto llamando al método `toString()`.

Concatenación de cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La concatenación de cadenas consiste en la unión secuencial de dos cadenas en una sola. Por ejemplo, se puede utilizar el operador suma `+` para concatenar dos cadenas:

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

También se puede utilizar el operador `+=` para producir el mismo resultado, como se indica en el siguiente ejemplo:

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Además, la clase String incluye un método `concat()`, que se puede utilizar como se muestra a continuación:

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Si se utiliza el operador `+` (o el operador `+=`) con un objeto String y un objeto que *no* es una cadena, ActionScript convierte automáticamente el objeto que no es una cadena en un objeto String para evaluar la expresión, como se indica en este ejemplo:

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

No obstante, se pueden utilizar paréntesis para agrupar con el fin de proporcionar contexto para el operador `+`, como se indica en el siguiente ejemplo:

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

Búsqueda de subcadenas y patrones en cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las subcadenas son caracteres secuenciales dentro de una cadena. Por ejemplo, la cadena `"abc"` tiene las siguientes subcadenas: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. ActionScript dispone de métodos para buscar subcadenas de una cadena.

En ActionScript los patrones se definen mediante cadenas o expresiones regulares. Por ejemplo, la siguiente expresión regular define un patrón específico que consiste en las letras A, B y C seguidas de un dígito (las barras diagonales son delimitadores de expresiones regulares):

```
/ABC\d/
```

ActionScript incluye métodos para buscar patrones en cadenas y para reemplazar las coincidencias por subcadenas. Estos métodos se describen en las secciones siguientes.

Las expresiones regulares permiten definir patrones complejos. Para obtener más información, consulte "Uso de expresiones regulares" en la página 77.

Búsqueda de una subcadena por posición de caracteres

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `substr()` y `substring()` son similares. Los dos métodos devuelven una subcadena de una cadena. Ambos utilizan dos parámetros. En cada uno de estos métodos, el primer parámetro es la posición del carácter inicial de la cadena en cuestión. No obstante, en el método `substr()`, el segundo parámetro es la *longitud* de la subcadena que debe devolverse, mientras que en el método `substring()`, el segundo parámetro es la posición del carácter *final* de la subcadena (que no se incluye en la cadena devuelta). Este ejemplo muestra la diferencia entre estos dos métodos:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

El método `slice()` funciona de forma similar al método `substring()`. Cuando se le facilitan como parámetros dos enteros no negativos, funciona exactamente de la misma forma. No obstante, el método `slice()` admite enteros negativos como parámetros, en cuyo caso la posición del carácter se mide desde el final de la cadena, como se indica en el siguiente ejemplo:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

Se pueden combinar enteros positivos y negativos como parámetros del método `slice()`.

Búsqueda de la posición de carácter de una subcadena coincidente

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar los métodos `indexOf()` y `lastIndexOf()` para localizar subcadenas coincidentes dentro de una cadena, como se muestra en el siguiente ejemplo.

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Hay que tener en cuenta que el método `indexOf()` distingue mayúsculas de minúsculas.

Se puede especificar un segundo parámetro para indicar la posición del índice en la cadena desde la que se debe iniciar la búsqueda, de la manera siguiente:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

El método `lastIndexOf()` localiza la última instancia de una subcadena en la cadena:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Si se incluye un segundo parámetro con el método `lastIndexOf()`, la búsqueda se realiza desde esa posición de índice en la cadena hacia atrás (de derecha a izquierda):

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

Creación de un conjunto de subcadenas segmentadas por un delimitador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar el método `split()` para crear un conjunto de subcadenas, que se divide en función de un delimitador. Por ejemplo, se puede segmentar en varias cadenas una cadena delimitada por comas o tabulaciones.

En el siguiente ejemplo se muestra la manera de dividir un conjunto en subcadenas con el carácter ampersand (&) como delimitador:

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

El segundo parámetro del método `split()`, que es opcional, define el tamaño máximo del conjunto devuelto.

También se puede utilizar una expresión regular como carácter delimitador:

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Para obtener más información, consulte "Uso de expresiones regulares" en la página 77 y Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Búsqueda de patrones en cadenas y sustitución de subcadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase String incluye los siguientes métodos para trabajar con patrones en cadenas:

- Los métodos `match()` y `search()` se utilizan para localizar subcadenas que coincidan con un patrón.
- El método `replace()` permite buscar subcadenas que coincidan con un patrón y sustituirlas por una subcadena especificada.

Estos métodos se describen en las secciones siguientes.

Se pueden utilizar cadenas o expresiones regulares para definir los patrones utilizados en estos métodos. Para obtener más información sobre las expresiones regulares, consulte "Uso de expresiones regulares" en la página 77.

Búsqueda de subcadenas coincidentes

El método `search()` devuelve la posición del índice de la primera subcadena que coincide con un patrón determinado, como se indica en este ejemplo:

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

También se pueden utilizar expresiones regulares para definir el patrón que se debe buscar, como se indica en este ejemplo:

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

La salida del método `trace()` es 0 porque el primer carácter de la cadena es la posición de índice 0. La búsqueda no distingue mayúsculas de minúsculas porque está establecido el indicador `i` en la expresión regular.

El método `search()` busca una sola coincidencia y devuelve su posición de índice inicial, aunque el indicador `g` (global) esté establecido en la expresión regular.

En el siguiente ejemplo se muestra una expresión regular más compleja, que coincide con una cadena entre comillas dobles:

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

El método `match()` funciona de manera similar. Busca una subcadena coincidente. Sin embargo, al utilizar el indicador global en un patrón de expresión regular, como en el siguiente ejemplo, `match()` devuelve un conjunto de subcadenas coincidentes:

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

El conjunto `results` se establece en:

```
["bob@example.com","omar@example.org"]
```

Para obtener más información sobre las expresiones regulares, consulte "Uso de expresiones regulares" en la página 77.

Sustitución de subcadenas coincidentes

Se puede utilizar el método `replace()` para buscar un patrón específico en una cadena y sustituir las coincidencias por la cadena de sustitución especificada, como se indica en el siguiente ejemplo:

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch")); //sche sells seaschells by the seaschore.
```

En este ejemplo se puede observar que las cadenas coincidentes no distinguen mayúsculas de minúsculas porque está establecido el indicador `i` (`ignoreCase`) en la expresión regular y se sustituyen todas las coincidencias porque está establecido el indicador `g` (`global`). Para obtener más información, consulte "Uso de expresiones regulares" en la página 77.

Se pueden incluir los siguientes códigos de sustitución `$`en la cadena de sustitución. El texto de sustitución mostrado en la tabla siguiente se inserta en lugar del código de sustitución `$`:

Código $	Texto de sustitución
`$$`	$
`$&`	La subcadena coincidente.
`$` `	La parte de la cadena que precede a la subcadena coincidente. Este código utiliza el carácter de comilla simple recta izquierda (`), no la comilla simple recta (') ni la comilla simple curva izquierda (').
`$'`	La parte de la cadena que sigue a la subcadena coincidente. Este código utiliza la comilla simple recta (').
$n	El *n*-ésimo grupo entre paréntesis coincidente capturado, donde n es un único dígito, 1-9, y $n no va seguido de un dígito decimal.
$nn	La *nn*-ésima coincidencia de grupo entre paréntesis capturada, donde *nn* es un número decimal de dos dígitos, 01–99. Si la *nn*-ésima captura no está definida, el texto de sustitución será una cadena vacía.

Por ejemplo, a continuación se muestra el uso de los códigos de sustitución `$2` y `$1`, que representan el primer y el segundo grupo coincidente captado:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

También se puede utilizar una función como segundo parámetro del método `replace()`. El texto coincidente se sustituye por el valor devuelto por la función.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Si se utiliza una función como segundo parámetro del método `replace()`, se pasan los siguientes argumentos a la función:

- La parte coincidente de la cadena.
- Las coincidencias de grupos de paréntesis de captura. El número de argumentos pasados de esta forma varía en función del número de grupos entre paréntesis coincidentes. Se puede determinar el número de grupos entre paréntesis coincidentes comprobando `arguments.length - 3` dentro del código de la función.
- La posición de índice en la que comienza la coincidencia en la cadena.
- La cadena completa.

Conversión de cadenas de mayúsculas a minúsculas y viceversa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Como se indica en el siguiente ejemplo, los métodos `toLowerCase()` y `toUpperCase()` convierten los caracteres alfabéticos de la cadena a minúsculas y a mayúsculas respectivamente.

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

Tras ejecutar estos métodos, la cadena original permanece intacta. Para transformar la cadena original, utilice el código siguiente:

```
str = str.toUpperCase();
```

Estos métodos funcionan con caracteres extendidos, no simplemente a–z y A–Z:

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

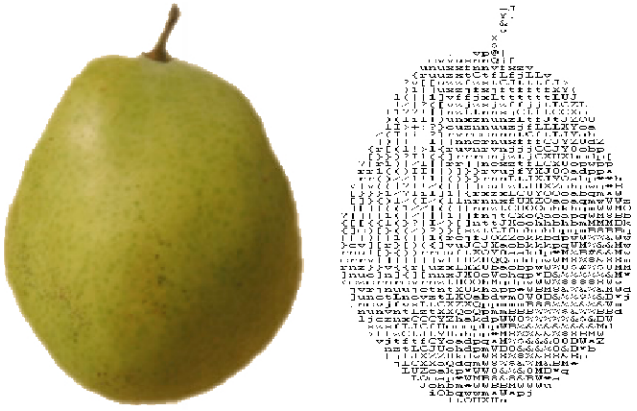
Ejemplo de cadenas: Arte ASCII

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo ASCII Art muestra diversas características del trabajo con la clase String en ActionScript 3.0, como las siguientes:

- Se utiliza el método `split()` de la clase String para extraer valores de una cadena delimitada por caracteres (información de imagen en un archivo de texto delimitado por tabulaciones).
- Se utilizan varias técnicas de manipulación de cadenas, como `split()`, la concatenación y la extracción de una parte de la cadena mediante `substring()` y `substr()` para convertir a mayúsculas la primera letra de cada palabra de los títulos de imagen.
- El método `getCharAt()` se utiliza para obtener un solo carácter de una cadena (a fin de determinar el carácter ASCII correspondiente a un valor de mapa de bits de escala de grises).
- Se utiliza la concatenación de cadenas para generar carácter a carácter la representación ASCII Art de una imagen.

El término *ASCII Art (arte ASCII)* designa representaciones textuales de una imagen, en las que representa la imagen en una cuadrícula de caracteres con fuente de espacio fijo, como los caracteres Courier New. La imagen siguiente muestra un ejemplo de arte ASCII producido por la aplicación:



La versión de arte ASCII del gráfico se muestra a la derecha

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación ASCIIArt se encuentran en la carpeta Samples/AsciiArt. La aplicación consta de los siguientes archivos:

Archivo	**Descripción**
AsciiArtApp.mxml o AsciiArtApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	La clase que proporciona la funcionalidad principal de la aplicación, incluidas la extracción de metadatos de imagen de un archivo de texto, la carga de imágenes y la administración del proceso de conversión de imagen a texto.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	Una clase que proporciona el método `parseBitmapData()` para convertir datos de imagen en una versión de tipo String.
com/example/programmingas3/asciiArt/Image.as	Una clase que representa una imagen de mapa de bits cargada.
com/example/programmingas3/asciiArt/ImageInfo.as	Una clase que representa metadatos de una imagen de arte ASCII (como el título, el URL del archivo de imagen, etc.).
image/	Una carpeta que contiene imágenes utilizadas por la aplicación.
txt/ImageData.txt	Un archivo de texto delimitado por tabulaciones, que contiene información sobre las imágenes que la aplicación debe cargar.

Extracción de valores delimitados por tabulaciones

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En este ejemplo se sigue la práctica común de almacenar datos de aplicación por separado de la aplicación en sí; de esta manera, si los datos cambian (por ejemplo, si se añade otra imagen o cambia el título de una imagen), no es necesario volver a generar el archivo SWF. En este caso, los metadatos de imagen, incluidos el título de la imagen, el URL del archivo de imagen real y algunos valores que se utilizan para manipular la imagen, se almacenan en un archivo de texto (el archivo txt/ImageData.txt file del proyecto). A continuación se describe el contenido del archivo de texto:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

El archivo utiliza un formato delimitado por tabulaciones específico. La primera línea (fila) es una fila de encabezado. Las restantes líneas contienen los siguientes datos para cada mapa de bits que se va a cargar:

- El nombre de archivo del mapa de bits.
- El nombre para mostrar del mapa de bits.
- Los valores de los umbrales blanco y negro para los mapas de bits. Son valores hexadecimales por encima o por debajo de los cuales, un píxel se considerará completamente blanco o completamente negro.

En cuanto se inicia la aplicación, la clase AsciiArtBuilder carga y analiza el contenido del archivo de texto para crear la "pila" de imágenes que se mostrará, utilizando el código siguiente del método `parseImageInfo()` de la clase AsciiArtBuilder:

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

Todo el contenido del archivo de texto está en una sola instancia de String, la propiedad `_imageInfoLoader.data`. Si se utiliza el método `split()` con el carácter de nueva línea (`"\n"`) como parámetro, la instancia de String se divide en un conjunto (`lines`) cuyos elementos son las líneas individuales del archivo de texto. A continuación, el código utiliza un bucle para trabajar con cada una de las líneas (salvo la primera, ya que contiene solo encabezados, no contenido real). Dentro del bucle, se vuelve a utilizar el método `split()` para dividir el contenido de la línea individual en un conjunto de valores (el objeto Array denominado `imageProperties`). El parámetro utilizado con el método `split()` en este caso es el carácter de tabulación (`"\t"`), ya que los valores de cada línea están delimitados por tabulaciones.

Uso de métodos String para normalizar títulos de imágenes

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las decisiones de diseño para esta aplicación es que todos los títulos de imágenes deben mostrarse con un formato estándar, con la primera letra de cada palabra convertida a mayúsculas (con la excepción de unas pocas palabras que no se suelen escribir en mayúsculas en los títulos en inglés). En lugar de suponer que el archivo de texto contiene títulos con el formato correcto, la aplicación aplica el formato a los títulos cuando los extrae del archivo de texto.

En el listado de código anterior se utiliza la siguiente línea de código como parte de la extracción de valores de metadatos de imagen individuales:

```
        imageInfo.title = normalizeTitle(imageProperties[1]);
```

En ese código, se aplica al título de imagen del archivo de texto el método `normalizeTitle()` antes de almacenarlo en el objeto ImageInfo:

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

Este método utiliza el método `split()` para dividir el título en palabras individuales (separadas por el carácter espacio), aplica a cada palabra el método `capitalizeFirstLetter()` y después utiliza el método `join()` de la clase Array para volver a combinar las palabras en una sola cadena.

Como indica su nombre, el método `capitalizeFirstLetter()` convierte a mayúscula la primera letra de cada palabra:

```
/**
 * Capitalizes the first letter of a single word, unless it's one of
 * a set of words that are normally not capitalized in English.
 */
private function capitalizeFirstLetter(word:String):String
{
    switch (word)
    {
        case "and":
        case "the":
        case "in":
        case "an":
        case "or":
        case "at":
        case "of":
        case "a":
            // Don't do anything to these words.
            break;
        default:
            // For any other word, capitalize the first character.
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

En inglés, el carácter inicial de cada una de las palabras de un título *no* va en mayúsculas si es alguna de estas palabras: “and”, “the”, “in”, “an”, “or”, “at”, “of”, o “a”. (Esta es una versión simplificada de las reglas.) Para ejecutar esta lógica, el código utiliza primero una sentencia `switch` para comprobar si la palabra es una de las palabras que no se deben escribir en mayúsculas. Si es así, el código simplemente sale de la sentencia `switch`. Por otra parte, si hubiera que escribir la palabra en mayúsculas, se hará en varios pasos, como se indica a continuación:

1. La primera letra de la palabra se extrae mediante `substr(0, 1)`, que extrae una subcadena que empieza por el carácter correspondiente al índice 0 (la primera letra de la cadena, como indica el primer parámetro `0`). La subcadena tendrá una longitud de un carácter (como indica el segundo parámetro, `1`).
2. El carácter se convierte a mayúscula mediante el método `toUpperCase()`.
3. Los caracteres restantes de la palabra original se extraen mediante `substring(1)`, que extrae una subcadena que empieza en el índice 1 (la segunda letra) hasta el final de la cadena (lo que se indica omitiendo el segundo parámetro del método `substring()`).
4. La última palabra se crea combinando la primera letra que se acaba de convertir en mayúscula con las restantes letras mediante concatenación de cadenas: `firstLetter + otherLetters`.

Creación de texto de arte ASCII

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase BitmapToAsciiConverter proporciona la funcionalidad de convertir una imagen de mapa de bits en su representación de texto ASCII. Este proceso, que se muestra aquí parcialmente, lo lleva a cabo el método `parseBitmapData()`:

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

Este código define primero una instancia de String denominada `result` que se utilizará para generar la versión de arte ASCII de la imagen de mapa de bits. A continuación, recorre píxeles individuales de la imagen de mapa de bits original. Utiliza varias técnicas de manipulación de colores (que se omiten aquí por brevedad), convierte los valores de los colores rojo, verde y azul de un píxel individual en un solo valor de escala de grises (un número entre 0 y 255). A continuación el código divide ese valor por 4 (como se indica) para convertirlo en un valor en la escala 0-63, que se almacena en la variable `index`. (Se utiliza la escala 0-63 porque la "paleta" de caracteres ASCII disponibles que utiliza esta aplicación contiene 64 valores.) La paleta de caracteres se define como una instancia de String en la clase BitmapToAsciiConverter:

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String =
"@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Como la variable `index` define el carácter ASCII de la paleta que corresponde al píxel actual de la imagen de mapa de bits, ese carácter se recupera de la instancia String de la `paleta` mediante el método `charAt()`. A continuación se añade `result` a la instancia de tipo String mediante el operador de concatenación y asignación (`+=`). Además, al final de cada fila de píxeles, un carácter de nueva línea se concatena con el final de la cadena`result`, lo que fuerza un ajuste de línea para crear una nueva fila de "píxeles" de caracteres.

Capítulo 3: Trabajo con conjuntos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los conjuntos permiten almacenar varios valores en una sola estructura de datos. Se pueden utilizar conjuntos indexados simples que almacenan valores con índices enteros ordinales fijos o conjuntos asociativos complejos que almacenan valores con claves arbitrarias. Los conjuntos también pueden ser multidimensionales y contener elementos que son conjuntos en sí mismos. Finalmente, puede utilizar un vector para un conjunto cuyos elementos son todos instancias del mismo tipo de datos.

Más temas de ayuda

Array

Vector

Fundamentos de la utilización de conjuntos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al programar es frecuente tener que trabajar con un conjunto de elementos en lugar de con un solo objeto. Por ejemplo, una aplicación de reproductor de música puede tener una lista de canciones esperando para ser reproducidas. No tiene sentido crear una variable independiente para cada canción de la lista. Es preferible agrupar todos los objetos Song y trabajar con ellos como un grupo.

Un conjunto es un elemento de programación que actúa como un contenedor para un grupo de elementos, como una lista de canciones. Normalmente todos los elementos de un conjunto son instancias de la misma clase, pero esto no es un requisito en ActionScript. Los elementos individuales de un conjunto se denominan *elementos* del conjunto. Un conjunto es como un cajón de archivador para variables. Se pueden añadir variables al conjunto como elementos (es como colocar una carpeta en el cajón del archivador). Se puede trabajar con el conjunto como una sola variable (es como llevar todo el cajón a un lugar distinto). Se puede trabajar con las variables como un grupo (es como recorrer las carpetas una a una buscando información). También se puede acceder a ellas de forma individual (es como abrir el cajón y seleccionar una sola carpeta).

Por ejemplo, suponga que está creando una aplicación de reproductor de música en la que un usuario puede seleccionar varias canciones y añadirlas a la lista de reproducción. Se puede crear en el código ActionScript un método denominado `addSongsToPlaylist()` que acepte un solo conjunto como parámetro. Independientemente del número de canciones que se añada a la lista (unas pocas, muchas o solo una), basta con llamar al método `addSongsToPlaylist()` una vez para transferirle el conjunto que contiene los objetos Song. Se puede utilizar un bucle en el método `addSongsToPlaylist()` para recorrer los elementos del conjunto (las canciones) de uno en uno y añadirlos a la lista de reproducción.

El tipo más común de conjunto de ActionScript es un *conjunto indexado*. En un conjunto indexado, cada elemento se almacena en una posición numerada (que recibe el nombre de *índice*). Para acceder a los elementos, se utiliza el número, como en una dirección. Los conjuntos indexados son suficiente para la mayoría de las necesidades de programación. La clase Array es una de las clases habituales utilizadas para representar un conjunto indexado.

A menudo, se utiliza un conjunto indexado para guardar varios elementos del mismo tipo (objetos que son instancias de la misma clase). La clase Array no tiene capacidad para restringir el tipo de elementos que contiene. La clase Vector es un tipo de conjunto indexado en el que todos los elementos de un conjunto simple son del mismo tipo. Al utilizar una instancia de Vector en lugar de una instancia de Array, también se consiguen mejoras de rendimiento y otras ventajas. la clase Vector está disponible a partir de Flash Player 10 y Adobe AIR 1.5.

Un uso especial de un conjunto indexado es un *conjunto multidimensional*. Un conjunto multidimensional es un conjunto indexado cuyos elementos son conjuntos indexados (que, a su vez, contienen otros elementos).

Otro tipo de conjunto es el *conjunto asociativo*, que utiliza una *clave* de tipo String en lugar de un índice numérico para identificar elementos individuales. Por último, ActionScript 3.0 también contiene la clase Dictionary, que representa un *diccionario*. Un diccionario es un conjunto que permite utilizar cualquier tipo de objeto como clave para distinguir entre elementos.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al programar rutinas de administración de vectores y conjuntos:

Array Objeto que se utiliza como contenedor para agrupar varios objetos.

Operador de acceso a conjunto ([]) Par de corchetes que rodean un índice o una clave que identifica de forma exclusiva a un elemento de conjunto. Esta sintaxis se utiliza después de un nombre de variable de conjunto para especificar un solo elemento del conjunto, no el conjunto completo.

Conjunto asociativo Conjunto que utiliza claves de tipo cadena para identificar elementos individuales.

Tipo base Tipo de datos que puede almacenar una instancia de Vector.

Diccionario Conjunto cuyos elementos constan de un par de objetos, denominados clave y valor. Se utiliza la clave en lugar de un índice numérico para identificar un elemento individual.

Elemento Elemento individual de un conjunto.

Índice "Dirección" numérica que se utiliza para identificar un elemento individual de un conjunto indexado.

Conjunto indexado Tipo estándar de conjunto que almacena cada elemento en una posición numerada y utiliza el número (índice) para identificar elementos individuales.

Tecla Cadena u objeto que se utiliza para identificar un elemento individual en un conjunto asociativo o un diccionario.

Conjunto multidimensional Conjunto que contiene elementos que son conjuntos en lugar de valores individuales.

T Convención estándar que se utiliza en esta documentación para representar el tipo base de una instancia de Vector, independientemente del tipo base. La convención T se utiliza para representar un nombre de clase, tal como se indica en la descripción del parámetro Type. (“T” viene de “tipo”, como en “tipo de datos”.)

Parámetro Type Sintaxis que se utiliza en el nombre de la clase Vector para especificar el tipo base del vector (tipo de datos de los objetos que almacena). La sintaxis está formada por un punto (`.`) seguido del nombre del tipo de datos encerrado entre paréntesis angulares (`<>`). Todo junto, presenta el siguiente aspecto: `Vector.<T>`. En esta documentación, la clase especificada en el parámetro Type se representa genéricamente como `T`.

Vector Tipo de conjunto cuyos elementos son todos instancias del mismo tipo de datos.

Conjuntos indexados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los conjuntos indexados almacenan una serie de uno o varios valores organizados de forma que se pueda acceder a cada valor mediante un valor entero sin signo. El primer número de índice siempre es 0 y aumenta una unidad por cada elemento que se añada al conjunto. En ActionScript 3.0, se utilizan dos clases como conjuntos indexados: la clase Array y la clase Vector.

Los conjuntos indexados usan un entero de 32 bits sin signo como número de índice. El tamaño máximo de un conjunto indexado es 2^{32} - 1 o 4.294.967.295. Si se intenta crear un conjunto de tamaño superior al tamaño máximo, se producirá un error en tiempo de ejecución.

Para acceder a un elemento individual de un conjunto indexado, debe utilizarse el operador de acceso a conjunto (`[]`) para especificar la posición del índice del elemento al que se desee acceder. Por ejemplo, el siguiente código representa el primer elemento (elemento en el índice 0) de un conjunto indexado llamado `songTitles`:

```
songTitles[0]
```

La combinación del nombre de la variable de conjunto seguido del índice entre corchetes hace las veces de identificador único. (Dicho de otro modo, se puede utilizar en cualquier situación en la que se utilizaría un nombre de variable). Es posible asignar un valor a un elemento de conjunto indexado utilizando el nombre y el índice situados a la izquierda de la sentencia de asignación:

```
songTitles[1] = "Symphony No. 5 in D minor";
```

Del mismo modo, se puede recuperar el valor de un elemento de conjunto indexado utilizando el nombre y el índice situados a la derecha de una sentencia de asignación:

```
var nextSong:String = songTitles[2];
```

También se puede utilizar una variable entre corchetes, en vez de proporcionar un valor explícito. (La variable debe contener un valor entero no negativo, como un uint, un int positivo o una instancia positiva entera de Number). Esta técnica suele utilizarse para “recorrer en bucle” los elementos de un conjunto indexado y llevar a cabo una operación en uno o en todos los elementos. El siguiente código muestra esta técnica. El código utiliza un bucle para acceder a cada valor de un objeto Array llamado `oddNumbers`. Utiliza la sentencia `trace()` para imprimir cada valor con la forma “oddNumber[*índice*] = *valor*”:

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Clase Array

El primer tipo de conjunto indexado es la clase Array. Una instancia de Array puede contener un valor de cualquier tipo de datos. El mismo objeto Array puede contener objetos que sean de distinto tipo de datos. Por ejemplo, una sola instancia de Array puede tener un valor de String en el índice 0, una instancia de Number en el índice 1 y un objeto XML en el índice 2.

Clase Vector

Otro tipo de conjunto indexado disponible en ActionScript 3.0 es la clase Vector. Una instancia de Vector es un *conjunto de tipos*, lo que significa que todos los elementos de una instancia de Vector siempre son del mismo tipo.

Nota: *la clase Vector está disponible a partir de Flash Player 10 y Adobe AIR 1.5.*

Al declarar una variable Vector o crear una instancia de un objeto Vector, se especifica explícitamente el tipo de datos de los objetos que dicho vector puede contener. El tipo de datos especificado se conoce como *tipo base* del vector. En tiempo de ejecución y de compilación (en modo estricto), se comprueba cualquier código que establezca el valor de un elemento Vector o que recupere un valor desde un vector. Si el tipo de datos del objeto que se añade o se recupera no coincide con el tipo base del vector, se produce un error.

Además de la restricción del tipo de datos, la clase Vector tiene otras limitaciones que la distinguen de la clase Array:

- Una vector es un conjunto denso. Un objeto Array puede tener valores en los índices 0 y 7, incluso si no tiene valores en las posiciones 1-6. Sin embargo, un vector debe tener un valor (o `null`) en todos los índices.
- Opcionalmente, un vector puede tener longitud fija. Esto significa que el número de elementos contenidos por el vector no puede cambiar.
- El acceso a los elementos de un vector está definido por sus límites. Nunca se puede leer un valor de un índice superior a la (`longitud` - 1) del elemento final. Nunca se puede establecer un valor con un índice superior al índice final actual. Dicho de otro modo, solo se puede establecer un valor en un índice existente o en una `[longitud]` de índice.

La consecuencia de estas restricciones hace que utilizar un vector presente tres ventajas principales frente a una instancia de Array cuyos elementos sean todos instancias de una sola clase:

- Rendimiento: el acceso y la iteración en los elementos de un conjunto son mucho más rápidos si se utiliza una instancia de Vector y no una de Array.
- Seguridad de tipos: en modo estricto, el compilador puede identificar errores de tipos de datos. Algunos ejemplos de este tipo de errores incluyen la asignación de un valor del tipo de datos incorrecto a un vector, o esperar un tipo de datos incorrecto al leer un valor en un vector. En tiempo de ejecución, los tipos de datos también se comprueban al añadir datos o leerlos en un objeto Vector. Tenga en cuenta, no obstante, que cuando se utiliza el método `push()` o el método `unshift()` para añadir valores a un vector, los tipos de datos de los argumentos no se comprueban en tiempo de compilación. Cuando se utilizan estos métodos, los valores se siguen comprobando en tiempo de ejecución.
- Fiabilidad: la comprobación del rango del motor de ejecución (o la comprobación de la longitud fija) aumenta la fiabilidad sobre los conjuntos en gran medida.

Dejando a un lado las restricciones y ventajas adicionales, la clase Vector es muy similar a la clase Array. Las propiedades y los métodos de un objeto Vector son similares (la mayoría de las veces, idénticos) a las propiedades y métodos de un objeto Array. En la mayor parte de las situaciones en las que utilizaría un objeto Array con todos sus elementos del mismo tipo de datos, es preferible usar una instancia del objeto Vector.

Creación de conjuntos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar diversas técnicas para crear una instancia de Array o una instancia de Vector. No obstante, las técnicas para crear cada tipo de conjunto son ligeramente distintas.

Creación de una instancia de Array

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear un objeto Array, llame al constructor `Array()` o utilice la sintaxis literal de Array.

La función constructora de `Array()` se puede utilizar de tres maneras distintas. En primer lugar, si se llama al constructor sin argumentos, se obtiene un conjunto vacío. Se puede utilizar la propiedad `length` de la clase Array para comprobar que el conjunto no tiene elementos. Por ejemplo, el código siguiente llama al constructor de `Array()` sin argumentos:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

En segundo lugar, si se utiliza un número como único parámetro del constructor de `Array()`, se crea un conjunto de esa longitud y se establece el valor de cada elemento en `undefined`. El argumento debe ser un entero sin signo entre 0 y 4.294.967.295. Por ejemplo, el código siguiente llama al constructor de `Array()` con un solo argumento numérico:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

En tercer lugar, si se llama al constructor y se le pasa una lista de elementos como parámetros, se crea un conjunto con elementos correspondientes a cada uno de los parámetros. El código siguiente pasa tres argumentos al constructor de `Array()`:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

También es posible crear conjuntos con literales de conjunto. Un literal de conjunto se puede asignar directamente a una variable de tipo conjunto, como se indica en el siguiente ejemplo:

```
var names:Array = ["John", "Jane", "David"];
```

Creación de una instancia de Vector

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Para crear una instancia de Vector, llame al constructor `Vector.<T>()`. También es posible crear un vector llamando a la función global `Vector.<T>()`. Esta función convierte el objeto especificado en una instancia de Vector. En Flash Professional CS5, Flash Builder 4 y Flex 4 y versiones posteriores de estas aplicaciones, también se puede crear una instancia del vector utilizando la sintaxis literal de Vector.

Cada vez que se declara una variable Vector (o, del mismo modo, un parámetro del método Vector o un tipo devuelto del método), se debe especificar el tipo base de la variable Vector. También se debe especificar el tipo base al crear una instancia de Vector llamando al constructor `Vector.<T>()`. Dicho de otro modo, cada vez que se utiliza el término `Vector` en ActionScript, va acompañado de un tipo base.

Puede especificar el tipo base de Vector mediante la sintaxis de parámetros de tipos. El parámetro type va inmediatamente después de la palabra `Vector` en el código. Está formado por un punto (`.`) seguido de la clase base encerrada entre paréntesis angulares (`<>`), tal como puede verse en este ejemplo:

```
var v:Vector.<String>;
v = new Vector.<String>();
```

En la primera línea del ejemplo, la variable `v` se declara como una instancia de `Vector.<String>`. Es decir, representa un conjunto indexado que solo puede contener instancias de String. La segunda línea llama al constructor `Vector()` para crear una instancia del mismo tipo de vector (es decir, un vector cuyos elementos sean todos objetos String). Asigna el objeto a `v`.

Uso del constructor Vector.<T>()

Si utiliza el constructor `Vector.<T>()` sin ningún argumento, crea una instancia vacía de Vector. Puede comprobar si un vector está vacío observando su propiedad `length`. Por ejemplo, el siguiente código llama al constructor `Vector.<T>()` sin ningún argumento:

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

Si conoce con antelación el número de elementos que necesita el vector inicialmente, puede predefinir el número de elementos contenidos en Vector. Para crear un vector con un determinado número de elementos, transfiera el número de elementos como primer parámetro (el parámetro `length`). Puesto que los elementos Vector no pueden estar vacíos, se llenan con instancias del tipo base. Si el tipo base es un tipo de referencia que admite valores `null`, todos los elementos contendrán `null`. En caso contrario, todos los elementos contienen el valor predeterminado para la clase. Por ejemplo, una variable uint no puede ser `null`. En consecuencia, en el siguiente código, el vector llamado `ages` se crea con siete elementos (cada uno con el valor 0):

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

Finalmente, con el constructor `Vector.<T>()`, también es posible crear un vector de longitud fija si se transfiere `true` como segundo parámetro (el parámetro `fixed`). En ese caso, el vector se crea con el número especificado de elementos y este número no puede cambiar. Debe tenerse en cuenta, no obstante, que sigue siendo posible cambiar los valores de los elementos de un vector de longitud fija.

Uso del constructor de sintaxis literal de Vector

En Flash Professional CS5, Flash Builder 4, Flex 4 y versiones posteriores de estas aplicaciones, se puede transmitir una lista de valores al constructor `Vector.<T>()` para especificar los valores iniciales de Vector:

```
// var v:Vector.<T> = new <T>[E0, ..., En-1 ,];
// For example:
var v:Vector.<int> = new <int>[0,1,2,];
```

La siguiente información se aplica a esta sintaxis:

- La coma final es opcional.
- Los elementos vacíos del conjunto no se admiten; una sentencia como, por ejemplo, `var v:Vector.<int> = new <int>[0,,2,]` genera un error del compilador.
- No se puede especificar una longitud predeterminada para la instancia de Vector. En cambio, la longitud es la misma que el número de elementos en la lista de inicialización.
- No es posible especificar si la instancia de Vector presenta una longitud fija. En su lugar, utilice la propiedad `fixed`.
- Los errores o la pérdida de datos se pueden producir si los elementos transmitidos como valores no coinciden con el tipo especificado. Por ejemplo:

  ```
  var v:Vector.<int> = new <int>[4.2]; // compiler error when running in strict mode
  trace(v[0]); //returns 4 when not running in strict mode
  ```

Uso del constructor Vector.<T>()

Además del constructor `Vector.<T>()` y los constructores de sintaxis literal de Vector, también se puede utilizar la función global `Vector.<T>()` para crear un objeto Vector. La función global `Vector.<T>()` es una función de conversión. Al llamar a la función global `Vector.<T>()`, se especifica el tipo base del vector devuelto por el método. Se transfiere un conjunto indexado sencillo (instancia de Array o de Vector) como argumento. El método devuelve un vector con el tipo base especificado y con los valores del argumento del conjunto original. El siguiente código muestra la sintaxis para llamar a la función global `Vector.<T>()`:

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

La función global `Vector.<T>()` lleva a cabo la conversión de tipos de datos en dos niveles. En primer lugar, cuando se transfiere una instancia de Array a la función, se devuelve una instancia de Vector. En segundo lugar, sea o no el conjunto original una instancia de Array o de Vector, la función intenta convertir los elementos del conjunto original en valores del tipo base. La conversión utilizar reglas de conversión estándar de tipos de datos de ActionScript. Por ejemplo, el siguiente código convierte los valores de String del objeto Array original en enteros del vector resultante. La parte decimal del primer valor (`"1.5"`) se corta y el tercer valor no numérico (`"Waffles"`) se convierte a 0 en el resultado:

```
var numbers:Vector.<int> = Vector.<int>(["1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

Si no es posible convertir alguno de los elementos originales, se produce un error.

Cuando el código llama a la función global `Vector.<T>()`, si un elemento del conjunto original es una instancia de una subclase del tipo base especificado, el elemento se añade al vector resultante (y no se produce ningún error). Utilizar la función global `Vector.<T>()` es el único modo de convertir un vector con tipo base `T` en uno con un tipo base que sea una superclase de `T`.

Inserción de elementos de conjunto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La forma más sencilla de añadir un elemento a un conjunto indexado es utilizar el operador de acceso a conjunto (`[]`). Para establecer el valor de un elemento de conjunto indexado, utilice el nombre del objeto Array o Vector y el número de índice situado a la izquierda de una sentencia de asignación:

```
songTitles[5] = "Happy Birthday";
```

Si el objeto Array o Vector no tiene aún ningún elemento en ese índice, este se crea y se almacena el valor en él. Si ya existe un valor en el índice, el nuevo valor sustituye al existente.

Un objeto Array permite crear un elemento en cualquier índice. Sin embargo, con un objeto Vector solo es posible asignar un valor a un índice existente o al siguiente índice disponible. El siguiente índice disponible corresponde a la propiedad `length` del objeto Vector. El modo más seguro de añadir un nuevo elemento a un objeto Vector es utilizar un código como el siguiente:

```
myVector[myVector.length] = valueToAdd;
```

Tres de los métodos de la clase Array y Vector (`push()`, `unshift()` y `splice()`) permiten insertar elementos en un conjunto indexado. El método `push()` añade uno o varios elementos al final de un conjunto. Es decir, el último elemento insertado en el conjunto mediante el método `push()` tendrá el número de índice más alto. El método `unshift()` inserta uno o más elementos al principio de un conjunto, que siempre tiene el número de índice 0. El método `splice()` insertará un número arbitrario de elementos en un índice especificado del conjunto.

En el ejemplo siguiente se ilustran los tres métodos. Un conjunto denominado `planets` se crea para almacenar los nombres de los planetas en orden de proximidad al sol. En primer lugar, se llama al método `push()` para agregar el elemento inicial, `Mars`. En segundo lugar, se llama al método `unshift()` para insertar el elemento que debe estar al principio del conjunto, `Mercury`. Por último, se llama al método `splice()` para insertar los elementos `Venus` y `Earth` a continuación de `Mercury`, pero antes de `Mars`. El primer argumento enviado a `splice()`, el entero 1, dirige la inserción para que se realice en el número de índice 1. El segundo argumento enviado a `splice()`, el entero 0, indica que no se debe eliminar ningún elemento. Por último, el tercer y el cuarto argumento enviados a `splice()`, `Venus` y `Earth` son los elementos que se van a insertar.

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

Los métodos `push()` y `unshift()` devuelven un entero sin signo que representa la longitud del conjunto modificado. El método `splice()` devuelve un conjunto vacío cuando se utiliza para insertar elementos, algo que puede parecer extraño, pero que tiene sentido si se tiene en cuenta la versatilidad que ofrece `splice()`. Se puede utilizar el método `splice()` no solo para insertar elementos en un conjunto, sino también para eliminar elementos de un conjunto. Cuando se utiliza para eliminar elementos, el método `splice()` devuelve un conjunto que contiene los elementos eliminados.

Nota: *si la propiedad `fixed` de un objeto Vector es `true`, el número total de elementos de dicho vector es invariable. Si intenta añadir un nuevo elemento a un vector de longitud fija con las técnicas descritas anteriormente, se producirá un error.*

Recuperación de valores y eliminación de elementos de conjunto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El modo más sencillo de recuperar el valor de un elemento en un conjunto indexado es utilizar el operador de acceso a conjunto (`[]`). Para recuperar el valor de un elemento de conjunto indexado, utilice el nombre del objeto Array o Vector y el número de índice situado a la derecha de una sentencia de asignación:

```
var myFavoriteSong:String = songTitles[3];
```

Se puede intentar recuperar un valor de un objeto Array o Vector utilizando un índice en el que no exista ningún elemento. En ese caso, un objeto Array devuelve el valor undefined y un objeto Vector emite una excepción RangeError.

Tres métodos de las clases Array y Vector (`pop()`, `shift()` y `splice()`) permiten eliminar elementos. El método `pop()` elimina un elemento del final del conjunto. Es decir, elimina el elemento que tenga el número de índice más alto. El método `shift()` elimina un elemento del principio del conjunto, lo que significa que siempre elimina el elemento con el número de índice 0. El método `splice()`, que también se puede utilizar para insertar elementos, elimina un número arbitrario de elementos empezando por el que tiene el número de índice especificado por el primer argumento enviado al método.

En el ejemplo siguiente se utilizan los tres métodos para eliminar elementos de una instancia de Array. Se crea un conjunto denominado `oceans` para almacenar los nombres de grandes masas de agua. Algunos de los nombres del conjunto son lagos, no océanos, por lo que hay que eliminarlos.

En primer lugar, se utiliza el método `splice()` para eliminar los elementos `Aral` y `Superior`, e insertar los elementos `Atlantic` e `Indian`. El primer argumento enviado a `splice()`, el entero 2, indica que la operación debe empezar por el tercer elemento de la lista, que tiene el índice 2. El segundo argumento, 2, indica que hay que eliminar dos elementos. Los restantes argumentos, `Atlantic` e `Indian`, son valores que deben insertarse a partir del índice 2.

En segundo lugar, se utiliza el método `pop()` para eliminar el último elemento del conjunto, `Huron`. Y por último, se utiliza el método `shift()` para eliminar el primer elemento del conjunto, `Victoria`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

Los métodos `pop()` y `shift()` devuelven el elemento eliminado. En una instancia de Array, el tipo de datos del valor devuelto es Object, ya que los conjuntos pueden contener valores de cualquier tipo de datos. En una instancia de Vector, el tipo de datos del valor es el tipo base del vector. El método `splice()` devuelve un objeto Array o Vector que contiene los valores eliminados. Se puede modificar el ejemplo del conjunto `oceans` de forma que la llamada a `splice()` asigne el conjunto devuelto a una nueva variable de Array, como se indica en el siguiente ejemplo:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

Es posible encontrar código que utilice el operador `delete` en un elemento del objeto Array. El operador `delete` establece el valor de un elemento de conjunto en `undefined`, pero no elimina el elemento del objeto Array. Por ejemplo, el código siguiente utiliza el operador `delete` en el tercer elemento del conjunto `oceans`, pero la longitud del conjunto sigue siendo 5:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

Se puede utilizar la propiedad `length` de un conjunto para truncar un objeto Array o Vector. Si se establece la propiedad `length` de un conjunto indexado como una longitud inferior a la longitud actual del conjunto, este se trunca y se eliminan todos los valores almacenados en los números de índice superiores al nuevo valor de `length` menos 1. Por ejemplo, si el conjunto `oceans` se ordena de modo que todas las entradas válidas están al principio del conjunto, se puede utilizar la propiedad `length` para eliminar las entradas situadas al final, tal como se indica en el siguiente código:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

Nota: *si la propiedad `fixed` de un objeto Vector es `true`, el número total de elementos de dicho vector es invariable. Si intenta eliminar o truncar un elemento de un vector de longitud fija utilizando las técnicas descritas anteriormente, se producirá un error.*

Ordenación de un conjunto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Hay tres métodos, `reverse()`, `sort()` y `sortOn()`, que permiten cambiar el orden de un conjunto indexado, ordenando o invirtiendo el orden. Todos estos métodos modifican el conjunto existente. En la tabla siguiente se resumen estos métodos y su comportamiento en los objetos Array y Vector:

Método	Comportamiento de Array	Comportamiento de Vector
`reverse()`	Cambia el orden del conjunto de forma que el último elemento se convierte en el primero, el penúltimo elemento se convierte en el segundo, etc.	Idéntico al comportamiento de Array.
`sort()`	Permite ordenar los elementos del conjunto de diversas formas predefinidas, por ejemplo, por orden alfabético o numérico. También es posible especificar un algoritmo de ordenación personalizado.	Ordena los elementos según el algoritmo de ordenación personalizado especificado.
`sortOn()`	Permite ordenar objetos con una o varias propiedades comunes y utilizar las propiedades como claves de ordenación.	No disponible en la clase Vector.

Método reverse()

El método `reverse()` no admite parámetros y no devuelve un valor, pero permite alternar el orden del conjunto de su estado actual al orden inverso. El ejemplo siguiente invierte el orden de los océanos que se muestra en el conjunto `oceans`:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

Ordenación básica con el método sort() (solo en la clase Array)

En una instancia de Array, el método `sort()` reorganiza los elementos de un conjunto con el *orden de clasificación predeterminado*. El orden de clasificación predeterminado tiene las siguientes características:

- En la ordenación se distinguen mayúsculas de minúsculas, lo que significa que los caracteres en mayúsculas preceden a los caracteres en minúsculas. Por ejemplo, la letra D precede a la letra b.
- La ordenación es ascendente, lo que significa que los códigos de caracteres inferiores (como A) preceden a los códigos de caracteres superiores (como B).
- La ordenación coloca juntos los valores idénticos, pero no usa un orden específico.
- La ordenación se basa en cadenas, lo que significa que los elementos se convierten en cadenas antes de ser comparados (por ejemplo, 10 precede a 3 porque el código de carácter de la cadena `"1"` es inferior al de la cadena `"3"`).

A veces hay que ordenar el conjunto sin tener en cuenta mayúsculas o minúsculas, o en orden descendente, o si el conjunto contiene números hay que ordenar numéricamente en lugar de alfabéticamente. El método `sort()` de la clase Array tiene un parámetro `options` que permite modificar todas las características del orden de clasificación predeterminado. Las opciones se definen mediante un conjunto de constantes estáticas de la clase Array, como se indica en la lista siguiente:

- `Array.CASEINSENSITIVE`: esta opción hace que no se tengan en cuenta las diferencias de mayúsculas y minúsculas en la ordenación. Por ejemplo, la b minúscula precede a la D mayúscula.
- `Array.DESCENDING`: invierte la ordenación ascendente predeterminada. Por ejemplo, la letra B precede a la letra A.
- `Array.UNIQUESORT`: hace que se cancele la ordenación si se encuentran dos valores idénticos.
- `Array.NUMERIC`: hace que la ordenación sea numérica, de forma que 3 preceda a 10.

En el ejemplo siguiente se ilustran algunas de estas opciones. Se crea un conjunto denominado `poets` que se ordena con varias opciones distintas.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

Ordenación personalizada con el método sort() (clases Array y Vector)

Además de la ordenación básica disponible en los objetos Array, también es posible definir una regla de ordenación personalizada. Esta técnica es la única forma del método `sort()` disponible para la clase Vector. Para definir una ordenación personalizada, debe escribir una función de ordenación personalizada y transferirla como argumento al método `sort()`.

Por ejemplo, si se tiene una lista de nombres en la que cada elemento de la lista contiene el nombre completo de una persona, pero se desea ordenar la lista por apellido, hay que utilizar una función de ordenación personalizada que analice cada elemento y use el apellido en la función de ordenación. El código siguiente muestra cómo se puede hacer esto con una función personalizada que se utiliza como parámetro del método `Array.sort()`:

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

La función de ordenación personalizada `orderLastName()` utiliza una expresión regular para extraer el apellido de cada elemento que se va a utilizar para la operación de comparación. Se utiliza el identificador de la función `orderLastName` como único parámetro al llamar al método `sort()` del conjunto `names`. La función de ordenación acepta dos parámetros, `a` y `b`, ya que procesa dos elementos de conjunto cada vez. El valor devuelto por la función de ordenación indica cómo deben ordenarse los elementos:

- Si el valor devuelto es -1, indica que el primer parámetro, `a`, precede al segundo parámetro, `b`.
- Si el valor devuelto es 1, indica que el segundo parámetro, `b`, precede al primero, `a`.
- Si el valor devuelto es 0, indica que los elementos tienen igual precedencia de ordenación.

Método sortOn() (solo en la clase Array)

El método `sortOn()` está diseñado para objetos Array con elementos que contienen objetos. Se espera que estos objetos tengan al menos una propiedad en común que se pueda utilizar como criterio de ordenación. El uso del método `sortOn()` en conjuntos de cualquier otro tipo puede producir resultados inesperados.

Nota: *la clase Vector no incluye un método `sortOn()`. Este método solo está disponible en objetos Array.*

El ejemplo siguiente revisa el conjunto `poets` de forma que cada elemento sea un objeto en lugar de una cadena. Cada objeto contiene el apellido del poeta y el año de nacimiento.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

Se puede utilizar el método `sortOn()` para ordenar el conjunto por la propiedad `born`. El método `sortOn()` define dos parámetros, `fieldName` y `options`. El argumento `fieldName` debe especificarse como una cadena. En el ejemplo siguiente, se llama a `sortOn()` con dos argumentos, `"born"` y `Array.NUMERIC`. Se utiliza el argumento `Array.NUMERIC` para asegurarse de que la ordenación se realiza numéricamente en lugar de alfabéticamente. Esto es una práctica recomendable, incluso en el caso de que todos los números tengan el mismo número de dígitos, ya que garantiza que la ordenación seguirá comportándose de la manera esperada si posteriormente se añade un número con menos (o más) dígitos al conjunto.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Ordenación sin modificar el conjunto original (solo en la clase Array)

Generalmente, los métodos `sort()` y `sortOn()` modifican un conjunto. Si se desea ordenar un conjunto sin modificarlo, se debe pasar la constante `Array.RETURNINDEXEDARRAY` como parte del parámetro `options`. Esta opción ordena a los métodos que devuelvan un nuevo conjunto que refleje la ordenación y deje el conjunto original sin cambios. El conjunto devuelto por los métodos es un conjunto simple de números de índice que refleja el nuevo orden de clasificación y no contiene ningún elemento del conjunto original. Por ejemplo, para ordenar el conjunto `poets` por año de nacimiento sin modificar el conjunto, se debe incluir la constante `Array.RETURNINDEXEDARRAY` como parte del argumento pasado para el parámetro `options`.

El ejemplo siguiente almacena la información de índice devuelta en un conjunto denominado `indices` y utiliza el conjunto `indices` junto con el conjunto `poets` sin modificar para mostrar los poetas ordenados por año de nacimiento:

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Realización de consultas en un conjunto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuatro métodos de las clases Array y Vector (`concat()`, `join()`, `slice()` y `toString()`) consultan el conjunto para obtener información, pero no lo modifican. Los métodos `concat()` y `slice()` devuelven conjuntos nuevos, mientras que los métodos `join()` y `toString()` devuelven cadenas. El método `concat()` requiere una nuevo conjunto o lista de elementos como argumentos, y los combina con el conjunto existente para crear una nuevo conjunto. El método `slice()` tiene dos parámetros, denominados `startIndex` y `endIndex`, y devuelve un nuevo conjunto que contiene una copia de los elementos "extraídos" del conjunto existente. Se empieza por extraer el elemento con índice `startIndex` y el último elemento extraído es el que tiene el índice inmediatamente anterior a `endIndex`. Esto conlleva una repetición: el elemento con índice `endIndex` no se incluye en el valor devuelto.

En el ejemplo siguiente se utilizan `concat()` y `slice()` para crear conjuntos nuevos a partir de elementos de otros conjuntos:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

Se pueden utilizar los métodos `join()` y `toString()` para consultar el conjunto y devolver su contenido en forma de cadena. Si no se utiliza ningún parámetro para el método `join()`, los dos métodos se comportarán de forma idéntica: devolverán una cadena que contiene una lista delimitada por comas de todos los elementos del conjunto. El método `join()`, a diferencia del método `toString()`, admite un parámetro denominado `delimiter`, que permite elegir el símbolo que se debe utilizar como separador entre cada elemento de la cadena devuelta.

En el ejemplo siguiente se crea un conjunto denominado `rivers` y se llama a `join()` y `toString()` para devolver los valores del conjunto en forma de cadena. El método `toString()` se utiliza para devolver valores separados por comas (`riverCSV`), mientras que el método `join()` se utiliza para devolver valores separados por el carácter `+`.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

Una característica del método `join()` que hay tener en cuenta es que las instancias anidadas de Array o Vector siempre se devuelven con los valores separados por comas, independientemente del separador que se especifique para los elementos del conjunto principal, como se indica en el siguiente ejemplo:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

Conjuntos asociativos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un conjunto asociativo, también denominado a veces *hash* o *asignación*, utiliza *claves* en lugar de un índice numérico para organizar los valores almacenados. Cada clave de un conjunto asociativo es una cadena única que se utiliza para acceder a un valor almacenado. Un conjunto asociativo es una instancia de la clase Object, lo que significa que cada clave corresponde a un nombre de propiedad. Los conjuntos asociativos son colecciones no ordenadas de pares formados por una clave y un valor. Hay que tener en cuenta en el código que las claves de un conjunto asociativo no tienen un orden específico.

ActionScript 3.0 también introduce un tipo avanzado de conjunto asociativo denominado *dictionary*. Los diccionarios, que son instancias de la clase Dictionary del paquete flash.utils, utilizan claves que pueden ser de cualquier tipo de datos. Es decir, las claves de diccionario no están limitadas a valores de tipo String.

Conjuntos asociativos con claves de tipo cadena

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen dos formas de crear conjuntos asociativos en ActionScript 3.0. La primera es utilizar una instancia de Object. Al utilizar una instancia de Object, es posible inicializar el conjunto con un literal de objeto. Una instancia de la clase Object, conocida también como *objeto genérico*, es funcionalmente idéntica a un conjunto asociativo. El nombre de cada propiedad del objeto genérico actúa a modo de clave que permite acceder a un valor almacenado.

Este código crea un conjunto asociativo denominado `monitorInfo` y utiliza un literal de objeto para inicializar el conjunto con dos pares clave/valor.

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Si no es necesario inicializar el conjunto en el momento de declararlo, se puede utilizar el constructor Object para crear el conjunto de la manera siguiente:

```
var monitorInfo:Object = new Object();
```

Una vez creado un conjunto con un literal de objeto o el constructor de la clase Object, se pueden añadir valores nuevos al conjunto mediante el operador de acceso a conjunto (`[]`) o el operador punto (`.`). En el ejemplo siguiente se añaden dos valores nuevos a `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

Hay que tener en cuenta que la clave denominada `aspect ratio` contiene un espacio en blanco. Esto es posible con el operador de acceso a conjunto (`[]`) pero genera un error si se intenta con el operador punto. No es recomendable utilizar espacios en los nombres de claves.

La segunda forma de crear un conjunto asociativo consiste en utilizar el constructor Array (o el constructor de cualquier clase dinámica) y posteriormente utilizar el operador de acceso a conjunto (`[]`) o el operador punto (`.`) para agregar pares de clave y valor al conjunto. Si se declara el conjunto asociativo con el tipo Array, no se podrá utilizar un literal de objeto para inicializar el conjunto. En el ejemplo siguiente se crea un conjunto asociativo denominado `monitorInfo` empleando el constructor de Array y se añaden claves denominadas `type` y `resolution`, junto con sus valores:

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

La utilización del constructor de Array para crear un conjunto asociativo no aporta ninguna ventaja. No se puede utilizar la propiedad `Array.length` ni ninguno de los métodos de la clase Array con los conjuntos asociativos, incluso si se utiliza el constructor de Array o el tipo de datos Array. Es mejor dejar el uso del constructor de Array para la creación de conjuntos indexados.

Conjuntos asociativos con claves de tipo objeto (Diccionarios)

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar la clase Dictionary para crear un conjunto asociativo que utilice objetos como claves en lugar de cadenas. Estos conjuntos se llaman a veces diccionarios, hashes o asignaciones. Por ejemplo, considérese una aplicación que determina la ubicación de un objeto Sprite a partir de su asociación con un contenedor específico. Se puede utilizar un objeto Dictionary para asignar cada objeto Sprite a un contenedor.

El código siguiente crea tres instancias de la clase Sprite que serán las claves del objeto Dictionary. A cada clave se le asigna un valor `GroupA` o `GroupB`. Los valores pueden ser de cualquier tipo de datos, pero en este ejemplo `GroupA` y`GroupB` son instancias de la clase Object. Posteriormente se podrá acceder al valor asociado con cada clave mediante el operador de acceso a conjunto (`[]`), como se indica en el código siguiente:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

Iteración con claves de tipo objeto

Puede repetir el contenido de un objeto Dictionary con un bucle `for..in` o un bucle `for each..in`. El bucle `for..in` permite la repetición en función de las claves, mientras que el bucle `for each..in` lo hace en función de los valores asociados con cada clave.

Utilice el bucle `for..in` para dirigir el acceso a las claves de tipo Object de un objeto Dictionary. También se puede acceder a los valores del objeto Dictionary con el operador de acceso a conjunto (`[]`). El código siguiente utiliza el ejemplo anterior del diccionario `groupMap` para mostrar la forma de repetición de un objeto Dictionary con el bucle `for..in`:

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

El bucle `for each..in` para dirigir el acceso a los valores de un objeto Dictionary. El código siguiente también utiliza el diccionario `groupMap` para mostrar la forma de repetición de un objeto Dictionary con el bucle `for each..in`:

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

Claves de tipo objeto y administración de memoria

Adobe® Flash® Player y Adobe® AIR™ utilizan un sistema de eliminación de datos innecesarios para recuperar la memoria que ya no se está utilizando. Cuando ya no quedan referencias a un objeto, el objeto se convierte en disponible para la eliminación de datos innecesarios y se recupera la memoria la próxima vez que se ejecute el sistema de eliminación de datos innecesarios. Por ejemplo, el código siguiente crea un nuevo objeto y asigna una referencia al objeto a la variable `myObject`:

```
var myObject:Object = new Object();
```

Con que exista una referencia al objeto, el sistema de eliminación de datos innecesarios no recuperará la memoria ocupada por el objeto. Si se cambia el valor de `myObject` de forma que haga referencia a un objeto distinto o se establece en el valor `null`, la memoria ocupada por el objeto original se convierte en disponible para la eliminación de datos innecesarios, pero solo si no hay otras referencias al objeto original.

Si se utiliza `myObject` como clave de un objeto Dictionary, se crea otra referencia al objeto original. Por ejemplo, el código siguiente crea dos referencias a un objeto: la variable `myObject` y la clave del objeto `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Para hacer que el objeto al que hace referencia `myObject` esté disponible para la eliminación de datos innecesarios, hay que eliminar todas las referencias a dicho objeto. En este caso, hay que cambiar el valor de `myObject` y eliminar la clave `myObject` de `myMap`, como se indica en el código siguiente:

```
myObject = null;
delete myMap[myObject];
```

Como alternativa, se puede utilizar el parámetro `useWeakReference` del constructor de Dictionary para convertir todas las claves del diccionario en *referencias débiles*. El sistema de eliminación de datos innecesarios no tiene en cuenta las referencias débiles, lo que significa que un objeto que solo tenga referencias débiles estará disponible para la eliminación de datos innecesarios. Por ejemplo, en el código siguiente no es necesario eliminar la clave `myObject` de `myMap` para hacer que el objeto esté disponible para la eliminación de datos innecesarios:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

Conjuntos multidimensionales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los conjuntos multidimensionales contienen otros conjuntos como elementos. Piense, por ejemplo, en una lista de tareas almacenadas en forma de conjunto de cadenas indexadas:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Si se desea almacenar una lista independiente de tareas por cada día de la semana, se puede crear un conjunto multidimensional con un elemento por cada día de la semana. Cada elemento contiene un conjunto indexado, similar al conjunto `tasks`, que almacena la lista de tareas. Se puede utilizar cualquier combinación de conjuntos indexados o asociativos en conjuntos multidimensionales. Los ejemplos de las secciones siguientes utilizan dos conjuntos indexados o un conjunto asociativo de conjuntos indexados. Se pueden probar las otras combinaciones como ejercicio.

Dos conjuntos indexados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se utilizan dos conjuntos indexados, se puede visualizar el resultado como una tabla o una hoja de cálculo. Los elementos del primer conjunto representan las filas de la tabla, mientras que los elementos del segundo conjunto representan las columnas.

Por ejemplo, el siguiente conjunto multidimensional utiliza dos conjuntos indexados para hacer un seguimiento de las listas de tareas para cada día de la semana. El primer conjunto, `masterTaskList`, se crea mediante el constructor de la clase Array. Cada elemento del conjunto representa un día de la semana, donde el índice 0 representa el lunes y el índice 6 representa el domingo. Estos elementos pueden considerarse como las filas de la tabla. Se puede crear la lista de tareas de cada día asignando un literal de conjunto a cada uno de los siete elementos creados en el conjunto `masterTaskList`. Los literales de conjunto representan las columnas en la tabla.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

Puede acceder a elementos individuales de cualquiera de las listas de tareas mediante el operador de acceso a conjunto (`[]`). El primer conjunto de corchetes representa el día de la semana y el segundo conjunto de corchetes representa la lista de tareas para ese día. Por ejemplo, para recuperar la segunda tarea de la lista del miércoles, se debe utilizar primero el índice 2 correspondiente al miércoles y después el índice 1 correspondiente a la segunda tarea de la lista.

```
trace(masterTaskList[2][1]); // output: dentist
```

Para recuperar la primera tarea de la lista del domingo se debe utilizar el índice 6 correspondiente al domingo y el índice 0 correspondiente a la primera tarea de la lista.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

Conjunto asociativo con un conjunto indexado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para facilitar el acceso a los conjuntos individuales, se puede utilizar un conjunto asociativo para los días de la semana y un conjunto indexado para las listas de tareas. El uso de un conjunto asociativo permite utilizar la sintaxis con punto al hacer referencia a un día específico de la semana, pero a cambio hay un procesamiento adicional en tiempo de ejecución para acceder a cada elemento del conjunto asociativo. En el ejemplo siguiente se utiliza un conjunto asociativo como la base de una lista de tareas, con un par clave-valor para cada día de la semana:

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

La sintaxis con punto facilita la lectura del código al evitar la necesidad de utilizar varios juegos de corchetes.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

Puede repetir la lista de tareas utilizando un bucle `for..in`, pero debe utilizar el operador de acceso a conjunto (`[]`) en lugar de la sintaxis de puntos para acceder al valor asociado a cada clave. Como `masterTaskList` es un conjunto asociativo, los elementos no se recuperan necesariamente en el orden esperado, como se muestra en el siguiente ejemplo:

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

Clonación de conjuntos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Array no tiene ningún método incorporado para hacer copias de conjuntos. Se puede crear una *copiasuperficial* de un conjunto llamando a los métodos `concat()` o `slice()` sin argumentos. En una copia superficial, si el conjunto original tiene elementos que son objetos, solo se copian las referencias a los objetos, en lugar de los mismos objetos. La copia señala a los mismos objetos que el conjunto original. Los cambios realizados en los objetos se reflejan en ambos conjuntos.

En una *copia completa* también se copian los objetos del conjunto original, de forma que el nuevo conjunto no señale a los mismos objetos que el conjunto original. La copia completa requiere más de una línea de código, que normalmente ordenan la creación de una función. Dicha función se puede crear como una función de utilidad global o como un método de una subclase Array.

En el ejemplo siguiente se define una función denominada `clone()` que realiza una copia completa. Se utiliza un algoritmo de una técnica de programación común en Java. La función crea una copia completa serializando el conjunto en una instancia de la clase ByteArray y leyendo a continuación el conjunto en un nuevo conjunto. Esta función acepta un objeto de forma que se pueda utilizar tanto con conjuntos indexados como con conjuntos asociativos, como se indica en el código siguiente:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

Ampliación de la clase Array

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Array es una de las pocas clases principales que no son finales, lo que significa que es posible crear una subclase de Array. Esta sección proporciona un ejemplo de cómo se puede crear una subclase de Array y se describen algunos de los problemas que pueden surgir durante el proceso.

Como se mencionó anteriormente, en ActionScript los conjuntos no tienen tipo, pero se puede crear una subclase de Array que acepte elementos de un solo tipo de datos específico. El ejemplo de las secciones siguientes define una subclase de Array denominada TypedArray que limita sus elementos a valores del tipo de datos especificado en el primer parámetro. La clase TypedArray se presenta simplemente como un ejemplo de cómo ampliar la clase Array y puede no ser adecuado para fines de producción por diversas razones. En primer lugar, la verificación de tipos se realiza en tiempo de ejecución, no en tiempo de compilación. En segundo lugar, cuando un método TypedArray encuentra un tipo no coincidente, se omite el tipo no coincidente y no se emite ninguna excepción, aunque los métodos pueden ser fácilmente modificados para emitir excepciones. Además, la clase no puede evitar el uso del operador de acceso a un conjunto para insertar valores de cualquier tipo en el conjunto. Por último, el estilo de programación favorece la simplicidad frente a la optimización del rendimiento.

Nota: *puede utilizar la técnica que se describe aquí para crear un conjunto de tipos. Sin embargo, se recomienda utilizar un objeto Vector. Una instancia de Vector es un conjunto de tipos y ofrece rendimiento y otras mejoras en comparación con la clase Array o cualquiera de sus subclases. La finalidad de esta argumentación es demostrar cómo se crea una subclase de Array.*

Declaración de la subclase

La palabra clave `extends` permite indicar que una clase es una subclase de Array. Una subclase de Array debe utilizar el atributo `dynamic`, igual que la clase Array. De lo contrario, la subclase no funcionará correctamente.

El código siguiente muestra la definición de la clase TypedArray, que contiene una constante en la que se almacena el tipo de datos, un método constructor y los cuatro métodos que pueden añadir elementos al conjunto. En este ejemplo se omite el código de cada método, pero se describe y explica completamente en las secciones siguientes:

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

Los cuatro métodos sustituidos utilizan el espacio de nombres AS3 en lugar del atributo `public`, ya que en este ejemplo se supone que la opción de compilador `-as3` está establecida en `true` y la opción de compilador `-es` está establecida en `false`. Esta es la configuración predeterminada para Adobe Flash Builder y AdobeFlashProfessional.

Los programadores expertos que prefieren utilizar herencia de prototipo pueden hacer dos pequeños cambios en la clase TypedArray para que se compile con la opción de compilador `-es` establecida en `true`. En primer lugar, deben quitarse todas las instancias del atributo `override` y debe sustituirse el espacio de nombres AS3 por el atributo `public`. En segundo lugar, debe sustituirse `Array.prototype` para las cuatro instancias de `super`.

Constructor de TypedArray

El constructor de la subclase supone un reto interesante, ya que debe aceptar una lista de argumentos de longitud arbitraria. El reto consiste en pasar los argumentos al superconstructor para crear el conjunto. Si se pasa la lista de argumentos en forma de conjunto, el superconstructor considerará que se trata de un solo argumento de tipo Array y el conjunto resultante siempre tendrá una longitud de 1 elemento. La manera tradicional de controlar las listas de argumentos es utilizar el método `Function.apply()`, que admite un conjunto de argumentos como segundo parámetro, pero la convierte en una lista de argumentos al ejecutar la función. Por desgracia, el método `Function.apply()` no se puede utilizar con constructores.

La única opción que queda es volver a generar la lógica del constructor de Array in el constructor de TypedArray. El código siguiente muestra el algoritmo utilizado en el constructor de clase Array, que se puede reutilizar en el constructor de la subclase de Array:

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

El constructor de TypedArray comparte la mayor parte del código del constructor de Array, con tan solo cuatro cambios. En primer lugar, la lista de parámetros incluye un nuevo parámetro requerido de tipo Class que permite especificar el tipo de datos del conjunto. En segundo lugar, el tipo de datos pasado al constructor se asigna a la variable `dataType`. En tercer lugar, en la sentencia `else`, el valor de la propiedad `length` se asigna después del bucle `for`, de forma que `length` incluya únicamente argumentos del tipo adecuado. Por último, el cuerpo del bucle `for` utiliza la versión sustituida del método `push()` de forma que solo se añadan al conjunto los argumentos que tengan el tipo de datos correcto. En el siguiente ejemplo se muestra la función constructora de TypedArray:

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

Métodos sustituidos de TypedArray

La clase TypedArray reemplaza los cuatro métodos de la clase Array que pueden añadir elementos a un conjunto. En cada caso, el método sustituido añade una verificación de tipos que evita la adición de elementos que no tienen el tipo de datos correcto. Posteriormente, cada método llama a la versión de sí mismo de la superclase.

El método `push()` repite la lista de argumentos con un bucle `for..in` y realiza una verificación de tipos en cada argumento. Cualquier argumento que no sea del tipo correcto se quitará del conjunto `args` con el método `splice()`. Una vez finalizado el bucle `for..in`, el conjunto `args` solo incluirá valores de tipo `dataType`. A continuación, se llama a la versión de `push()` de la superclase con el conjunto `args` actualizada, como se indica en el código siguiente:

```
AS3 override function push(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.push.apply(this, args));
}
```

El método `concat()` crea un objeto TypedArray temporal denominado `passArgs` para almacenar los argumentos que superen la verificación de tipos. Esto permite reutilizar el código de verificación de tipos que existe en el método `push()`. El bucle `for..in` repite el conjunto `args` y llama a `push()` en cada argumento. Como `passArgs` es de tipo TypedArray, se ejecuta la versión de `push()` de TypedArray. A continuación, el método `concat()` llama a su propia versión de la superclase, como se indica en el código siguiente:

```
    AS3 override function concat(...args):Array
    {
        var passArgs:TypedArray = new TypedArray(dataType);
        for (var i:* in args)
        {
            // type check done in push()
            passArgs.push(args[i]);
        }
        return (super.concat.apply(this, passArgs));
    }
```

El método `splice()` admite una lista de argumentos arbitraria, pero los dos primeros argumentos siempre hacen referencia a un número de índice y al número de elementos que se desea eliminar. Por esta razón, el método `splice()` sustituido solo hace la verificación de tipos para los elementos del conjunto `args` cuya posición de índice sea 2 o superior. Un aspecto interesante del código es que parece una llamada recursiva a `splice()` desde el bucle `for`, pero no es una llamada recursiva, ya que `args` es de tipo Array, no TypedArray, lo que significa que la llamada a `args.splice()` es una llamada a la versión del método de la superclase. Una vez finalizado el bucle `for..in`, el conjunto `args` solo incluirá valores del tipo correcto en posiciones cuyo índice sea 2 o superior, y `splice()` llamará a su propia versión de la superclase, como se indica en el siguiente código:

```
    AS3 override function splice(...args):*
    {
        if (args.length > 2)
        {
            for (var i:int=2; i< args.length; i++)
            {
                if (!(args[i] is dataType))
                {
                    args.splice(i,1);
                }
            }
        }
        return (super.splice.apply(this, args));
    }
```

El método `unshift()`, que añade elementos al principio de un conjunto, también acepta una lista de argumentos arbitraria. El método `unshift()` sustituido utiliza un algoritmo muy similar al utilizado por el método`push()`, como se indica en el siguiente ejemplo código:

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

Ejemplo de conjuntos: PlayList

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo PlayList ilustra técnicas para trabajar con conjuntos, en el contexto de una aplicación de lista de reproducción de música que administra una lista de canciones. Estas técnicas son:

- Creación de un conjunto indexado
- Añadir elementos a un conjunto indexado
- Ordenación de un conjunto de objetos por distintas propiedades y utilizando distintas opciones de ordenación
- Conversión de un conjunto en una cadena delimitada por caracteres

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación PlayList se encuentran en la carpeta Samples/PlayList. La aplicación consta de los siguientes archivos:

Archivo	Descripción
PlayList.mxml o PlayList.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/playlist/PlayList.as	Una clase que representa una lista de canciones. Utiliza un elemento Array para guardar la lista y gestiona la ordenación de los elementos de la lista.
com/example/programmingas3/playlist/Song.as	Un objeto de valor que representa información sobre una sola canción. Los elementos administrados por la clase PlayList son instancias de Song.
com/example/programmingas3/playlist/SortProperty.as	Una seudoenumeración cuyos valores disponibles representan las propiedades de la clase Song por las que una lista de objetos Song puede ordenarse.

Información general sobre la clase PlayList

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase PlayList administra un conjunto de objetos Song. Dispone de métodos públicos con funcionalidad para añadir una canción a la lista de reproducción (el método `addSong()`) y ordenar las canciones de la lista (el método `sortList()`). Además, la clase incluye una propiedad de descriptor de acceso de solo lectura, `songList`, que proporciona acceso al conjunto de canciones de la lista de reproducción. Internamente, la clase PlayList hace un seguimiento de sus canciones mediante una variable privada de tipo Array:

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

Además de la variable `_songs` de tipo Array utilizada por la clase PlayList para hacer un seguimiento de su lista de canciones, otras dos variables privadas hacen un seguimiento de si la lista debe ser ordenada (`_needToSort`) y por qué propiedad está ordenada la lista de canciones en un momento dado (`_currentSort`).

Al igual que ocurre con los objetos, declarar una instancia de Array es solo la mitad del trabajo de crear un objeto Array. Antes de acceder a las propiedades o métodos de una instancia de Array, hay que crear una instancia en el constructor de la clase PlayList.

```
public function PlayList()
{
    this._songs = new Array();
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
}
```

La primera línea del constructor crea una instancia de la variable `_songs` lista para usar. Además, se llama al método `sortList()` para establecer la propiedad por la que se va a ordenar inicialmente.

Añadir una canción a la lista

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando un usuario introduce una nueva canción en la aplicación, el código del formulario de entrada de datos llama al método `addSong()` de la clase PlayList.

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

En `addSong()`, se llama al método `push()` del conjunto `_songs`, para añadir el objeto Song que se pasó a `addSong()` como un elemento nuevo del conjunto. Con el método `push()` se añade el nuevo elemento al final del conjunto, independientemente de la ordenación que se haya aplicado previamente. Esto significa que tras llamar al método `push()`, es probable que la lista de canciones ya no esté ordenada correctamente, por lo que se establece la variable `_needToSort` en `true`. En teoría, el método `sortList()` podría llamarse inmediatamente, lo que eliminaría la necesidad de controlar si la lista está ordenada o no en un momento determinado. No obstante, en la práctica no es necesario ordenar la lista de canciones hasta inmediatamente antes de recuperarla. Al aplazar la operación de ordenación, la aplicación no realiza una ordenación innecesaria si, por ejemplo, se añaden varias canciones a la lista antes de recuperarla.

Ordenación de la lista de canciones

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Como las instancias de Song administradas por la lista de reproducción son objetos complejos, es posible que los usuarios de la aplicación deseen ordenar la lista de reproducción según distintas propiedades, como el título de la canción o el año de publicación. En la aplicación PlayList, la tarea de ordenación de la lista de canciones tiene tres partes: identificación de la propiedad con la que se debe ordenar la lista, indicación de las opciones de ordenación que se deben utilizar al ordenar con dicha propiedad y la ejecución de la operación real de ordenación.

Propiedades para la ordenación

Un objeto Song hace un seguimiento de varias propiedades, como el título de la canción, el artista, el año de publicación, el nombre de archivo y un conjunto de géneros a los que pertenece la canción seleccionada por el usuario. De estas, solo las tres primeras son prácticas para ordenar. Para mayor comodidad de los desarrolladores, el ejemplo incluye la clase SortProperty, que actúa como una enumeración con valores que representan las propiedades disponibles para ordenar.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

La clase SortProperty contiene tres constantes, `TITLE`, `ARTIST` y `YEAR`, cada una de las cuales almacena una cadena que contiene el nombre real de propiedad de la clase Song asociada que se puede utilizar para ordenar. En el resto del código, siempre que se indique una propiedad para ordenar, se hará con el miembro de la enumeración. Por ejemplo, en el constructor de PlayList, la lista se ordena inicialmente llamando al método `sortList()` de la manera siguiente:

```
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
```

Como la propiedad para ordenar se especifica como `SortProperty.TITLE`, las canciones se ordenan por su título.

Ordenación por propiedades y especificación de opciones de ordenación

El trabajo de ordenar la lista de canciones lo realiza la clase PlayList en el método `sortList()`, de la manera siguiente:

```
/**
 * Sorts the list of songs according to the specified property.
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

Al ordenar por título o por artista, tiene sentido ordenar alfabéticamente, pero al ordenar por año, es más lógico realizar una ordenación numérica. La sentencia `switch` se utiliza para definir la opción de ordenación apropiada, almacenada en la variable `sortOptions`, según el valor especificado en el parámetro `sortProperty`. En este caso también se utilizan los miembros de la enumeración designados para distinguir entre propiedades, en lugar de utilizar valores especificados en el código.

Con la propiedad de ordenación y las opciones de ordenación determinadas, el conjunto `_songs` se ordena llamando a su método `sortOn()` y pasándole esos dos valores como parámetros. Se registra la propiedad de ordenación actual, ya que la lista de canciones está ordenada actualmente.

Combinación de elementos de conjunto en una cadena delimitada por caracteres

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además de utilizar un conjunto para mantener la lista de canciones de la clase PlayList, en este ejemplo también se utilizan conjuntos en la clase Song para administrar la lista de géneros a los que pertenece una canción determinada. Considérese este fragmento de la definición de la clase Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

Al crear una nueva instancia de Song, se define el parámetro `genres` que se utiliza para especificar el género (o los géneros) al que pertenece la canción como una instancia de Array. Esto hace que sea cómodo agrupar varios géneros en una sola variable que se puede pasar al constructor. No obstante, internamente la clase Song mantiene los géneros en la variable privada `_genres` como una instancia de tipo String de valores separados por signos de punto y coma. El parámetro Array se convierte en una cadena de valores separados por signos de punto y coma llamando a su método `join()` con el valor de literal de cadena `";"` como delimitador especificado.

Con este mismo símbolo, los descriptores de acceso de `genres` permiten establecer o recuperar géneros como un conjunto:

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

Los descriptores de acceso `set` de `genres` se comportan exactamente igual que el constructor; aceptan un conjunto y llaman al método `join()` para convertirlo en una cadena de valores separados por signos de punto y coma. El descriptor de acceso `get` realiza la operación contraria: se llama el método `split()` de la variable `_genres` y la cadena se divide en un conjunto de valores utilizando el delimitador especificado (valor de cadena literal `";"`, como antes).

Capítulo 4: Gestión de errores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

"Gestionar" errores implica crear una lógica en la aplicación que responda a errores o los corrija. Dichos errores se generan cuando se compila una aplicación o se ejecuta una aplicación compilada. Si la aplicación gestiona errores, se produce *algo* como respuesta a la detección de un error, en lugar de dejar que el proceso genere el error sin emitir ninguna respuesta). Si se usa correctamente, la gestión de errores ayuda a proteger la aplicación y sus usuarios de comportamientos inesperados.

Sin embargo, la gestión de errores es una categoría extensa que incluye respuestas a muchas clases de errores generados durante la compilación o cuando se está ejecutando una aplicación. En este capí-tulo se analiza cómo gestionar errores en tiempo de ejecución, los distintos tipos de errores que se pueden generar y las ventajas del sistema de gestión de errores de ActionScript 3.0.

Fundamentos de la gestión de errores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un error en tiempo de ejecución es algo que no funciona en el código ActionScript y que impide la ejecución prevista del contenido de ActionScript. Para asegurarse de que los usuarios pueden ejecutar correctamente el código ActionScript, se debe añadir a la aplicación código para gestionar errores, es decir, para corregirlos o al menos indicar al usuario qué ha sucedido. Este proceso se denomina *gestión de errores*.

La gestión de errores es un campo amplio que incluye respuestas a muchas clases de errores generados durante la compilación o mientras se ejecuta una aplicación. Los errores que se producen en tiempo de compilación suelen ser más fáciles de identificar; se deben corregir para completar el proceso de creación de un archivo SWF.

La detección de errores en tiempo de ejecución puede resultar más difícil, ya que para que se produzcan, el código debe ejecutarse. Si un segmento de su programa tiene varias ramas de código, como una sentencia `if..thenthen..else`, debe comprobar todas las condiciones posibles, con todos los valores de entrada posibles que puedan utilizar los usuarios reales, con el fin de confirmar que el código no contenga errores.

Los errores en tiempo de ejecución se pueden dividir en dos categorías: los *errores de programa* son errores del código ActionScript, como la especificación del tipo de datos incorrecto para un parámetro de método; los *errores lógicos* son errores de la lógica (la comprobación de datos y la manipulación de valores) del programa, como la utilización de la fórmula incorrecta para calcular tipos de interés en una aplicación bancaria. Los dos tipos de error pueden detectarse y corregirse si se prueba la aplicación minuciosamente.

Lo ideal sería que se identificaran y se eliminaran todos los errores de la aplicación antes de que pasara a disposición de los usuarios finales. Sin embargo, no todos los errores pueden predecirse o evitarse. Por ejemplo, imagine que la aplicación ActionScript carga información de un sitio web determinado que se encuentra fuera de su control. Si en algún momento dicho sitio no está disponible, la parte de la aplicación que depende de los datos externos no se comportará correctamente. El aspecto más importante de la gestión de errores implica prepararse para estos casos desconocidos, así como gestionarlos adecuadamente para que los usuarios puedan seguir utilizando la aplicación, o al menos obtengan un mensaje de error claro que indique el fallo de funcionamiento.

Los errores en tiempo de ejecución se representan de dos formas en ActionScript:

- Clases Error: muchos errores tienen una clase de error asociada a ellos. Cuando se produce un error, el motor de ejecución de Flash (como Flash Player o Adobe AIR) crea una instancia de la clase de error específica asociada a dicho error. El código puede utilizar información contenida en ese objeto de error para generar una respuesta adecuada al error.
- Eventos de error: a veces se produce un error cuando el motor de ejecución de Flash normalmente desencadenaría un evento. En estos casos, se desencadena un evento de error. Cada evento de error tiene una clase asociada y el motor de ejecución de Flash transmite una instancia de dicha clase a los métodos que están suscritos al evento de error.

Para determinar si un método concreto puede desencadenar un error o un evento de error, consulte la entrada del método en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al programar rutinas de gestión de errores:

Asincrónico Comando de programa, como una llamada de método que no proporciona un resultado inmediato; en su lugar, origina un resultado (o error) en forma de evento.

Capturar Cuando se produce una excepción (error en tiempo de ejecución) y el código la reconoce, se dice que el código *captura* la excepción. Una vez capturada la excepción, el motor de ejecución de Flash deja de notificar la excepción a otro código ActionScript.

Versión de depuración Versión especial del motor de ejecución de Flash como, por ejemplo, versión de depuración de Flash Player o AIR Debug Launcher (ADL), que incluye código para notificar a los usuarios errores en tiempo de ejecución. En la versión estándar de Flash Player o Adobe AIR (la que tienen la mayoría de los usuarios), se omiten los errores no gestionados por el código ActionScript. En las versiones de depuración (que se incluye en Adobe Flash CS4 Professional y Adobe Flash Builder), aparece un mensaje de advertencia cuando se produce un error no gestionado.

Excepción Error que sucede mientras se ejecuta una aplicación y que el motor de ejecución de Flash no puede resolver por sí solo.

Nueva emisión Cuando el código detecta una excepción, el motor de ejecución de Flash no vuelve a notificar otros objetos de la excepción. Si resulta importante notificar la excepción a otros objetos, el código debe *volver a emitir* la excepción para que se vuelva a iniciar el proceso de notificación.

Sincrónico Comando de programa, como una llamada de método, que proporciona un resultado inmediato (o emite inmediatamente un error), lo que implica que la respuesta puede utilizarse con el mismo bloque de código.

Emitir Operación de notificación al motor de ejecución de Flash (y, por lo tanto, notificación a otros objetos y al código ActionScript) de que se ha producido un error; se conoce como *emitir* un error.

Tipos de errores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al desarrollar y ejecutar aplicaciones, se detectan diferentes tipos de errores, con su correspondiente terminología. En la tabla siguiente se presentan los principales tipos de errores y sus términos relacionados:

- El compilador de ActionScript captura los *errores en tiempo de compilación* durante la compilación del código. Estos errores se producen cuando los problemas sintácticos del código impiden crear la aplicación.

- Los *errores en tiempo de ejecución* se producen al ejecutar la aplicación tras compilarla. Los errores en tiempo de ejecución representan errores causados mientras un archivo SWF se reproduce en el motor de ejecución de Flash (como Adobe Flash Player o Adobe AIR). En la mayoría de los casos, es posible gestionar los errores en tiempo de ejecución conforme suceden, informando sobre ellos al usuario y adoptando medidas para que la aplicación se siga ejecutando. Si el error es grave (por ejemplo, no se puede establecer una conexión con un sitio web remoto o cargar los datos requeridos), se puede recurrir a la gestión de errores para que la aplicación finalice sin problemas.
- Los *errores sincrónicos* son errores en tiempo de ejecución que se producen cuando se llama una función. Por ejemplo, si se intenta utilizar un método específico y el método que se transmite al argumento no es válido, por lo que el motor de ejecución de Flash genera una excepción. La mayor parte de los errores se producen de forma sincrónica (cuando se ejecuta la sentencia) y el flujo del control se pasa inmediatamente a la sentencia `catch` más apropiada.

  Por ejemplo, el fragmento de código siguiente genera un error en tiempo de ejecución porque no se llama al método `browse()` antes de que el programa intente cargar un archivo:

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload(new URLRequest("http://www.yourdomain.com/fileupload.cfm"));
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  En este caso, un error en tiempo de ejecución se genera sincrónicamente, ya que Flash Player determinó que el método `browse()` no se llamó antes de intentar cargar el archivo.

  Para obtener información detallada sobre la gestión de errores sincrónicos, consulte "Gestión de errores sincrónicos en una aplicación" en la página 59.

- Los *erroresasincrónicos* son errores en tiempo de ejecución que se producen fuera del flujo normal del programa. Generan eventos y los detectores de eventos los capturan. En las operaciones asincrónicas, una función inicia una operación, pero no espera a que se complete. Se puede crear un detector de eventos de error para esperar a que la aplicación o el usuario realicen alguna operación; si esta falla, el error se captura con un detector de eventos y se responde al evento de error. A continuación, el detector de eventos llama a una función de controlador de eventos para responder al evento de error de una manera útil. Por ejemplo, el controlador de eventos puede iniciar un cuadro de diálogo que solicite al usuario que resuelva el error.

  Considérese el ejemplo de error sincrónico relativo a la carga de archivo presentado anteriormente. Si se llama correctamente al método `browse()` antes de iniciar una carga de archivo, Flash Player distribuirá varios eventos. Por ejemplo, cuando se inicia una carga, se distribuye el evento `open`. Si la operación de carga de archivo finaliza correctamente, se distribuye el evento `complete`. Puesto que la gestión de eventos es asincrónica (es decir, que no se produce en momentos específicos, conocidos y designados previamente), hay que utilizar el método `addEventListener()` para detectar estos eventos específicos, tal como se muestra en el código siguiente:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

Para obtener información detallada sobre la gestión de errores asincrónicos, consulte "Respuesta al estado y a los eventos de error" en la página 64.

- Las *excepciones no capturadas* son errores generados sin una lógica (como una sentencia `catch`) que pueda responder a ellos. Si la aplicación genera un error y no se encuentran controladores de eventos o sentencias `catch` en el nivel actual o superior para gestionar el error, este se considera una excepción no capturada.

  Cuando se produce un error no capturado, el motor de ejecución distribuye un evento `uncaughtError`. Este evento también se denomina "controlador de error global". Este evento se distribuye mediante el objeto UncaughtErrorEvents del archivo SWF, que está disponible a través de la propiedad `LoaderInfo.uncaughtErrorEvents`. Si no hay detectores registrados para el evento `uncaughtError`, el motor de ejecución omite los errores no capturados e intenta continuar ejecutándose, siempre que el error no detenga el archivo SWF.

  Además de distribuir el evento `uncaughtError`, las versiones de depuración del motor de ejecución de Flash responden a errores no capturados terminando el script actual. Posteriormente, muestran el error no capturado en la salida de la sentencia `trace`, o bien, mediante la especificación del mensaje de error en un archivo de registro. Si el objeto de excepción es una instancia de la clase Error o una de sus subclases, la información de seguimiento de pila también se visualizará en la salida. Para obtener más información sobre el uso de la versión de depuración de los motores de ejecución de Flash, consulte "Trabajo con las versiones de depuración de los motores de ejecución de Flash" en la página 58.

  Nota: *durante el procesamiento de un evento uncaughtError, si se emite un evento error desde un controlador uncaughtError, el controlador de eventos se llama varias veces. Esto tiene como resultado un bucle infinito de excepciones. Se recomienda evitar este escenario.*

Gestión de errores en ActionScript 3.0

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puesto que muchas aplicaciones pueden ejecutarse sin crear la lógica para gestionar errores, los desarrolladores pueden sentirse tentados de aplazar la aplicación de la gestión de errores en sus aplicaciones. Sin embargo, sin la gestión de errores, una aplicación puede bloquearse fácilmente o frustrar al usuario si algo no funciona de la forma prevista. ActionScript 2.0 presenta una clase Error que permite crear una lógica en funciones personalizadas y generar una excepción con un mensaje específico. Puesto que la gestión de errores es esencial para facilitar la utilización de las aplicaciones, ActionScript 3.0 incluye una arquitectura ampliada para capturar errores.

Nota: *aunque en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash se documentan las excepciones que generan muchos métodos, puede que no se incluyan todas las excepciones posibles de cada método. Puede que un método genere una excepción para errores de sintaxis u otros problemas que no se menciona explícitamente en la descripción del método, aun cuando en esta se enumeren algunas de las excepciones.*

Elementos de la gestión de errores de ActionScript 3.0

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye muchas herramientas para la gestión de errores:

- Clases Error. ActionScript 3.0 incluye una amplia gama de clases Error que amplían el ámbito de situaciones que pueden producir objetos de error. Cada clase Error ayuda a las aplicaciones a gestionar y responder a condiciones de error específicas, ya sean relativas a errores de sistema (como una condición MemoryError), errores de codificación (como una condición ArgumentError) errores de red y comunicación (como una condición URIError), o bien otras situaciones. Para obtener más información sobre cada clase, consulte "Comparación de las clases Error" en la página 68.
- Menos errores sin mensaje. En versiones anteriores de Flash Player, los errores se generaban y mostraban solo si se usaba explícitamente la sentencia `throw`. En Flash Player 9 y versiones posteriores de los motores de ejecución de Flash, los métodos y las propiedades de ActionScript generan errores en tiempo de ejecución que permiten gestionar estas excepciones de forma más eficaz cuando se producen, así como reaccionar individualmente ante cada excepción.
- Se muestran mensajes de error claros durante la depuración. Si se usa la versión de depuración de un motor de ejecución de Flash, el código o las situaciones que produzcan problemas generarán mensajes de error detallados que permitirán identificar fácilmente los motivos del error en un bloque de código determinado. Estos mensajes hacen que la solución de errores resulte más eficaz. Para obtener más información, consulte "Trabajo con las versiones de depuración de los motores de ejecución de Flash" en la página 58.
- Los errores precisos permiten que los usuarios vean mensajes de error claros. En las versiones anteriores de Flash Player, el método `FileReference.upload()` devolvía un valor booleano `false` si la llamada `upload()` era incorrecta, e indicaba uno de cinco errores posibles. Si se produce un error al llamar al método `upload()` en ActionScript 3.0, cuatro errores específicos ayudan a mostrar mensajes de error más precisos a los usuarios finales.
- Gestión de errores mejorada. Se generan errores diferentes para una gran variedad de situaciones habituales. Por ejemplo, en ActionScript 2.0, antes de que se llenara un objeto FileReference, la propiedad `name` tenía el valor `null`; de este modo, antes de poder usar o mostrar la propiedad `name`, es necesario asegurarse de que el valor está establecido y no es `null`. En ActionScript 3.0, si se intenta acceder a la propiedad `name` antes de que se especifique su valor, Flash Player generar un error de tipo IllegalOperationError, que indica que no se ha establecido el valor y que se pueden usar bloques `try..catch..finally` para gestionar el error. Para obtener más información, consulte "Uso de sentencias try..catch..finally, sentencias" en la página 59.

- No se producen problemas de rendimiento significativos. La utilización de los bloques `try..catch..finally` para gestionar errores no consume recursos adicionales (o consume muy pocos recursos) en comparación con las versiones anteriores de ActionScript.
- Existe una clase ErrorEvent que permite crear detectores de eventos de errores asincrónicos específicos. Para obtener más información, consulte "Respuesta al estado y a los eventos de error" en la página 64.

Estrategias de gestión de errores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Mientras la aplicación no detecte una situación problemática, se podrá ejecutar correctamente si no se crea una lógica de gestión de errores en el código. Sin embargo, si no se gestionan los errores activamente y la aplicación detecta un problema, los usuarios nunca sabrán por qué falla la aplicación.

Hay diferentes formas de abordar la gestión de errores en la aplicación. En la lista siguiente se resumen las tres opciones principales para gestionar errores:

- Uso de las sentencias `try..catch..finally`. Estas sentencias capturarán los errores cuando se produzcan. Se pueden anidar las sentencias en una jerarquía para capturar excepciones en varios niveles de la ejecución del código. Para obtener más información, consulte "Uso de sentencias try..catch..finally, sentencias" en la página 59.
- Creación de objetos de error personalizados. Se puede usar la clase Error para crear objetos de error personalizados, lo que permite hacer un seguimiento de operaciones específicas de la aplicación que no estén incluidas en los tipos de error integrados. Posteriormente puede utilizar sentencias `try..catch..finally` en sus propios objetos de error personalizados. Para obtener más información, consulte "Creación de clases de error personalizadas" en la página 63.
- Especificación de detectores y controladores de eventos para responder a eventos de error. Mediante esta estrategia, se pueden crear controladores de error globales que permiten gestionar eventos similares sin duplicar demasiado código en bloques `try..catch..finally`. Además, es más probable capturar errores asincrónicos con este método. Para obtener más información, consulte "Respuesta al estado y a los eventos de error" en la página 64.

Trabajo con las versiones de depuración de los motores de ejecución de Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Adobe ofrece a los desarrolladores ediciones especiales de los motores de ejecución de Flash para ayudar en las operaciones de depuración. Al instalar Adobe Flash Professional o Adobe Flash Builder, se obtiene una copia de la versión de depuración de Flash Player. También se obtiene una utilidad para la depuración de aplicaciones de Adobe AIR, denominada ADL, al instalar una de estas herramientas o como parte del SDK de Adobe AIR.

Existe una diferencia notable en la forma en que las versiones de depuración y la versiones comerciales de Flash Player y Adobe AIR indican los errores. Las versiones de depuración muestran el tipo de error (como Error, IOError, o EOFError de carácter genérico), el número de error y un mensaje de error legible para el usuario. Las versiones comerciales muestran únicamente el tipo y número de error. Por ejemplo, considérese el fragmento de código siguiente:

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

Si el método `readBoolean()` genera un error de tipo EOFError en la versión de depuración de Flash Player, se mostrará el mensaje siguiente en el campo de texto `tf`: "EOFError: Error #2030: Se ha detectado el final del archivo".

El mismo código en una versión comercial de Flash Player o Adobe AIR mostraría el siguiente texto: "EOFError: Error #2030."

Nota: *los reproductores de depuración emiten un evento llamado "allComplete"; evite crear eventos presonalizados con el nombre "allComplete". Si lo hace, el comportamiento será impredecible durante la depuración.*

Para reducir los recursos y el tamaño en las versiones comerciales, no se muestran cadenas de mensajes de error. El número de error se puede consultar en la documentación (apéndices de Referencia de Adobe ActionScript 3.0 para la plataforma de Adobe Flash) para conocer el significado del mensaje de error. También es posible reproducir el error con las versiones de depuración de Flash Player y AIR para ver el mensaje completo.

Gestión de errores sincrónicos en una aplicación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La gestión de errores más habitual hace referencia a la lógica de gestión de errores sincrónicos, en la que se insertan sentencias en el código para capturar errores sincrónicos mientras se ejecuta una aplicación. El tipo de gestión de errores permite que la aplicación observe los errores en tiempo de ejecución y se recupere cuando fallen las funciones. La lógica para capturar un error sincrónico incluye sentencias `try..catch..finally`, que literalmente intentan una operación, capturan cualquier respuesta de error del motor de ejecución de Flash y finalmente ejecutan alguna otra operación para administrar la operación fallida.

Uso de sentencias try..catch..finally, sentencias

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando trabaje con errores en tiempo de ejecución sincrónicos, utilice sentencias `try..catch..finally` para capturar errores. Si se produce un error de tiempo de ejecución, el motor de ejecución de Flash genera una excepción, lo que significa que se suspende la ejecución normal y se crea un objeto especial de tipo Error. El objeto Error se genera en el primer bloque `catch` disponible.

La sentencia `try` incluye sentencias que podrían provocar errores. La sentencia `catch` siempre se debe utilizar con una sentencia `try`. Si se detecta un error en una de las sentencias del bloque `try`, se ejecutarán las sentencias `catch` asociadas a la sentencia `try`.

La sentencia `finally` incluye sentencias que se ejecutarán independientemente de que se produzcan errores en el bloque `try`. Si no hay errores, las sentencias del bloque `finally` se ejecutarán una vez que hayan finalizado las sentencias del bloque `try`. Si se producen errores, primero se ejecutará la sentencia `catch` correspondiente, seguida de las instancias del bloque `finally`.

El código siguiente muestra la sintaxis necesaria para usar las sentencias `try..catch..finally`:

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Cada sentencia `catch` identifica el tipo de excepción específico que gestiona. La sentencia `catch` puede especificar solo clases de errores que son subclases de la clase Error. Cada sentencia `catch` se comprueba por orden. Solo se ejecutará la primera sentencia `catch` que coincida con el tipo de error generado. Dicho de otra forma, si primero se comprueba la clase Error de nivel superior y luego la subclase de la clase Error, únicamente coincidirá la clase Error de nivel superior. En el código siguiente se muestra este escenario:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

El código anterior muestra el resultado siguiente:

```
<Error> I am an ArgumentError
```

Para capturar correctamente la clase de error ArgumentError, es necesario asegurarse de que los tipos de error más específicos se enumeran primero, y que los más genéricos aparecen después, tal como se muestra en el código siguiente:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

Varios métodos y propiedades de la API de ActionScript generan errores en tiempo de ejecución si detectan errores durante la ejecución. Por ejemplo, el método `close()` de la clase Sound genera un error IOError si no puede cerrar el flujo de audio, tal como se muestra en el código siguiente:

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

A medida que el usuario se familiarice con Referencia de ActionScript 3.0 para la plataforma de Adobe Flash, reconocerá los métodos que generan excepciones, tal y como se indica en la descripción de cada método.

La sentencia throw

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los motores de ejecución de Flash generan excepciones cuando localizan errores en la aplicación que se está ejecutando. Asimismo, se pueden generar excepciones explícitamente mediante la sentencia `throw`. Si se generan errores de forma explícita, Adobe recomienda generar instancias de la clase Error o sus subclases. En el código siguiente se muestra una sentencia `throw` que genera una instancia de la clase Error, `MyErr`, y que termina por llamar a la función `myFunction()` como respuesta tras la generación del error:

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

Debe tenerse en cuenta que las sentencias `catch` se ordenan de manera que los tipos de datos más específicos se muestren en primer lugar. Si la sentencia `catch` del tipo de datos Number se muestra primero, las sentencias catch de los tipos de datos uint e int nunca se ejecutarán.

Nota: *en el lenguaje de programación Java, cada función que puede generar una excepción debe declararlo y enumerar las clases de excepción generables en una cláusula* `throws` *asociada a la declaración de la función. ActionScript no requiere la declaración de las excepciones que puede generar una función.*

Visualización de un mensaje de error simple

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las mayores ventajas del nuevo modelo de eventos de error y excepción radica en que permite indicar a los usuarios cuándo y por qué falla una acción. La parte del usuario consiste en escribir el código para que se muestre el mensaje y se ofrezcan opciones a modo de respuesta.

El siguiente código incluye una sencilla sentencia `try..catch` para mostrar el error en un campo de texto:

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

Al usar una mayor gama de clases de error y errores de compilador incorporados, ActionScript 3.0 proporciona más información que las versiones anteriores acerca del motivo por el que falla algo. Esta información permite crear aplicaciones más estables con una mejor gestión de errores.

Regeneración de errores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al crear aplicaciones, pueden existir varias circunstancias en las que sea necesario volver a generar un error, si este no consigue gestionarse adecuadamente. Por ejemplo, el siguiente código muestra un bloque anidado `try..catch`, que regenera un error de tipo ApplicationError personalizado si el bloque anidado `catch` no puede gestionar el error:

```
try
{
    try
    {
        trace("<< try >>");
        throw new ApplicationError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

La salida del fragmento anterior será de esta forma:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

El bloque `try` anidado genera un error ApplicationError personalizado que captura el bloque `catch` posterior. Este bloque `catch` anidado puede intentar gestionar el error y, si no lo consigue, puede generar el objeto ApplicationError en el bloque `try..catch` en el que está contenido.

Creación de clases de error personalizadas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede ampliar una de las clases Error estándar para crear clases de error especializadas propias en ActionScript. Existen varias razones por las que se crean clases de error personalizadas:

- Para identificar errores o grupos de errores específicos que son exclusivos de la aplicación.

  Por ejemplo, es posible que se deseen realizar diferentes acciones en el caso de errores generados por el código personalizado, además de los que captura un motor de ejecución de Flash. Se puede crear una subclase de la clase Error para hacer un seguimiento del nuevo tipo de datos de error en bloques `try..catch`.

- Para proporcionar funciones de visualización de error exclusivas para errores generados por la aplicación.

  Por ejemplo, se puede crear un nuevo método `toString()` que aplique formato a los mensajes de error de una manera determinada. Asimismo, se puede definir un método `lookupErrorString()` que capture un código de error y recupere el mensaje adecuado según la preferencia de idioma del usuario.

Las clases de error especializadas deben ampliar la clase Error principal de ActionScript. A continuación, se muestra un ejemplo de clase AppError especializada que amplía la clase Error:

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

Ejemplo de la utilización de AppError en un proyecto:

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

Nota: *si se desea sustituir el método `Error.toString()` en la subclase, hay que proporcionarle un parámetro . . . (resto). La especificación del lenguaje ECMAScript en la que está basado ActionScript 3.0 define el método `Error.toString()` de esta forma, y ActionScript 3.0 lo define del mismo modo para conseguir compatibilidad con versiones anteriores. Por lo tanto, si se sustituye el método `Error.toString()`, los parámetros deben coincidir exactamente. No deben transmitirse parámetros al método `toString()` en tiempo de ejecución, ya que dichos parámetros se omitirán.*

Respuesta al estado y a los eventos de error

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las mejoras más evidentes en la gestión de errores de ActionScript 3.0 es la compatibilidad con la gestión de eventos de error, que permite responder a errores asincrónicos mientras se ejecuta una aplicación. (Para obtener una definición de errores asincrónicos, consulte "Tipos de errores" en la página 54.)

Se pueden crear detectores y controladores de eventos para responder a los eventos de error. Muchas clases distribuyen eventos de error igual que otros eventos. Por ejemplo, una instancia de la clase XMLSocket suele distribuir tres tipos de eventos: `Event.CLOSE`, `Event.CONNECT` y `DataEvent.DATA`. No obstante, cuando se produce un problema, la clase XMLSocket puede distribuir los errores `IOErrorEvent.IOError` o `SecurityErrorEvent.SECURITY_ERROR`. Para obtener más información sobre detectores y controladores de eventos, consulte "Gestión de eventos" en la página 127.

Los eventos de error se enmarcan en una de estas dos categorías:

- Eventos de error que amplían la clase ErrorEvent

  La clase flash.events.ErrorEvent contiene las propiedades y los métodos para gestionar los errores relacionados con las operaciones de red y comunicaciones en una aplicación en ejecución. Las clases AsyncErrorEvent, IOErrorEvent y SecurityErrorEvent amplían la clase ErrorEvent. Si se usa la versión de depuración de un motor de ejecución de Flash, un cuadro de diálogo indicará en tiempo de ejecución los eventos de error que encuentre el reproductor sin las funciones de detección.

- Eventos de error basados en estado

  Los eventos de error basados en estado hacen referencia a las propiedades `netStatus` y `status` de las clases de red y comunicaciones. Si el motor de ejecución de Flash detecta un problema al leer o escribir datos, el valor de las propiedades `netStatus.info.level` o `status.level` (según el objeto de clase que se utilice) se establece en `"error"`. A este error se responde comprobando si la propiedad `level` contiene el valor `"error"` en la función de controlador de eventos.

Trabajo con eventos de error

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ErrorEvent y sus subclases contienen tipos de error para gestionar errores distribuidos por los motores de ejecución de Flash cuando intentan leer o escribir datos.

En el siguiente ejemplo se utiliza una sentencia `try..catch` y controladores de eventos de error para mostrar todos los errores detectados al intentar leer un archivo local. Se puede añadir código de gestión más avanzado a fin de proporcionar opciones a los usuarios o bien gestionar el error automáticamente en los lugares indicados con el comentario "escribir código de gestión de errores aquí":

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }
```

```
        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

Trabajo con eventos de cambio de estado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los motores de ejecución de Flash cambian dinámicamente el valor de las propiedades `netStatus.info.level` o `status.level` para las clases que admiten la propiedad `level` mientras se ejecuta una aplicación. Las clases que tienen la propiedad `netStatus.info.level` son NetConnection, NetStream y SharedObject. Las clases que tienen la propiedad `status.level` son HTTPStatusEvent, Camera, Microphone y LocalConnection. Se puede escribir una función de controlador para responder al cambio del valor `level` y hacer un seguimiento de los errores de comunicación.

En el siguiente ejemplo se usa una función `netStatusHandler()` para probar el valor de la propiedad `level`. Si la propiedad `level` indica que se ha detectado un error, el código realiza un seguimiento del mensaje "Video stream failed".

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

Comparación de las clases Error

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript proporciona varias clases Error predefinidas. Sin embargo, puede utilizar la mismas clases Error en su propio código. Existen dos tipos principales de clases Error en ActionScript 3.0: las clases Error principales de ActionScript y las clases Error del paquete flash.error. El contenido de paquete flash.error está formado por clases adicionales, introducidas para ayudar al desarrollo y la depuración de aplicaciones de ActionScript 3.0.

Clases Error principales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Entre las clases de error principales se incluyen Error, ArgumentError, EvalError, RangeError, ReferenceError, SecurityError, SyntaxError, TypeError, URIError y VerifyError. Cada una de estas clases se encuentra en el espacio de nombres de nivel superior.

Nombre de clase	Descripción	Notas
Error	La clase Error puede usarse para generar excepciones. Se trata de la clase base para otras clases de excepción definidas en ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError y URIError.	La clase Error es la clase base para todos los errores en tiempo de ejecución y se recomienda para cualquier clase de error personalizada.
ArgumentError	La clase ArgumentError representa un error que se produce cuando los valores de parámetro proporcionados durante una llamada de función no coinciden con los parámetros definidos para dicha función.	Ejemplos de errores de argumento: • Se proporcionan pocos o demasiados argumentos a un método. • Se esperaba que un argumento fuera miembro de una enumeración, pero no fue así.
EvalError	La excepción EvalError se genera si se pasan parámetros al constructor de la clase Function, o bien si el código del usuario llama a la función `eval()`.	En ActionScript 3.0, se ha eliminado la compatibilidad con la función `eval()`; los intentos de uso de esta función generarán errores. Las versiones anteriores de Flash Player utilizaban la función `eval()` para acceder a variables, propiedades, objetos o clips de película por su nombre.
RangeError	Se genera una excepción RangeError si un valor numérico queda fuera del rango admitido.	Por ejemplo, la clase Timer generará una excepción RangeError si la demora es negativa o no finita. También se puede generar RangeError si se intenta añadir un objeto de visualización a una profundidad no válida.
ReferenceError	Se emite una excepción ReferenceError cuando se intenta realizar una referencia a una propiedad no definida en un objeto cerrado (no dinámico). Las versiones del compilador de ActionScript anteriores a ActionScript 3.0 no generaban errores al intentar acceder a una propiedad `undefined`. Sin embargo, ActionScript 3.0 emite la excepción ReferenceError en estas condiciones.	Las excepciones para variables no definidas señalan errores potenciales, lo que ayuda a mejorar la calidad del software. Sin embargo, si el usuario no está acostumbrado a inicializar las variables, es posible que este nuevo comportamiento de ActionScript requiera cambios en sus hábitos de programación.

Nombre de clase	Descripción	Notas
SecurityError	La excepción SecurityError se genera cuando se produce una infracción de la seguridad y se deniega el acceso.	Ejemplos de errores de seguridad: • Se realiza un acceso a una propiedad o una llamada a un método no autorizada a través del límite del entorno limitado de seguridad. • Ha habido un intento de acceso a una URL no permitida por el entorno limitado de seguridad. • Se ha intentado establecer una conexión de socket con un puerto, pero la política de sockets necesaria no estaba presente. • Ha habido un intento de acceso a la cámara o al micrófono del usuario; este ha denegado la solicitud de acceso al dispositivo.
SyntaxError	Se genera una excepción SyntaxError cuando se produce un error de análisis en el código de ActionScript.	Se puede generar SyntaxError en las circunstancias siguientes: • ActionScript emite excepciones SyntaxError cuando la clase RegExp analiza una expresión regular no válida. • ActionScript emite excepciones SyntaxError cuando la clase XMLDocument analiza XML no válido.
TypeError	Se genera la excepción TypeError cuando el tipo real de un operando es diferente del tipo esperado.	Se puede generar una excepción TypeError en las circunstancias siguientes: • No se ha podido forzar el tipo de parámetro formal de un parámetro real de una función o un método. • Se ha asignado un valor a una variable y no se puede forzar el tipo de la variable. • El lado derecho del operador `is` o `instanceof` no es un tipo válido. • La palabra clave `super` se utiliza de manera no permitida. • Una consulta de propiedad da como resultado más de una vinculación y, por consiguiente, resulta ambigua. • Se ha llamado a un método en un objeto incompatible. Por ejemplo, se genera una excepción TypeError si se inserta un método de la clase RegExp en un objeto genérico y luego se llama.
URIError	Se genera la excepción URIError cuando una de las funciones de gestión de URI global se utiliza de manera incompatible con esta definición.	Se puede generar una excepción URIError en las circunstancias siguientes: Se especifica un URI no válido para una función de la API de Flash Player que espera un URI válido, como `Socket.connect()`.
VerifyError	Se genera una excepción VerifyError cuando se detecta un archivo SWF mal formado o dañado.	Cuando un archivo SWF carga otro archivo SWF, el archivo SWF principal puede capturar una excepción VerifyError generada por el archivo SWF cargado.

Clases Error del paquete flash.error

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El paquete flash.error contiene clases Error que se consideran partes integrantes de la API del motor de ejecución de Flash. A diferencia de las clases Error descritas anteriormente, el paquete flash.error comunica eventos de error específicos de los motores de ejecución de Flash (Flash Player o Adobe AIR).

Nombre de clase	Descripción	Notas
EOFError	La excepción EOFError se emite al intentar leer más allá del final de los datos disponibles.	Por ejemplo, se emitirá una excepción EOFError si se llama a uno de los métodos de lectura de la interfaz IDataInput y no hay datos suficientes para satisfacer la petición de lectura.
IllegalOperationError	Se genera una excepción IllegalOperationError si un método no se implementa, o bien si la implementación no abarca el uso actual.	Ejemplos de excepciones de errores de operación no permitida: • Una clase base, como DisplayObjectContainer, proporciona mayor funcionalidad de la que puede admitir el escenario. Por ejemplo, si se intenta obtener o establecer una máscara en el escenario (mediante `stage.mask`), el motor de ejecución de Flash generará un error IllegalOperationError con el mensaje "La clase Stage no implementa esta propiedad o método". • Una subclase hereda un método que no requiere y que no desea admitir. • Se llama a determinados métodos de accesibilidad al compilar Flash Player sin compatibilidad de accesibilidad. • Las funciones exclusivas de edición se llaman desde una versión de Flash Player en tiempo de ejecución. • Se intenta establecer el nombre de un objeto que se coloca en la línea de tiempo.
IOError	Se genera una excepción IOError cuando se produce algún tipo de excepción de E/S.	Este error se obtiene, por ejemplo, si se intenta realizar una operación de lectura y escritura en un socket no conectado o que haya perdido la conexión.

Nombre de clase	Descripción	Notas
MemoryError	Se genera una excepción MemoryError cuando falla una petición de asignación de memoria.	De forma predeterminada, la máquina virtual de ActionScript (ActionScript Virtual Machine 2) no impone ningún límite en cuanto a la memoria que puede asignar un programa de ActionScript. En un sistema escritorio, los errores de asignación de memoria no son frecuentes. Se generará un error cuando el sistema no pueda asignar la memoria necesaria para una operación. De este modo, en un sistema de escritorio, esta excepción es poco habitual, a menos que la petición de asignación sea muy grande. Por ejemplo, una petición de 3.000 millones de bytes resulta imposible, ya que un programa para Microsoft® Windows® de 32 bits solo puede acceder a 2 GB del espacio de direcciones.
ScriptTimeoutError	Se genera una excepción ScriptTimeoutError cuando se alcanza un intervalo de tiempo de espera del script de 15 segundos. Si se captura una excepción ScriptTimeoutError, se puede gestionar el tiempo de espera del script con mayor comodidad. Si no hay controlador de excepciones, la excepción no capturada mostrará un cuadro de diálogo con un mensaje de error.	Para evitar que un desarrollador malintencionado capture la excepción y permanezca en un bucle infinito, solo se puede capturar la primera excepción ScriptTimeoutError generada durante la ejecución de un script determinado. El código del usuario no podrá capturar una excepción ScriptTimeoutError posterior; esta se dirigirá inmediatamente al controlador de excepciones.
StackOverflowError	La excepción StackOverflowError se genera cuando se agota la pila disponible para el script.	Una excepción StackOverflowError puede indicar que se ha producido recursión infinita.

Ejemplo de gestión de errores: Aplicación CustomErrors

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación CustomErrors muestra técnicas para trabajar con errores personalizados al crear una aplicación. Estas técnicas son:

- Validar un paquete XML
- Escribir un error personalizado
- Generar errores personalizados
- Notificar un error a los usuarios cuando se genere

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de aplicación CustomErrors pueden encontrarse en la carpeta Samples/CustomError. La aplicación consta de los siguientes archivos:

Archivo	Descripción
CustomErrors.mxml o CustomErrors.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/errors/ApplicationError.as	Una clase que constituye la clase de error base para las clases FatalError y WarningError.

Archivo	Descripción
com/example/programmingas3/errors/FatalError.as	Una clase que define un error FatalError emitido por la aplicación. Esta clase amplía la clase ApplicationError personalizada.
com/example/programmingas3/errors/Validator.as	Una clase que define un método único que valida un paquete XML para empleados proporcionado por el usuario.
com/example/programmingas3/errors/WarningError.as	Una clase que define un error WarningError que puede generar la aplicación. Esta clase amplía la clase ApplicationError personalizada.

Información general sobre la aplicación CustomErrors

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se carga la aplicación, se llama al método `initApp()` para las aplicaciones de Flex o el código de la línea de tiempo (sin función) se ejecuta en Flash Professional. Este código define un paquete XML de ejemplo que comprobará la clase Validator. Se ejecuta el siguiente código:

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

El paquete XML se muestra después en una instancia de componente TextArea en el escenario, lo que permite modificar el paquete antes de intentar su revalidación.

Cuando el usuario hace clic en el botón Validate, se llama al método `validateData()`. Este método valida el paquete XML para empleados con el método `validateEmployeeXML()` de la clase Validator. El código siguiente muestra el método `validateData()`:

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

En primer lugar, se crea un objeto XML temporal con el contenido de la instancia de componente TextArea `xmlText`. A continuación, el método `validateEmployeeXML()` de la clase Validator personalizada (com.example.programmingas3/errors/Validator.as) se llama y transmite el objeto XML temporal como parámetro. Si el paquete XML es válido, la instancia de componente Label `status` muestra un mensaje de operación realizada correctamente y se cierra la aplicación. Si el método `validateEmployeeXML()` genera un error personalizado (es decir, FatalError, WarningError o Error de tipo genérico), la sentencia `catch` correspondiente llama al método `showFatalError()`, `showWarningError()` o `showGenericError()` y lo ejecuta. Cada uno de estos métodos muestra un mensaje apropiado en un área de texto denominada `statusText` para notificar al usuario el error concreto que se ha producido. Los métodos también actualizan la instancia de componente Label `status` con un mensaje determinado.

Si se produce un error grave durante el intento de validar el paquete XML para empleados, el mensaje de error se muestra en el área de texto `statusText` y el componente TextArea `xmlText` y la instancia del componente Button `validateBtn` se deshabilitan, tal como se indica en el código siguiente:

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

Si se produce un error de advertencia en lugar de un error grave, el mensaje de error se muestra en la instancia de TextArea `statusText`, pero las instancias del componente TextField y Button `xmlText` no se deshabilitan. El método `showWarningError()` muestra el mensaje de error personalizado en el área de texto `statusText`. El mensaje también pregunta al usuario si desea continuar con la validación del XML o cancelar la ejecución del script. El siguiente fragmento de código muestra el método `showWarningError()`:

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

Cuando el usuario hace clic en el botón Sí o No, se llama al método`closeHandler()`. El extracto siguiente muestra el método `closeHandler()`:

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

Si el usuario decide cancelar la ejecución del script haciendo clic en Sí, se genera un error de tipo FatalError, lo que provoca el cierre de la aplicación.

Creación de un validador personalizado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Validator personalizada contiene un solo método, `validateEmployeeXML()`. El método `validateEmployeeXML()` solo procesa un único argumento, `employee`, que es el paquete XML que se desea validar. El método `validateEmployeeXML()` funciona del modo siguiente:

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

Para que se valide un empleado, este debe pertenecer única y exclusivamente a un centro de coste. Si el empleado no pertenece a ningún centro de coste, el método genera un error de tipo FatalError, que se propaga hasta el método `validateData()` del archivo de aplicación principal. Si el empleado pertenece a más de un centro de coste, se genera un error WarningError. La última comprobación en el validador de XML consiste en confirmar que el usuario tiene exactamente un número de seguridad social definido (el nodo `ssn` del paquete XML). Si no hay exactamente un nodo `ssn`, se genera un error de tipo FatalError.

Se pueden añadir otras comprobaciones al método `validateEmployeeXML()`; por ejemplo, para asegurarse de que el nodo `ssn` contiene un número válido, o bien el empleado tiene definidos al menos un número de teléfono y una dirección de correo electrónico, y los dos valores son válidos. Asimismo, se pueden modificar los datos XML para que cada empleado tenga un ID único y especifique el ID de su responsable.

Definición de la clase ApplicationError

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ApplicationError es la clase base para las clases FatalError y WarningError. La clase ApplicationError amplía la clase Error y define sus propios métodos y propiedades, entre los que se incluye la definición de ID y gravedad de errores, así como objetos XML que contienen códigos y mensajes de error personalizados. Esta clase también define dos constantes estáticas que se usan para definir la gravedad de cada tipo de error.

El método del constructor de la clase ApplicationError es el siguiente:

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

Cada nodo de error del objeto XML contiene un código numérico único y un mensaje de error. Los mensajes de error pueden consultarse por su código de error mediante E4X, tal como se muestra en el método `getMessageText()` siguiente:

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

El método `getMessageText()` procesa un solo argumento de tipo entero, `id`, y devuelve una cadena. El argumento `id` es el código de error que debe consultar el error. Por ejemplo, al pasar el `id` 9001, se recupera un error que indica que los empleados deben asignarse a un solo centro de coste. Si hay más de un error con el mismo código, ActionScript solo devuelve el mensaje de error para el primer resultado encontrado (`message[0]` del objeto XMLList devuelto).

El siguiente método de esta clase, `getTitle()`, no procesa ningún parámetro y devuelve un valor de cadena que contiene el ID de este error específico. Este valor se emplea para ayudar a identificar fácilmente el error exacto, producido durante la validación del paquete XML. El siguiente fragmento de código muestra el método `getTitle()`:

```
public function getTitle():String
{
    return "Error #" + id;
}
```

El último método de la clase ApplicationError es `toString()`. Este método sustituye la función definida en la clase Error, de manera que se puede personalizar la presentación del mensaje de error. El método devuelve una cadena que identifica el mensaje y el número de error específicos que se han producido.

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

Definición de la clase FatalError

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase FatalError amplía la clase personalizada ApplicationError y define tres métodos: el constructor FatalError, `getTitle()` y `toString()`. El primer método (el constructor de FatalError) procesa un solo argumento de tipo entero (`errorID`), y establece la gravedad del error con los valores de constantes estáticas definidas en la clase ApplicationError. El mensaje de error específico se obtiene llamando al método `getMessageText()` de la clase ApplicationError. El constructor de FatalError es el siguiente:

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

El siguiente método de la clase FatalError, `getTitle()`, sustituye el método `getTitle()` definido anteriormente en la clase ApplicationError, y añade el texto "-- FATAL" al título para informar al usuario de que se ha producido un error grave. El método `getTitle()` funciona del modo siguiente:

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

El último método de esta clase, `toString()`, sustituye el método `toString()` definido en la clase ApplicationError. El método `toString()` funciona es:

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

Definición de la clase WarningError

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase WarningError amplía la clase ApplicationError y es casi idéntica a la clase FatalError, salvo por un par de cambios en cadenas poco importantes. La gravedad de los errores se establece en ApplicationError.WARNING en lugar de ApplicationError.FATAL, tal como se muestra en el código siguiente:

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

Capítulo 5: Uso de expresiones regulares

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una expresión regular describe un patrón que se utiliza para buscar y manipular texto coincidente en cadenas. Las expresiones regulares parecen cadenas, pero pueden incluir códigos especiales para describir patrones y repeticiones. Por ejemplo, la siguiente expresión regular detecta una cadena que empiece por el carácter A seguido de uno o más dígitos:

```
/A\d+/
```

Los siguientes temas describen la sintaxis básica para crear expresiones regulares. Sin embargo, las expresiones regulares pueden tener muchas complejidades y matices. Se puede encontrar información detallada sobre expresiones regulares en Internet y en librerías. Hay que tener en cuenta que los distintos entornos de programación implementan las expresiones regulares de distinta manera. ActionScript 3.0 implementa la definición de las expresiones regulares incluida en la especificación del lenguaje ECMAScript edición 3 (ECMA-262).

Más temas de ayuda

RegExp

Fundamentos de la utilización de expresiones regulares

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una expresión regular describe un patrón de caracteres. Las expresiones regulares se suelen utilizar para comprobar que un valor de texto se ajusta a un patrón específico (por ejemplo, para comprobar que un número de teléfono especificado por el usuario tiene el número de dígitos correcto) o para sustituir partes de un valor de texto que coincidan con un patrón específico.

Las expresiones regulares pueden ser sencillas. Por ejemplo, se puede desear confirmar que una cadena específica coincide con "ABC" o reemplazar cada instancia de "ABC" en una cadena por otro texto. En este caso, se puede utilizar la siguiente expresión regular, que define el patrón formado por las letras A, B y C en secuencia:

```
/ABC/
```

Hay que tener en cuenta que el literal de expresión regular se delimita con el carácter barra diagonal (`/`).

Los patrones de expresión regular también pueden ser complejos y, a veces, aparentemente crípticos, como la siguiente expresión para detectar una dirección de correo electrónico válida:

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

Normalmente las expresiones regulares se utilizan para buscar patrones en cadenas y para reemplazar caracteres. En estos casos, se creará un objeto de expresión regular y se utilizará como parámetro de uno de varios métodos de la clase String. Los siguientes métodos de la clase String toman como parámetros expresiones regulares: `match()`, `replace()`, `search()` y `split()`. Para más información sobre estos métodos, consulte “Búsqueda de patrones en cadenas y sustitución de subcadenas” en la página 17

La clase RegExp incluye los métodos siguientes: `test()` y `exec()`. Para más información, consulte “Métodos para utilizar expresiones regulares con cadenas” en la página 91.

Conceptos y términos importantes

La siguiente lista de referencias contiene términos importantes que resultan relevantes para esta función:

Carácter de escape Carácter que indica que el carácter que sigue debe tratarse como un metacarácter en lugar de como un carácter literal. En sintaxis de expresiones regulares, el carácter de barra diagonal inversa (\) es el carácter de escape, por lo que una barra diagonal inversa seguida de otro carácter se interpretará como un código especial en lugar de interpretarse literalmente.

Indicador Carácter que especifica alguna opción sobre cómo debe utilizarse el patrón de expresión regular (p. ej., distinguir entre caracteres en mayúsculas y caracteres en minúsculas.

Metacarácter Carácter que tiene un significado especial en un patrón de expresión regular, en lugar de representar literalmente dicho carácter en el patrón.

Cuantificador Carácter (o conjunto de caracteres) que indica cuántas veces debe repetirse una parte del patrón. Por ejemplo, se puede utilizar un cuantificador para especificar que un código postal de Estados Unidos debe contener cinco o nueve números.

Expresión regular Sentencia de programa que define un patrón de caracteres que se puede utilizar para confirmar si otras cadenas coinciden con ese patrón o para sustituir partes de una cadena.

Sintaxis de las expresiones regulares

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En esta sección se describen todos los elementos de sintaxis de las expresiones regulares de ActionScript. Sin embargo, las expresiones regulares pueden tener muchas complejidades y matices. Se puede encontrar información detallada sobre expresiones regulares en Internet y en librerías. Hay que tener en cuenta que los distintos entornos de programación implementan las expresiones regulares de distinta manera. ActionScript 3.0 implementa la definición de las expresiones regulares incluida en la especificación del lenguaje ECMAScript edición 3 (ECMA-262).

Generalmente, se utilizan expresiones regulares que detectan patrones más complicados que una simple cadena de caracteres. Por ejemplo, la siguiente expresión regular define el patrón formado por la secuencia de letras A, B y C seguida de un dígito:

```
/ABC\d/
```

El código `\d` representa "cualquier dígito". El carácter barra diagonal inversa (`\`) se denomina carácter de escape y, combinado con el carácter siguiente (en este caso, la letra d), tiene un significado especial en la expresión regular.

La siguiente expresión regular define el patrón de las letras ABC seguidas de un número arbitrario de dígitos (obsérvese el asterisco):

```
/ABC\d*/
```

El asterisco (`*`) es un *metacarácter*. Un metacarácter es un carácter que tiene un significado especial en las expresiones regulares. El asterisco es un tipo específico de metacarácter denominado *cuantificador* que se utiliza para cuantificar la cantidad de repeticiones de un carácter o grupo de caracteres. Para más información, consulte “Cuantificadores” en la página 83.

Además del patrón, una expresión regular puede contener indicadores, que especifican cómo debe detectarse la expresión regular. Por ejemplo, la siguiente expresión regular utiliza el indicador `i`, que especifica que la expresión regular no distinguirá mayúsculas de minúsculas en las cadenas coincidentes:

```
/ABC\d*/i
```

Para más información, consulte "Indicadores y propiedades" en la página 88.

Se pueden utilizar expresiones regulares con los siguientes métodos de la clase String: `match()`, `replace()` y `search()`. Para más información sobre estos métodos, consulte "Búsqueda de patrones en cadenas y sustitución de subcadenas" en la página 17

Creación de una instancia de expresión regular

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Hay dos maneras de crear una instancia de expresión regular. Una consiste en utilizar caracteres de barra diagonal (`/`) para delimitar la expresión regular; la otra utiliza el constructor `new`. Por ejemplo, las dos expresiones regulares siguientes son equivalentes:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

Las barras diagonales delimitan un literal de expresión regular de la misma manera que las comillas delimitan un literal de cadena. La parte de la expresión regular delimitada por las barras diagonales define el *patrón*. La expresión regular también puede incluir *indicadores* a continuación de la última barra de delimitación. Se considera que estos indicadores forman parte de la expresión regular, pero son independientes del patrón.

Al utilizar el constructor `new` se pueden utilizar dos cadenas para definir la expresión regular. La primera cadena define el patrón y la segunda cadena define los indicadores, como en el siguiente ejemplo:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

Si se incluye una barra diagonal *en* una expresión regular definida con los delimitadores de barra diagonal, hay que escribir inmediatamente antes de dicha barra diagonal el carácter de escape de barra diagonal inversa (`\`). Por ejemplo, la siguiente expresión regular detecta el patrón `1/2`:

```
var pattern:RegExp = /1\/2/;
```

Para incluir comillas *en* una expresión regular definida con el constructor `new`, hay que añadir el carácter de escape de barra diagonal inversa (`\`) antes de las comillas (de la misma manera que al definir un literal de cadena). Por ejemplo, las siguientes expresiones regulares detectan el patrón `eat at "joe's"`:

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

No se debe utilizar el carácter de escape barra diagonal inversa con comillas en expresiones regulares definidas con los delimitadores de barra diagonal. De forma similar, no se debe utilizar el carácter de escape con barras diagonales en expresiones regulares definidas con el constructor `new`. Las siguientes expresiones regulares son equivalentes y definen el patrón `1/2 "joe's"`:

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

Además, en una expresión definida con el constructor `new`, para utilizar una metasecuencia que comience con el carácter barra diagonal inversa (`\`) como, por ejemplo, `\d` (que coincide con cualquier dígito), debe escribirse el carácter barra diagonal inversa dos veces:

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

En este caso hay que escribir el carácter barra diagonal inversa dos veces, ya que el primer parámetro del método constructor `RegExp()` es una cadena y en un literal de cadena hay que escribir un carácter barra diagonal inversa dos veces para que se reconozca como un solo carácter de barra diagonal inversa.

En las secciones siguientes se describe la sintaxis para definir patrones de expresiones regulares.

Para más información sobre los indicadores, consulte "Indicadores y propiedades" en la página 88.

Caracteres, metacaracteres y metasecuencias

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La expresión regular más sencilla es la que detecta una secuencia de caracteres, como en el siguiente ejemplo:

```
var pattern:RegExp = /hello/;
```

No obstante, los siguientes caracteres, denominados metacaracteres, tienen significados especiales en las expresiones regulares:

```
^ $ \ . * + ? ( ) [ ] { } |
```

Por ejemplo, la siguiente expresión regular detecta la letra A seguida de cero o más instancias de la letra B (el metacarácter asterisco indica esta repetición), seguidas de la letra C:

```
/AB*C/
```

Para incluir un metacarácter sin su significado especial en un patrón de expresión regular, hay que utilizar el carácter de escape de barra diagonal inversa (`\`). Por ejemplo, la siguiente expresión regular detecta la letra A seguida de la letra B, seguida de un asterisco, seguido de la letra C:

```
var pattern:RegExp = /AB\*C/;
```

Una *metasecuencia*, al igual que un metacarácter, tiene un significado especial en una expresión regular. Las metasecuencias están formadas por más de un carácter. Las secciones siguientes proporcionan detalles sobre la utilización de metacaracteres y metasecuencias.

Metacaracteres

En la tabla siguiente se muestra un resumen de los metacaracteres que se pueden utilizar en las expresiones regulares:

Metacarácter	Descripción
^ (intercalación)	Detecta el principio de la cadena. Con el indicador `m` (`multiline`) establecido, el signo de intercalación también detecta el inicio de cada línea (consulte "Indicadores y propiedades" en la página 88). Hay que tener en cuenta que cuando se utiliza al principio de una clase de caracteres, el signo de intercalación indica negación, no el principio de una cadena. Para más información, consulte "Clases de caracteres" en la página 82.
$ (signo dólar)	Detecta el final de la cadena. Con el indicador `m` (`multiline`) establecido, `$` detecta la posición anterior a un carácter de nueva línea (`\n`). Para más información, consulte "Indicadores y propiedades" en la página 88.
\ (barra diagonal inversa)	Omite el significado especial de los metacaracteres. También debe utilizarse el carácter barra diagonal inversa si se desea utilizar un carácter barra diagonal en un literal de expresión regular, como `/1\/2/` (para detectar el carácter 1, seguido del carácter barra diagonal, seguido del carácter 2).
. (punto)	Detecta cualquier carácter individual. El punto detecta un carácter de nueva línea (`\n`) solo si está establecido el indicador `s` (`dotall`). Para más información, consulte "Indicadores y propiedades" en la página 88.
* (asterisco)	Detecta el elemento anterior repetido cero o más veces. Para más información, consulte "Cuantificadores" en la página 83.

Metacarácter	Descripción
+ (más)	Detecta el elemento anterior repetido una o más veces. Para más información, consulte "Cuantificadores" en la página 83.
? (signo de interrogación)	Detecta el elemento anterior repetido cero veces o una sola vez. Para más información, consulte "Cuantificadores" en la página 83.
(y)	Define grupos dentro de la expresión regular. Los grupos se pueden utilizar para: • Para limitar el ámbito del alternador |: `/(a|b|c)d/` • Para definir el ámbito de un cuantificador: `/(walla.){1,2}/` • En referencias a elementos detectados previamente. Por ejemplo, en la siguiente expresión regular, `\1` detectará lo que se haya detectado en el primer grupo delimitado con paréntesis del patrón: • `/(\w*) se repite: \1/` Para más información, consulte "Grupos" en la página 85.
[y]	Define una clase de caracteres, que especifica posibles coincidencias para un solo carácter: `/[aeiou]/` detecta cualquiera de los caracteres especificados. En las clases de caracteres se utiliza un guion (-)para designar un rango de caracteres: `/[A-Z0-9]/` detecta cualquier letra mayúscula (A a Z) o cualquier dígito (0 a 9). Además, se debe insertar una barra diagonal inversa para omitir el significado especial de los caracteres] y - caracteres: `/[+\-]\d+/` detecta + o - antes de uno o más dígitos. En las clases de caracteres, los caracteres que normalmente metacaracteres se tratan como caracteres normales (no metacaracteres), sin necesidad de utilizar una barra diagonal inversa: `/[$]/£` detecta `$` o £. Para más información, consulte "Clases de caracteres" en la página 82.
| *(barra vertical)*	Se utiliza para la alternancia, a fin de detectar la parte de la izquierda o la parte de la derecha: `/abc|xyz/` detecta `abc` o `xyz`.

Metasecuencias

Las metasecuencias son secuencias de caracteres que tienen un significado especial en un patrón de expresión regular. En la tabla siguiente se describen estas metasecuencias:

Metasecuencia	Descripción
{*n*} {*n*,} y {*n*,*n*}	Especifica un cuantificador numérico o un rango de cuantificadores para el elemento anterior: `/A{27}/` detecta el carácter `A` repetido `27` veces. `/A{3,}/` detecta el carácter `A` repetido `3` o más veces. `/A{3,5}/` detecta el carácter `A` repetido entre `3` y `5` veces. Para más información, consulte "Cuantificadores" en la página 83.
`\b`	Detecta la posición entre un carácter de palabra y un carácter de otro tipo. Si el primer o el último carácter de la cadena es un carácter de palabra, también detecta el principio o el final de la cadena.
`\B`	Detecta la posición entre dos caracteres de palabra. También detecta la posición entre dos caracteres de otro tipo.

Metasecuencia	Descripción
`\d`	Detecta un dígito decimal.
`\D`	Detecta cualquier carácter que no sea un dígito.
`\f`	Detecta un carácter de salto de página.
`\n`	Detecta el carácter de nueva línea.
`\r`	Detecta el carácter de retorno.
`\s`	Detecta cualquier carácter de espacio en blanco (un carácter de espacio, tabulación, nueva línea o retorno).
`\S`	Detecta cualquier carácter que no sea un espacio en blanco.
`\t`	Detecta el carácter de tabulación.
`\u`*nnnn*	Detecta el carácter Unicode con el código de carácter especificado por el número hexadecimal *nnnn*. Por ejemplo, `\u263a` es el carácter smiley.
`\v`	Detecta un carácter de avance vertical.
`\w`	Detecta un carácter de palabra (`AZ`–, `az`–, `0-9` o `_`). Hay que tener en cuenta que `\w` no detecta caracteres que no sean los del idioma inglés, como `é`, `ñ` o `ç` .
`\W`	Detecta cualquier carácter que no sea un carácter de palabra.
`\\x`*nn*	Detecta el carácter con el valor ASCII especificado, definido por el número hexadecimal *nn*.

Clases de caracteres

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar clases de caracteres para especificar una lista de caracteres con el fin de detectar una posición en la expresión regular. Las clases de caracteres se definen con corchetes (`[` y `]`). Por ejemplo, la siguiente expresión regular define una clase de caracteres que detecta `bag`, `beg`, `big`, `bog` o `bug`:

```
/b[aeiou]g/
```

Secuencias de escape en clases de caracteres

La mayoría de los metacaracteres y las metasecuencias que normalmente tienen significados especiales en una expresión regular *no* tienen el mismo significado dentro de una clase de caracteres. Por ejemplo, en una expresión regular, el asterisco se utiliza para indicar repetición, pero en una clase de caracteres no tiene este significado. La siguiente clase de caracteres detecta el asterisco literalmente, junto con cualquiera de los otros caracteres indicados:

```
/[abc*123]/
```

Sin embargo, los tres caracteres mostrados en la tabla siguiente funcionan como metacaracteres (tienen un significado especial) en las clases de caracteres:

Metacarácter	Significado en las clases de caracteres
`]`	Define el final de la clase de caracteres.
`-`	Define un rango de caracteres (consulte la siguiente sección "Rangos de caracteres en clases de caracteres").
`\`	Define metasecuencias y omite el significado especial de los metacaracteres.

Para que se reconozca cualquiera de estos caracteres como un carácter literal (sin su significado especial de metacarácter), hay que escribir el carácter de escape barra diagonal inversa inmediatamente antes del carácter. Por ejemplo, la siguiente expresión regular incluye una clase de caracteres que detecta cualquiera de los cuatro símbolos (`$`, `\`, `]` o `-`):

```
/[$\\\]\-]/
```

Además de los metacaracteres que conservan su significado especial, las siguientes metasecuencias funcionan como metasecuencias en las clases de caracteres:

Metasecuencia	Significado en las clases de caracteres
`\n`	Detecta un carácter de nueva línea.
`\r`	Detecta un carácter de retorno.
`\t`	Detecta un carácter de tabulación.
`\u`***nnnn***	Detecta el carácter con el valor de código Unicode especificado (definido por el número hexadecimal *nnnn*).
`\\x`***nn***	Detecta el carácter con el valor ASCII especificado (definido por el número hexadecimal *nn*).

Otros metacaracteres y metasecuencias de expresión regular se tratan como caracteres normales dentro de una clase de caracteres.

Rangos de caracteres en clases de caracteres

Se utiliza un guion para especificar un rango de caracteres, como `A-Z`, `a-z` o `0-9`. Estos caracteres deben constituir un rango válido en el conjunto de caracteres. Por ejemplo, la siguiente clase de caracteres detecta cualquiera de los caracteres del intervalo `a-z` o cualquier dígito:

```
/[a-z0-9]/
```

También se puede utilizar el código de carácter ASCII `\\x`*nn* para especificar un rango por valor ASCII. Por ejemplo, la siguiente clase de caracteres detecta cualquier carácter de un conjunto de caracteres extendidos ASCII (como `é` y `ê`):

```
\\x
```

Clases de caracteres denegados

Si se utiliza un carácter de intercalación (`^`) al principio de una clase de caracteres, deniega los caracteres de la clase: detectará cualquier carácter que no esté en la clase. La siguiente clase de caracteres detecta cualquier carácter *salvo* una letra minúscula (`a-z–`) o un dígito:

```
/[^a-z0-9]/
```

Para indicar la negación se debe escribir el carácter de intercalación (`^`) al *principio* de una clase de caracteres. De lo contrario, solo se añade el carácter de intercalación a los caracteres de la clase de caracteres. Por ejemplo, la siguiente clase de caracteres detecta cualquier carácter de un conjunto de caracteres de símbolo, incluido el signo de intercalación:

```
/[!.,#+*%$&^]/
```

Cuantificadores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los cuantificadores se utilizan para especificar repeticiones de caracteres o secuencias en patrones, de la manera siguiente:

Metacarácter de cuantificador	Descripción
* (asterisco)	Detecta el elemento anterior repetido cero o más veces.
+ (más)	Detecta el elemento anterior repetido una o más veces.
? (signo de interrogación)	Detecta el elemento anterior repetido cero veces o una sola vez.
{*n*} {*n*,} y {*n*,*n*}	Especifica un cuantificador numérico o un rango de cuantificadores para el elemento anterior: /A{27}/ detecta el carácter A repetido 27 veces. /A{3,}/ detecta el carácter A repetido 3 o más veces. /A{3,5}/ detecta el carácter A repetido entre 3 y 5 veces.

Se puede aplicar un cuantificador a un solo carácter, a una clase de caracteres o a un grupo:

- /a+/ detecta el carácter a repetido una o más veces.
- /\d+/ detecta uno o más dígitos.
- /[abc]+/ detecta una repetición de uno o más caracteres, cada uno de las cuales puede sera, b o c.
- /(very,)*/ detecta la palabra very seguida de una coma y un espacio repetido cero o más veces.

Se pueden utilizar cuantificadores dentro de grupos delimitados por paréntesis que tengan cuantificadores aplicados. Por ejemplo, el siguiente cuantificador detecta cadenas como word y word-word-word:

```
/\w+(-\w+)*/
```

De manera predeterminada, las expresiones regulares realizan lo que se conoce como una *detección de la coincidencia más larga posible (greedy matching).* Cualquier subpatrón de la expresión regular (como .*) intentará detectar la mayor cantidad posible de caracteres en la cadena antes de pasar a la siguiente parte de la expresión regular. Por ejemplo, considérense la expresión regular y la cadena siguientes:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

La expresión regular detecta toda la cadena:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Sin embargo, si solo se desea detectar una agrupación ... , se puede hacer lo siguiente:

```
<p>Paragraph 1</p>
```

Añadir un signo de interrogación (?) a continuación de cualquier cuantificador para convertirlo en un cuantificador *perezoso*, que detecta la coincidencia más corta posible. Por ejemplo, la siguiente expresión regular, que utiliza un cuantificador de este tipo, *? , detecta seguido del mínimo número posible de caracteres, seguidos de :

```
/<p>.*?<\/p>/
```

Hay que tener en cuenta los siguientes aspectos sobre los cuantificadores:

- Los cuantificadores {0} y {0,0} no excluyen un elemento de una coincidencia.
- No se deben combinar varios cuantificadores, como en /abc+*/.
- El punto (.) no abarca líneas a menos que se establezca el indicador s (dotall), aunque esté seguido de un cuantificador *. Por ejemplo, considérese el fragmento de código siguiente:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                 Multiline</p>
```

Para más información, consulte "Indicadores y propiedades" en la página 88.

Alternancia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El carácter `|` (barra vertical) se utiliza en una expresión regular para que el motor de expresiones regulares considere alternativas para la detección. Por ejemplo, la siguiente expresión regular detecta cualquiera de las palabras `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Se pueden utilizar paréntesis para definir grupos a fin de restringir el ámbito del alternador |. La siguiente expresión regular detecta `cat` seguida de `nap` o `nip`:

```
var pattern:RegExp = /cat(nap|nip)/;
```

Para más información, consulte "Grupos" en la página 85.

Las dos expresiones regulares siguientes, una con el alternador `|` y la otra con una clase de caracteres (definida con `[` y `]`), son equivalentes:

```
/1|3|5|7|9/
/[13579]/
```

Para más información, consulte "Clases de caracteres" en la página 82.

Grupos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede especificar un grupo en una expresión regular utilizando paréntesis, de la manera siguiente:

```
/class-(\d*)/
```

Un grupo es una subsección de un patrón. Los grupos se pueden utilizar para:

- Aplicar un cuantificador a más de un carácter.
- Delimitar subpatrones que debe aplicarse mediante alternancia (utilizando el carácter `|`).
- Capturar coincidencias de subcadenas (por ejemplo, utilizando `\1` en una expresión regular para detectar un grupo detectado previamente o utilizando `$1` de forma similar en el método `replace()` de la clase String).

En las secciones siguientes se proporcionan detalles sobre estos usos de los grupos.

Uso de grupos con cuantificadores

Si no se utiliza un grupo, un cuantificador se aplica al carácter o la clase de caracteres que lo precede, como se indica a continuación:

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

No obstante, se puede utilizar un grupo para aplicar un cuantificador a más de un carácter o clase de caracteres:

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as ababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

Para más información sobre los cuantificadores, consulte "Cuantificadores" en la página 83.

Uso de los grupos con el carácter alternador (|)

Se pueden utilizar grupos para definir el grupo de caracteres al que se desea aplicar un carácter alternador (`|`), de la manera siguiente:

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

Uso de grupos para capturar coincidencias de subcadenas

Si se define un grupo delimitado por paréntesis estándar en un patrón, se puede hacer referencia al mismo en una parte posterior de la expresión regular. Esto se denomina *referencia a un elemento detectado previamente (backreference)* y estos tipos de grupos se denominan *grupos de captura*. Por ejemplo, en la siguiente expresión regular, la secuencia `\1` detectará la subcadena que se haya detectado en el primer grupo de captura delimitado con paréntesis:

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

Se pueden especificar hasta 99 de estas referencias a elementos detectados previamente escribiendo `\1`, `\2`, ..., `\99`.

De forma similar, en el método `replace()` de la clase String, se puede utilizar `$1$99`– para insertar subcadenas coincidentes detectadas con un grupo de captura en la cadena de sustitución:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

Además, si se utilizan grupos de captura, el método `exec()` de la clase RegExp y el método `match()` de la clase String devuelven subcadenas que detectan los grupos de captura:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

Uso de grupos que no capturan y grupos de búsqueda hacia delante

Un grupo que no captura es un grupo que solo se utiliza para agrupar; no se "almacena" ni proporciona referencias numeradas de elementos detectados previamente. Para definir grupos que no capturan se utiliza `(?: y )`, de la manera siguiente:

```
var pattern = /(?:com|org|net);
```

Por ejemplo, véase la diferencia entre colocar `(com|org)` en grupo de captura y en un grupo que no captura (el método `exec()` muestra los grupos de captura después de la coincidencia completa):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

Un tipo especial de grupo que no captura es el *grupo de búsqueda hacia delante*, del cual hay dos tipos: el *grupo de búsqueda positiva hacia delante* y el *grupo de búsqueda negativa hacia delante*.

Para definir un grupo de búsqueda positiva hacia delante, que especifica que el subpatrón del grupo debe coincidir en la posición, se utiliza `(?= y )`. No obstante, la parte de la cadena que coincide con el grupo de búsqueda positiva hacia delante puede coincidir con los demás patrones de la expresión regular. Por ejemplo, como `(?=e)` es un grupo de búsqueda positiva hacia delante en el código siguiente, el carácter `e` que detecta puede ser detectado por una parte posterior de la expresión regular, en este caso, el grupo de captura, `\w*)`:

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Para definir un grupo de búsqueda negativa hacia delante, que especifica que el subpatrón del grupo `no` debe coincidir en la posición, se utiliza `(?!` y *)*. Por ejemplo:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

Uso de grupos con nombre

Un grupo con nombre es un tipo de grupo en una expresión regular al que se le da un identificador designado. Para definir el grupo con nombre se utiliza `(?P<nombre> y )` Por ejemplo, la siguiente expresión regular incluye un grupo con nombre con el identificador denominado `digits`:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

Cuando se utiliza el método `exec()`, se añade un grupo con nombre coincidente como una propiedad del conjunto `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Otro ejemplo, en el que se utilizan dos grupos con nombre, con los identificadores `name` y `dom`:

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

Nota: *los grupos con nombre no forman parte de la especificación del lenguaje ECMAScript. Son una característica añadida de ActionScript 3.0.*

Indicadores y propiedades

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En la tabla siguiente se muestran los cinco indicadores que se pueden establecer para expresiones regulares. Se puede acceder a cada indicador como una propiedad del objeto de expresión regular.

Indicador	Propiedad	Descripción
`g`	`global`	Detecta todas las coincidencias.
`i`	`ignoreCase`	Detecta sin distinguir mayúsculas de minúsculas. Se aplica a los caracteres `A`—`Z` y `a`—`z`, pero no a caracteres extendidos como `É` y `é`.
`m`	`multiline`	Con este indicador establecido, `$` y `^` pueden detectar el principio y el final de una línea, respectivamente.
`s`	`dotall`	Con este indicador establecido, `.` (punto) puede detectar el carácter de nueva línea (`\n`).
`x`	`extended`	Permite utilizar expresiones regulares extendidas. Estas expresiones permiten escribir espacios que se omitirán como parte del patrón. Esto facilita la lectura del código de una expresión regular.

Hay que tener en cuenta que estas propiedades son de solo lectura. Se puede establecer los indicadores (`g`, `i`, `m`, `s`, `x`) al establecer una variable de expresión regular, de la manera siguiente:

```
var re:RegExp = /abc/gimsx;
```

Sin embargo, las propiedades con nombre no se pueden establecer directamente. Por ejemplo, el siguiente código genera un error:

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

De manera predeterminada, los indicadores no se establecen y las propiedades correspondientes se establecen en `false`, a menos que se especifiquen en la declaración de la expresión regular.

Además, las expresiones regulares tienen otras dos propiedades:

- La propiedad `lastIndex` especifica la posición del índice en la cadena que se debe utilizar para la siguiente llamada al método `exec()` o `test()` de una expresión regular.
- La propiedad `source` especifica la cadena que define la parte del patrón de la expresión regular.

El indicador g (global)

Si el indicador `g` (`global`) *no* se incluye, la expresión regular detectará una sola coincidencia. Por ejemplo, si no se incluye el indicador `g` en la expresión regular, el método `String.match()` devuelve una sola cadena coincidente:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Si se establece el indicador `g`, el método `Sting.match()` devuelve varias coincidencias, como se indica a continuación:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

El indicador i (ignoreCase)

De manera predeterminada, las expresiones regulares distinguen mayúsculas de minúsculas al detectar coincidencias. Si se establece el indicador `i` (`ignoreCase`), no se distinguirán mayúsculas de minúsculas. Por ejemplo, la `s` minúscula de la expresión regular no detecta la letra `S` mayúscula, el primer carácter de la cadena:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

Sin embargo, con el indicador `i` establecido, la expresión regular detecta la letra `S` mayúscula:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

El indicador `i` no distingue mayúsculas de minúsculas únicamente para los caracteres `A–Z` y `a–z`, pero sí para caracteres extendidos como `É` y `é` .

El indicador m (multiline)

Si no se establece el indicador `m` (`multiline`), `^` detecta el principio de la cadena y `$` detecta el final de la cadena. Con `m` establecido, estos caracteres detectan el principio y el final de una línea, respectivamente. Considérese la siguiente cadena, que incluye un carácter de nueva línea:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Aunque se establezca el indicador `g` (`global`) en la expresión regular, el método `match()` detecta una sola subcadena, ya que hay una sola coincidencia con `^`, el principio de la cadena. El resultado es:

```
Test
```

A continuación se muestra el mismo código con el indicador `m` establecido:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

Esta vez, el resultado incluye las palabras al principio de ambas líneas:

```
Test,Multiline
```

Hay que tener en cuenta que solo el carácter `\n` indica el final de una línea. Los caracteres siguientes no:

- Carácter de retorno (`\r`)
- Carácter separador de línea Unicode (`\u2028`)

- Carácter separador de párrafo Unicode (`\u2029`)

El indicador s (dotall)

Si no se establece el indicador `s` (`dotall` o "dot all"), un punto (`.`) en un patrón de expresión regular no detectará un carácter de nueva línea (`\n`). Así, en el siguiente ejemplo no se detecta ninguna coincidencia:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Sin embargo, si se establece el indicador `s`, el punto detecta el carácter de nueva línea:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

En este caso, la coincidencia es toda la subcadena entre las etiquetas `<p>`, incluido el carácter de nueva línea:

```
<p>Test
Multiline</p>
```

El indicador x (extended)

Las expresiones regulares pueden ser difíciles de leer, especialmente cuando incluyen muchos metasímbolos y metasecuencias. Por ejemplo:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

Si se utiliza el indicador `x` (`extended`) en una expresión regular, se omitirán los espacios en blanco que se escriban en el patrón. Por ejemplo, la siguiente expresión regular es idéntica a la del ejemplo anterior:

```
/     <p     (>   |   (\s*   [^>]*   >))     .*?     <\/p>   /gix
```

Si se establece el indicador `x` y se desea detectar un carácter de espacio en blanco, se debe escribir una barra diagonal inversa inmediatamente antes del espacio en blanco. Por ejemplo, las dos expresiones normales siguientes son equivalentes:

```
/foo bar/
/foo \ bar/x
```

La propiedad lastIndex

La propiedad `lastIndex` especifica la posición de índice de la cadena en la que debe comenzar la siguiente búsqueda. Esta propiedad afecta a los métodos `exec()` y `test()` llamados en una expresión regular que tiene el indicador `g` establecido en `true`. Por ejemplo, considérese el fragmento de código siguiente:

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

La propiedad `lastIndex` está establecida en 0 de manera predeterminada (para iniciar las búsquedas al principio de la cadena). Después de cada detección, se establece en la posición de índice siguiente a la de la coincidencia. Por tanto, el resultado del código anterior es el siguiente:

```
0
5
11
18
25
36
44
```

Si el indicador `global` está establecido en `false`, los métodos `exec()` y `test()` no utilizan ni establecen la propiedad `lastIndex`.

Los métodos `match()`, `replace()` y `search()` de la clase String inician todas las búsquedas desde el principio de la cadena, independientemente del valor de la propiedad `lastIndex` de la expresión regular utilizada en la llamada al método. (Sin embargo, el método `match()` establece `lastIndex` en 0.)

Se puede establecer la propiedad `lastIndex` para ajustar la posición de inicio en la cadena para la detección de expresiones regulares.

La propiedad source

La propiedad `source` especifica la cadena que define la parte del patrón de una expresión regular. Por ejemplo:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Métodos para utilizar expresiones regulares con cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase RegExp incluye dos métodos: `exec()` y `test()`.

Además de los métodos `exec()` y `test()` de la clase RegExp, la clase String incluye los siguientes métodos que permiten detectar expresiones regulares en cadenas: `match()`, `replace()`, `search()` y `splice()`.

El método test()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `test()` de la clase RegExp comprueba simplemente la cadena suministrada para ver si contiene una coincidencia para la expresión regular, como se indica en el siguiente ejemplo:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

El método exec()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `exec()` de la clase RegExp comprueba la cadena suministrada para detectar una coincidencia con la expresión regular y devuelve un conjunto con lo siguiente:

- La subcadena coincidente
- La subcadena detecta grupos entre paréntesis en la expresión regular

El conjunto también incluye una propiedad `index`, que indica la posición del índice del inicio de la subcadena coincidente.

Por ejemplo, considérese el fragmento de código siguiente:

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

El método `exec()` se utiliza varias veces para detectar varias subcadenas cuando se establece el indicador `g` (`global`) para la expresión regular:

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

Métodos de cadena que utilizan parámetros de tipo RegExp

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los siguientes métodos de la clase String toman como parámetros expresiones regulares: `match()`, `replace()`, `search()` y `split()`. Para más información sobre estos métodos, consulte "Búsqueda de patrones en cadenas y sustitución de subcadenas" en la página 17

Ejemplo de expresiones regulares: Analizador Wiki

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Este ejemplo simple de conversión de texto de Wiki ilustra diversos usos de las expresiones regulares:

- Convertir líneas de texto que coinciden con un patrón de Wiki de origen con las cadenas HTML de salida apropiadas.

- Utilizar una expresión regular para convertir patrones de URL en etiquetas de hipervínculos HTML `<a>`.
- Usar una expresión regular para convertir cadenas con valores monetarios en dólares de EE.UU. (como `"$9,95"`) en cadenas que contienen valores monetarios en euros (como `"8,24 €"`).

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación WikiEditor se encuentran en la carpeta Samples/WikiEditor. La aplicación consta de los siguientes archivos:

Archivo	Descripción
WikiEditor.mxml o WikiEditor.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/regExpExamples/WikiParser.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir patrones de texto de Wiki de origen en la salida HTML equivalente.
com/example/programmingas3/regExpExamples/URLParser.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir cadenas de URL en etiquetas de hipervínculos HTML `<a>`.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	Una clase que incluye métodos que utilizan expresiones regulares para convertir cadenas con valores monetarios en dólares de EE.UU. en cadenas en euros.

Definición de la clase WikiParser

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase WikiParser incluye métodos que convierten texto Wiki de entrada en la salida HTML equivalente. Esta aplicación de conversión de Wiki no es muy sólida, pero ilustra algunos usos útiles de las expresiones regulares para la detección de patrones y la conversión de cadenas.

La función constructora, junto con el método `setWikiData()`, simplemente inicializa una cadena de ejemplo de texto de entrada de Wiki, de la manera siguiente:

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

Cuando el usuario hace clic en el botón Test de la aplicación de ejemplo, la aplicación invoca al método `parseWikiString()` del objeto WikiParser. Este método llama a otros métodos, que a su vez crean la cadena HTML resultante.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

Cada uno de los métodos llamados, `parseBold()`, `parseItalic()`, `linesToParagraphs()` y `parseBullets()` utiliza el método `replace()` de la cadena para sustituir patrones coincidentes, definidos por una expresión regular, a fin de transformar el texto de entrada del Wiki en texto con formato HTML.

Conversión de patrones en negrita y en cursiva

El método `parseBold()` busca un patrón de texto de Wiki en negrita (como `'''foo'''`) y lo transforma en su equivalente HTML (como `<b>foo</b>`), de la manera siguiente:

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Hay que tener en cuenta que la parte `(.?*)` de la expresión regular detecta un número arbitrario de caracteres (`*`) entre los dos patrones delimitadores`'''`. El cuantificador `?` hace que no se detecte la mayor cantidad posible de caracteres, de forma que para una cadena como `'''aaa''' bbb '''ccc'''`, la primera cadena detectada será `'''aaa'''` y no la cadena completa (que empieza y termina con el patrón `'''`).

Los paréntesis de la expresión regular definen un grupo de captura y el método `replace()` hace referencia a este grupo mediante el código `$1` en la cadena de sustitución. El indicador `g` (`global`) de la expresión regular garantiza que el método `replace()` sustituye todas las coincidencias de la cadena (no solo la primera).

El método `parseItalic()` funciona de forma similar al método `parseBold()`, con la diferencia de que busca dos apóstrofes (`''`), no tres, como delimitador del texto en cursiva:

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

Conversión de patrones de viñetas

Como se indica en el siguiente ejemplo, el método `parseBullet()` busca el patrón de línea de viñeta del Wiki (como `* foo`) y la transforma en su equivalente HTML (como `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

El símbolo `^` al principio de la expresión regular detecta el principio de una línea. El indicador `m` (`multiline`) de la expresión regular hace que la expresión regular detecte el símbolo `^` en cada principio de línea, no solo al principio de la cadena.

El patrón `\*` detecta un asterisco (se utiliza la barra diagonal inversa para indicar un asterisco literal en lugar del cuantificador `*`).

Los paréntesis de la expresión regular definen un grupo de captura y el método `replace()` hace referencia a este grupo mediante el código `$1` en la cadena de sustitución. El indicador `g` (`global`) de la expresión regular garantiza que el método `replace()` sustituye todas las coincidencias de la cadena (no solo la primera).

Conversión de patrones de párrafo de Wiki

El método `linesToParagraphs()` convierte cada línea de la cadena de entrada de Wiki en una etiqueta de párrafo HTML, `<p>`. Estas líneas del método eliminan las líneas vacías de la cadena de Wiki de entrada:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Los símbolos ^ y $ hacen que la expresión regular detecte el principio y el final de una línea. El indicador m (multiline) de la expresión regular hace que la expresión regular detecte el símbolo ^ en cada principio de línea, no solo al principio de la cadena.

El método replace() sustituye todas las subcadenas coincidentes (líneas vacías) con una cadena vacía (""). El indicador g (global) de la expresión regular garantiza que el método replace() sustituye todas las coincidencias de la cadena (no solo la primera).

Conversión de direcciones URL en etiquetas HTML <a>

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando el usuario hace clic en el botón Test en la aplicación de ejemplo, si seleccionó la casilla de verificación urlToATag, la aplicación llama al método estático URLParser.urlToATag() para convertir cadenas de dirección URL de la cadena de Wiki de entrada en etiquetas HTML <a>.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

La función constructora RegExp() se utiliza para crear una expresión regular (urlPattern) a partir de varios constituyentes. Estos constituyentes son cadenas que definen partes del patrón de la expresión regular.

La primera parte del patrón de la expresión regular, definida por la cadena protocol, define un protocolo de URL: http:// o ftp://. Los paréntesis definen un grupo que no captura, indicado por el símbolo ?. Esto significa que los paréntesis se utilizan simplemente para definir un grupo para el patrón de alternancia |; el grupo no detectará códigos de elementos detectados previamente ($1, $2, $3) en la cadena de sustitución del método replace().

Los otros elementos constituyentes de la expresión regular utilizan grupos de captura (indicados mediante paréntesis en el patrón), que se utilizan en los códigos de referencia a elementos detectados previamente ($1, $2, $3) en la cadena de sustitución del método replace().

La parte del patrón definida por la cadena urlPart detecta *al menos* uno de los siguientes caracteres: a-z, 0-9, _ o -. El cuantificador + indica que debe detectarse al menos un carácter. \. indica un carácter punto (.) requerido. Y el resto detecta otra cadena que conste al menos de uno de los siguientes caracteres: a-z, 0-9, _ o -.

La parte del patrón definida por la cadena optionalUrlPart detecta *cero o más* de los caracteres siguientes: un punto (. seguido de cualquier número de caracteres alfanuméricos (incluidos _ y -. El cuantificador * indica que deben detectarse cero o más caracteres.

La llamada al método replace() utiliza la expresión regular y crea la cadena HTML de sustitución utilizando referencias a elementos detectados previamente.

A continuación, el método urlToATag() llama al método emailToATag(), que utiliza técnicas similares para sustituir patrones de correo electrónico por cadenas de hipervínculos HTML <a>. Las expresiones regulares utilizadas para detectar direcciones URL HTTP, FTP y de correo electrónico en este archivo son bastante sencillas (para los fines del ejemplo); hay expresiones regulares mucho más complicadas para detectar direcciones URL de forma correcta.

Conversión de cadenas con valores monetarios en dólares de EE.UU. en cadenas con valores monetarios en euros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando el usuario hace clic en el botón Test de la aplicación de ejemplo, si activó la casilla de verificación `dollarToEuro`, la aplicación llama al método estático `CurrencyConverter.usdToEuro()` para convertir cadenas con valores monetarios en dólares de EE.UU. (como `"$9.95"`) en cadenas con valores monetarios en euros (como `"8.24€"`), de la manera siguiente:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

La primera línea define un patrón sencillo para detectar cadenas con valores monetarios en dólares de EE.UU. Obsérvese que antes del carácter `$` se escribe una barra diagonal inversa de escape (`\`).

El método `replace()` utiliza la expresión regular como detector de patrones y llama a la función `usdStrToEuroStr()` para determinar la cadena de sustitución (un valor en euros).

Cuando se utiliza un nombre de función como segundo parámetro del método `replace()`, se pasan a la función llamada los parámetros siguientes:

- La parte coincidente de la cadena.
- Las coincidencias detectadas por grupos de paréntesis de captura. El número de argumentos pasados de esta forma varía en función del número de capturas de grupos entre paréntesis. Se puede determinar el número de capturas de grupos entre paréntesis comprobando `arguments.length - 3` dentro del código de la función.
- La posición de índice en la que comienza la coincidencia en la cadena.
- La cadena completa.

El método `usdStrToEuroStr()` convierte patrones de cadena con valores monetarios en dólares de EE.UU. en cadenas con valores monetarios en euros, de la manera siguiente:

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Hay que tener en cuenta que `args[1]` representa la captura de grupo entre paréntesis detectada por la expresión regular `usdPrice`. Esta es la parte numérica de la cadena de dólares de EE.UU., es decir, la cantidad de dólares sin el símbolo `$`. El método aplica una conversión de tasa de cambio y devuelve la cadena resultante (con un símbolo € final, en lugar de un símbolo $ inicial).

Capítulo 6: Trabajo con XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye un grupo de clases basadas en la especificación de ECMAScript for XML (E4X) (ECMA-357 edición 2). Estas clases incluyen funciones eficaces y fáciles de usar para trabajar con datos XML. E4X permite desarrollar código con datos XML mucho más rápido que con las técnicas programación anteriores. Otra ventaja adicional es que el código que se cree será más fácil de leer.

Más temas de ayuda

XML, clase

Especificación ECMA-357

Fundamentos de la utilización de XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

XML es una forma estándar de representar información estructurada que los ordenadores pueden procesar fácilmente y que es razonablemente fácil de escribir y comprender para los humanos. XML es una abreviatura de eXtensible Markup Language (Lenguaje extensible de marcado). El estándar XML está disponible en www.w3.org/XML/.

XML ofrece una forma estándar y cómoda de clasificar datos y facilitar su lectura, acceso y manipulación. Utiliza una estructura de árbol y una estructura de etiquetas similares a las de HTML. A continuación se muestra un ejemplo sencillo de datos XML:

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Los datos XML también pueden ser más complejos, con etiquetas anidadas dentro de otras etiquetas así como atributos y otros componentes estructurales. A continuación se muestra un ejemplo más complejo de datos XML:

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

Este documento XML contiene otras estructuras XML completas (como las etiquetas `song` con sus elementos secundarios). También muestra otras estructuras XML como atributos (`tracknumber` y `length` en las etiquetas `song`) y etiquetas que contienen otras etiquetas en lugar de contener datos (como la etiqueta `tracks`).

Introducción a XML

A continuación se ofrece una descripción breve de los aspectos más comunes de los datos XML para usuarios con poca o ninguna experiencia en la utilización de XML. Los datos XML se escriben en formato de texto simple, con una sintaxis específica para organizar la información en un formato estructurado. Generalmente, un conjunto individual de datos XML se denomina *documento XML*. En formato XML, los datos se organizan en *elementos* (que pueden ser elementos de datos individuales o contenedores para otros elementos) con una estructura jerárquica. Cada documento XML tiene un elemento individual como elemento de nivel superior o principal; dentro de este elemento raíz puede haber un solo elemento de información, aunque es más probable que haya otros elementos, que a su vez contienen otros elementos, etc. Por ejemplo, este documento XML contiene información sobre un álbum de música:

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Cada elemento se distingue mediante un conjunto de *etiquetas*, constituidas por el nombre del elemento entre corchetes angulares (signos menor que y mayor que). La etiqueta inicial, que indica el principio del elemento, tiene el nombre de elemento:

```
<title>
```

La etiqueta final, que marca el final del elemento, tiene una barra diagonal antes del nombre del elemento:

```
</title>
```

Si un elemento no contiene nada, puede escribirse como un elemento vacío (y se representa con una sola etiqueta). En XML, este elemento:

```
<lastplayed/>
```

es idéntico a este elemento:

```
<lastplayed></lastplayed>
```

Además del contenido del elemento entre las etiquetas inicial y final, un elemento también puede incluir otros valores, denominados *atributos*, que se definen en la etiqueta inicial del elemento. Por ejemplo, este elemento XML define un solo atributo denominado `length`, con valor `"4:19"` :

```
<song length="4:19"></song>
```

Cada elemento XML tiene contenido, que puede ser un valor individual, uno o más elementos XML, o nada (en el caso de un elemento vacío).

Más información sobre XML

Para más información sobre la utilización de XML, hay vario libros y recursos adicionales, incluidos estos sitios web:

- Tutorial XML de W3Schools: http://w3schools.com/xml/
- Tutoriales, listas de discusión y otros elementos de XMLpitstop: http://xmlpitstop.com/

Clases de ActionScript para trabajar con XML

ActionScript 3.0 incluye varias clases que se utilizan para trabajar con información estructurada en formato XML. Las dos clases principales son:

- XML: representa un solo elemento XML, que puede ser un documento XML con varios elementos secundarios o un elemento con un solo valor en un documento.
- XMLList: representa un conjunto de elementos XML. El objeto XMLList se utiliza cuando hay varios elementos XML del mismo nivel (están en el mismo nivel y pertenecen al mismo elemento principal en la jerarquía del documento XML). Por ejemplo, una instancia de XMLList sería la manera más sencilla de trabajar con este conjunto de elementos XML (que se supone contenido en un documento XML):

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

Para usos más avanzados que requieran espacios de nombres XML, ActionScript también incluye las clases Namespace y QName. Para más información, consulte "Uso de espacios de nombres XML" en la página 112.

Además de las clases incorporadas para trabajar con XML, ActionScript 3.0 también incluye varios operadores que proporcionan funcionalidad específica para acceder a datos XML y manipularlos. Este enfoque para trabajar con XML mediante estas clases y operadores se denomina ECMAScript for XML (E4X) y está definido en la especificación de ECMA-357 edición 2.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al programar rutinas de gestión XML:

Elemento Elemento individual de un documento XML, identificado como una etiqueta inicial y una etiqueta final, y el contenido existente entre las etiquetas (incluidas estas). Los elementos XML pueden contener texto o elementos de otro tipo, o pueden estar vacíos.

Elemento vacío Elemento XML que no contiene elementos secundarios. Los elementos vacíos se suelen escribir con una sola etiqueta (como `<elemento/>`).

Documento Estructura XML individual. Un documento XML puede contener un número arbitrario de elementos (o puede constar únicamente de un elemento vacío); no obstante, debe tener un solo elemento de nivel superior que contenga a todos los demás elementos del documento.

Nodo Nombre alternativo para un elemento XML.

Atributo Valor con nombre asociado con un elemento que se escribe en la etiqueta inicial del elemento con el formato `nombreAtributo="valor"`, en lugar de escribirse como un elemento secundario independiente anidado dentro del elemento.

El enfoque E4X del procesamiento de XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La especificación de ECMAScript for XML define un conjunto de clases y funcionalidad para trabajar con datos XML. Este conjunto de clases y funcionalidades se denomina *E4X*. ActionScript 3.0 incluye las siguientes clases E4X: XML, XMLList, QName y Namespace.

Los métodos, propiedades y operadores de las clases de E4X se han diseñado con los siguientes objetivos:

- Simplicidad: siempre que sea posible, E4X facilita la escritura y comprensión del código para trabajar con datos XML.
- Coherencia: los métodos y la lógica que subyacen a E4X son coherentes internamente y con otros componentes de ActionScript.
- Familiaridad: los datos XML se manipulan con operadores conocidos, como el operador punto (.).

Nota: *ActionScript 2.0 tenía una clase XML. En ActionScript 3.0 se ha cambiado su nombre a XMLDocument para que no entre en conflicto con la clase XML de ActionScript 3.0 que forma parte de E4X. Las clases antiguas (XMLDocument, XMLNode, XMLParser y XMLTag) se incluyen en el paquete flash.xml principalmente por compatibilidad con código antiguo. Las nuevas clases de E4X son clases principales; no es necesario importar un paquete para utilizarlas. Para obtener información sobre las clases XML heredadas de ActionScript 2.0, consulte el paquete flash.xml en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.*

A continuación se muestra un ejemplo de manipulación de datos con E4X:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

A menudo, la aplicación cargará datos XML desde un origen externo, como un servicio web o un canal RSS. No obstante, para mayor claridad, los ejemplos de código proporcionados asignan datos XML como literales.

Tal y como se muestra en el código siguiente, E4X incluye algunos operadores intuitivos, como el operador punto (.) y el operador de identificador de atributo (@), para acceder a propiedades y atributos en datos XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

El método `appendChild()` se utiliza para asignar un nuevo nodo secundario a los datos XML, como se indica en el siguiente fragmento de código:

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Los operadores `@` y `.` se utilizan no solo para leer datos, sino también para asignar datos, como se indica a continuación:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Se puede utilizar un bucle `for` para recorrer nodos de los datos XML, de la manera siguiente:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

Objetos XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto XML puede representar un elemento, atributo, comentario, instrucción de procesamiento o elemento de texto XML.

Los objetos XML pueden clasificarse como de *contenido simple* o de *contenido complejo*. Un objeto XML que tiene nodos secundarios se clasifica como objeto de contenido complejo. Se dice que un objeto XML tiene contenido simple si es cualquiera de los siguientes elementos: un atributo, un comentario, una instrucción de procesamiento o un nodo de texto.

Por ejemplo, el siguiente objeto XML contiene contenido complejo, incluidos un comentario y una instrucción de procesamiento:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como se muestra en el siguiente ejemplo, ahora se pueden utilizar los métodos `comments()` y `processingInstructions()` para crear nuevos objetos XML, un comentario y una instrucción de procesamiento:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

Propiedades XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase XML tiene cinco propiedades estáticas:

- Las propiedades `ignoreComments` e `ignoreProcessingInstructions` determinan si deben omitirse los comentarios o las instrucciones de procesamiento cuando se analice el objeto XML.
- La propiedad `ignoreWhitespace` determina si deben omitirse los caracteres de espacio en blanco en las etiquetas de elemento y las expresiones incorporadas que solo estén separadas por caracteres de espacio en blanco.
- Las propiedades `prettyIndentyprettyPrinting` se utilizan para aplicar formato al texto devuelto por los métodos `toString()` y `toXMLString()` de la clase XML.

Para obtener información sobre estas propiedades, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Métodos XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los siguientes métodos permiten trabajar con la estructura jerárquica de los objetos XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`

- `insertChildBefore()`
- `parent()`
- `prependChild()`

Los siguientes métodos permiten trabajar con atributos de objetos XML:

- `attribute()`
- `attributes()`

Los siguientes métodos permiten trabajar con propiedades de objetos XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Los siguientes métodos sirven para trabajar con nombres completos y espacios de nombres:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Los siguientes métodos sirven para trabajar con (y determinar) tipos específicos de contenido XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Los siguientes métodos sirven para la conversión a cadenas y para aplicar formato a objetos XML:

- `defaultSettings()`
- `setSettings()`
- `configuración()`
- `normalize()`
- `toString()`
- `toXMLString()`

Hay algunos métodos adicionales:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Para obtener información sobre estos métodos, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Objetos XMLList

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una instancia de XMLList representa una colección arbitraria de objetos XML. Puede contener documentos XML completos, fragmentos XML o los resultados de una consulta XML.

Los siguientes métodos permiten trabajar con la estructura jerárquica de los objetos XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Los siguientes métodos permiten trabajar con atributos de objetos XMLList:

- `attribute()`
- `attributes()`

Los siguientes métodos permiten trabajar con las propiedades de XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Los siguientes métodos sirven para trabajar con (y determinar) tipos específicos de contenido XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Los siguientes métodos sirven para la conversión a cadenas y para aplicar formato al objeto XMLList:

- `normalize()`
- `toString()`
- `toXMLString()`

Hay algunos métodos adicionales:

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Para obtener información sobre estos métodos, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Para un objeto XMLList que contiene exactamente un elemento XML se pueden utilizar todas las propiedades y métodos de la clase XML, ya que un objeto XMLList con un elemento XML se trata igual que un objeto XML. Por ejemplo, en el código siguiente, como `doc.div` es un objeto XMLList que contiene un elemento, se puede utilizar el método `appendChild()` de la clase XML:

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Para obtener una lista de propiedades y métodos XML, consulte "Objetos XML" en la página 101.

Inicialización de variables XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede asignar un literal XML a un objeto XML de la manera siguiente:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como se indica en el siguiente fragmento de código, también se puede utilizar el constructor `new` para crear una instancia de un objeto XML de una cadena que contiene datos XML:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Si los datos XML de la cadena no están bien formados (por ejemplo, si falta una etiqueta final), aparecerá un error en tiempo de ejecución.

También se puede pasar datos por referencia (desde otras variables) a un objeto XML, como se indica en el siguiente ejemplo:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

Para cargar datos XML desde una dirección URL hay que utilizar la clase URLLoader, como se indica en el siguiente ejemplo:

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

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Para leer datos XML desde una conexión de socket hay que utilizar la clase XMLSocket. Para obtener más información, consulte la clase XMLSocket en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Construcción y transformación de objetos XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `prependChild()` y `appendChild()` permiten añadir una propiedad al principio o al final (respectivamente) de una lista de propiedades de un objeto XML, como se indica en el siguiente ejemplo:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Los métodos `insertChildBefore()` e `insertChildAfter()` permiten añadir una propiedad antes o después (respectivamente) de una propiedad especificada, como se indica a continuación:

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Como se indica en el siguiente ejemplo, también se pueden utilizar operadores de llave ({ y }) para pasar datos por referencia (desde otras variables) al construir objetos XML:

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

Se pueden asignar propiedades y atributos a un objeto XML utilizando el operador `=`, como se indica a continuación:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Esto establece el valor del objeto XML `x` en:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Se pueden utilizar los operadores + y += para concatenar objetos XMLList.

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Esto establece el valor del objeto XMLList `xList` en:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Navegación de estructuras XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las eficaces características de XML es su capacidad de proporcionar datos complejos y anidados a través de una cadena lineal de caracteres de texto. Al cargar datos en un objeto XML, ActionScript analiza los datos y carga su estructura jerárquica en memoria (o envía un error en tiempo de ejecución si los datos XML no están bien formados).

Los operadores y métodos de los datos XML y objetos XMLList facilitan la navegación de la estructura de datos XML.

El operador punto (.) y el operador descriptor de acceso descendente (..) permiten acceder a propiedades secundarias de un objeto XML. Considere el siguiente objeto XML:

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

El objeto `myXML.book` es un objeto XMLList que contiene propiedades secundarias del objeto `myXML` denominado `book`. Son dos objetos XML, que coinciden con las dos propiedades `book` del objeto `myXML`.

El objeto `myXML..lastName` es un objeto XMLList que contiene todas las propiedades de descendentes denominadas `lastName`. Son dos objetos XML, que coinciden con las dos propiedades `lastName` del objeto `myXML`.

El objeto `myXML.book.editor.lastName` es un objeto XMLList que contiene los elementos secundarios que tengan el nombre `lastName` de los elementos secundarios denominados `editor` de los elementos secundarios llamados `book` del objeto `myXML`: en este caso, es un objeto XMLList que contiene solo un objeto XML (la propiedad `lastName` con el valor "`Case`").

Acceso a nodos principales y secundarios

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `parent()` devuelve el elemento principal de un objeto XML.

Se pueden utilizar los valores de índice ordinales de una lista secundaria para acceder a objetos secundarios específicos. Por ejemplo, considérese un objeto XML `myXML` que tiene dos propiedades secundarias denominadas `book`. Cada propiedad secundaria denominada `book` tiene un número de índice asociado:

```
myXML.book[0]
myXML.book[1]
```

Para acceder a un elemento terciario específico se pueden especificar números de índice para los nombres del elemento secundario y el elemento terciario:

```
myXML.book[0].title[0]
```

Sin embargo, si `x.book[0]` solo tuviera un elemento secundario denominado `title`, se puede omitir la referencia al índice, como se muestra a continuación:

```
myXML.book[0].title
```

De forma similar, si solo hay un elemento secundario book del objeto `x` y dicho objeto secundario tiene un solo objeto de título, se pueden omitir ambas referencias de índice, como se muestra a continuación:

```
myXML.book.title
```

Se puede utilizar el método `child()` para desplazarse por los elementos secundarios con nombres basados en una variable o expresión, como se indica en el siguiente ejemplo:

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Acceso a atributos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El símbolo `@` (el operador identificador de atributo) se utiliza para acceder a atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Se puede utilizar el símbolo de comodín `*` con el símbolo `@` para acceder a todos los atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Se puede utilizar el método `attribute()` o `attributes()` para acceder a un atributo específico o a todos los atributos de un objeto XML o XMLList, como se muestra en el código siguiente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

También se puede utilizar la sintaxis siguiente para acceder a atributos, como se muestra en el siguiente ejemplo:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Todos ellos equivalen a `employee.@id`. Sin embargo, la sintaxis `employee.@id` es la preferida.

Filtrado por atributo o valor de elemento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar los operadores de paréntesis, `(` y `)`, para filtrar elementos con un nombre de elemento o un valor de atributo específicos. Considere el siguiente objeto XML:

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

Las siguientes expresiones son todas válidas:

- `x.employee.(lastName == "McGee")`; es el segundo nodo `employee`.
- `x.employee.(lastName == "McGee").firstName`; es la propiedad `firstName` del segundo nodo `employee`.
- `x.employee.(lastName == "McGee").@id`; es el valor del atributo `id` del segundo nodo `employee`.
- `x.employee.(@id == 347)`; es el primer nodo `employee`.
- `x.employee.(@id== 347).lastName`; es la propiedad `lastName` del primer nodo `employee`.
- `x.employee.(@id > 300)`; es un objeto XMLList con ambas propiedades `employee`.
- `x.employee.(position.toString().search("analyst") > -1)`; es un objeto XMLList con ambas propiedades `position`.

Si se intentan filtrar atributos o elementos que no existen, se emitirá una excepción. Por ejemplo, la línea final del código siguiente genera un error porque no hay ningún atributo `id` en el segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

De manera similar, la línea final del código siguiente genera un error porque no hay ningún atributo `b` en el segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Para evitar estos errores, se pueden identificar las propiedades que tienen los atributos o elementos coincidentes mediante los métodos `attribute()` y `elements()`, como se muestra en el código siguiente:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

También se puede utilizar el método `hasOwnProperty()`, como se muestra en el código siguiente:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

Uso de las sentencias for..in y for each..in

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye las sentencias `for..in` y `for each..in` para recorrer los objetos XMLList. Por ejemplo, considérese el objeto XML `myXML` y el objeto XMLList `myXML.item`. El objeto XMLList, `myXML.item`, consta de los dos nodos `item` del objeto XML.

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

El bucle `for..in` permite recorrer un conjunto de nombres de propiedad de un objeto XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

El bucle `for each..in` permite recorrer las propiedades del objeto XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

Uso de espacios de nombres XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los espacios de nombres de un objeto (o documento) XML identifican el tipo de datos que el objeto contiene. Por ejemplo, al enviar y entregar datos XML a un servicio web que utiliza el protocolo de mensajería SOAP, se declara el espacio de nombres en la etiqueta inicial de los datos XML:

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

El espacio de nombres tiene un prefijo, `soap`, y un URI que define el espacio de nombres, `http://schemas.xmlsoap.org/soap/envelope/`.

ActionScript 3.0 incluye la clase Namespace para trabajar con espacios de nombres XML. Para el objeto XML del ejemplo anterior se puede utilizar la clase Namespace de la manera siguiente:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

La clase XML incluye los siguientes métodos para trabajar con espacios de nombres: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` y `setNamespace()`.

La directiva `default xml namespace` permite asignar un espacio de nombres predeterminado para objetos XML. Por ejemplo, en el fragmento de código siguiente, `x1` y `x2` tienen el mismo espacio de nombres predeterminado:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

Conversión de tipo XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden convertir objetos XML y XMLList a valores de cadena. De forma similar, se pueden convertir cadenas en objetos XML y XMLList. También se debe tener en cuenta que todos los valores de atributos, nombres y valores de texto XML son cadenas. En las secciones siguientes se tratan todas estas formas de conversión de tipo XML.

Conversión de objetos XML y XMLList en cadenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las clases XML y XMLList incluyen un método `toString()` y un método `toXMLString()`. El método `toXMLString()` devuelve una cadena que incluye todas las etiquetas, los atributos, las declaraciones de espacios de nombres y el contenido del objeto XML. Para objetos XML con contenido complejo (elementos secundarios), el método `toString()` hace exactamente lo mismo que el método `toXMLString()`. Para objetos XML con contenido simple (los que contienen un solo elemento de texto), el método `toString()` devuelve únicamente el contenido de texto del elemento, como se indica en el siguiente ejemplo:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

Si se utiliza el método `trace()` sin especificar `toString()` ni `toXMLString()`, los datos se convierten con el método `toString()` de manera predeterminada, como se muestra en el código siguiente:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

Al utilizar el método `trace()` para depurar código, generalmente se deseará utilizar el método `toXMLString()` para que el método `trace()` devuelva datos más completos.

Conversión de cadenas a objetos XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar el constructor `new XML()` para crear un objeto XML de una cadena, de la manera siguiente:

```
var x:XML = new XML("<a>test</a>");
```

Si se intenta convertir una cadena en XML a partir de una cadena que representa datos XML no válidos o que no están bien formados, se emitirá un error en tiempo de ejecución, como se muestra a continuación:

```
var x:XML = new XML("<a>test"); // throws an error
```

Conversión de valores de atributos, nombres y valores de texto de tipo cadena

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todos los valores de atributos, nombres y valores de texto XML son del tipo de datos String y es posible que sea necesario convertirlos a otros tipos de datos. Por ejemplo, el código siguiente utiliza la función `Number()` para convertir valores de texto en números:

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.95;
```

Si este código no utilizara la función `Number()`, interpretaría el operador + como el operador de concatenación de cadenas y el método `trace()` en la última línea emitiría lo siguiente:

```
01.003.95
```

Lectura de documentos XML externos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar la clase URLLoader para cargar datos XML desde una dirección URL. Para utilizar el código siguiente en las aplicaciones hay que sustituir el valor de `XML_URL` del ejemplo por una dirección URL válida:

```
import flash.events.Event;
import flash.net.URLLoader;

var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener(Event.COMPLETE, xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

También se puede utilizar la clase XMLSocket para configurar una conexión de socket XML asíncrona con un servidor. Para obtener más información, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Ejemplo de XML en ActionScript: Carga de datos RSS desde Internet

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación de ejemplo RSSViewer muestra diversas características del trabajo con XML en ActionScript, incluidas las siguientes:

- Uso de métodos XML para recorrer datos XML en forma de canal RSS.
- Uso de métodos XML para crear datos XML en formato HTML para utilizarlos en un campo de texto.

Se ha extendido el uso del formato RSS para sindicar noticias en formato XML. El aspecto de un archivo de datos RSS simple será similar al siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

La aplicación SimpleRSS lee datos RSS de Internet, analiza dichos datos en busca de titulares (títulos), vínculos y descripciones, y devuelve esos datos. La clase SimpleRSSUI proporciona la interfaz de usuario y llama a la clase SimpleRSS, que lleva a cabo todo el procesamiento de los datos XML.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación RSSViewer se encuentran en la carpeta Samples/RSSViewer. La aplicación consta de los siguientes archivos:

Archivo	**Descripción**
RSSViewer.mxml o RSSViewer.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/rssViewer/RSSParser.as	Una clase que contiene métodos que utilizan E4X para atravesar datos RSS (XML) y generan una representación HTML correspondiente.
RSSData/ak.rss	Un archivo RSS de ejemplo. La aplicación está configurada para leer datos RSS de Internet, en un canal RSS de Flex alojado por Adobe. No obstante, se puede modificar fácilmente para que lea datos RSS de este documento, que utiliza un esquema ligeramente distinto del que utiliza el canal RSS de Flex.

Lectura y análisis de datos XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase RSSParser incluye un método `xmlLoaded()` que convierte los datos RSS de entrada, almacenados en la variable `rssXML`, en una cadena que contiene la salida en formato HTML, `rssOutput`.

Casi al principio del método, el código establece el espacio de nombres XML predeterminado si el origen de datos RSS incluye un espacio de nombres predeterminado:

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

Las líneas siguientes recorren el contenido del origen de datos XML, examinando cada propiedad descendente denominada `item`:

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

Las tres primeras líneas son simplemente un conjunto de variables de cadena para representar las propiedades de título, descripción y vínculo de la propiedad `item` de los datos XML. A continuación, la siguiente línea llama al método `buildItemHTML()` para obtener datos HTML en forma de objeto XMLList, utilizando las tres nuevas variables de cadena como parámetros.

Construcción de datos XMLList

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los datos HTML (un objeto XMLList) tienen la siguiente forma:

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

Las primeras líneas del método borran el espacio de nombres XML predeterminado:

```
default xml namespace = new Namespace();
```

La directiva `default xml namespace` tiene ámbito de nivel de bloque de función. Esto significa que el ámbito de esta declaración es el método `buildItemHTML()`.

Las líneas siguientes crean el objeto XMLList basándose en los argumentos de cadena pasados a la función:

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

Este objeto XMLList representa una cadena datos adecuada para un campo de texto HTML de ActionScript.

El método `xmlLoaded()` utiliza el valor devuelto por el método `buildItemHTML()` y lo convierte en una cadena:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Extracción del título del canal RSS y envío de un evento personalizado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `xmlLoaded()` establece una variable de cadena `rssTitle` a partir de la información de los datos XML RSS de origen:

```
rssTitle = rssXML.channel.title.toString();
```

Por último, el método `xmlLoaded()` genera un evento, que notifica a la aplicación que los datos ha sido analizados y están disponibles:

```
            dataWritten = new Event("dataWritten", true);
```

Capítulo 7: Uso de la funcionalidad JSON nativa

ActionScript 3.0 proporciona una API nativa para codificar y descodificar objetos de ActionScript con formato JavaScript Object Notation (JSON). La clase JSON y las funciones de miembros admitidas siguen la especificación de la quinta edición del ECMA-262 con algunas variaciones.

Todd Anderson es un usuario de la comunidad y nos ofrece una comparación entre la API JSON nativa y la clase JSON as3corelib de otro fabricante. Consulte Working with Native JSON in Flash Player 11 (en inglés).

Más temas de ayuda

JSON

Descripción de la API JSON

La API JSON de ActionScript consta de la clase JSON y de las funciones de miembros `toJSON()` en varias clases nativas. En el caso de aplicación que requieren una codificación JSON personalizada para alguna clase, la estructura de ActionScript proporciona alternativas para omitir la codificación predeterminada.

La clase JSON gestiona internamente la importación y la exportación de cualquier clase de ActionScript que no proporcione un miembro de `toJSON()`. En esos casos, JSON recorre las propiedades públicas de cada objeto que encuentra. Si un objeto contiene otros objetos, JSON lo recorre repetidamente los objetos anidados y hace su trabajo. Si algún objeto proporciona un método `toJSON()`, JSON utiliza dicho método personalizado en lugar de su algoritmo interno.

La interfaz JSON está formada por un método de codificación, `stringify()`, y uno de descodificación, `parse()`. Cada uno de ellos proporciona un parámetro que permite insertar su propia lógica en el flujo de trabajo de codificación y descodificación JSON. Para `stringify()`, este parámetro se llama `replacer`; para `parse()`, se llama `reviver`. Estos parámetros toman una definición de función con dos argumentos y para ello utilizan la siguiente firma:

```
function(k, v):*
```

Métodos toJSON()

La firma de los métodos `toJSON()` es

```
public function toJSON(k:String):*
```

`JSON.stringify()` llama a `toJSON()`, si existe, en cada propiedad pública que encuentra durante su recorrido por el objeto. Una propiedad está formada por una pareja clave-valor. Si `stringify()` llama a `toJSON()`, transfiere la clave `k` de la propiedad que se está analizando. Una implementación típica de `toJSON()` calcula cada nombre y devuelve la codificación deseada de su valor.

El método `toJSON()` puede devolver un valor de cualquier tipo (indicado con *), no solo de tipo String. Este tipo devuelto variable permite que `toJSON()` devuelva un objeto, si procede. Por ejemplo, si una propiedad de la clase personalizada contiene un objeto de la biblioteca de un tercero, puede devolver dicho objeto cuando `toJSON()` encuentre su propiedad. Entonces JSON lo recorrerá repetidas veces el objeto del tercero. El flujo del proceso de codificación se comporta del modo siguiente:

- Si `toJSON()` devuelve un objeto que no da como resultado una cadena, `stringify()` recorre repetidas veces dicho objeto.
- Si `toJSON()` devuelve una cadena, `stringify()` resume dicho valor en otra cadena, devuelve la cadena resumida y, después, pasa al siguiente valor.

En muchos casos, es preferible devolver un objeto a devolver una cadena JSON creada por la aplicación. Devolver un objeto aprovecha el algoritmo de codificación JSON integrado y también permite que JSON pueda ejecutarse repetidas veces en objetos anidados.

El método `toJSON()` no se define en la clase Object ni en la mayor parte de clases nativas. Su ausencia hace que JSON lleve a cabo su recorrido habitual por las propiedades públicas del objeto. Si lo desea, también puede utilizar `toJSON()` para mostrar las propiedades privadas del objeto.

Algunas clases representan retos que las bibliotecas de ActionScript no son capaces de resolver de forma eficaz en todos los casos. Para estas clases, ActionScript proporciona una implementación trivial que los clientes pueden volver a implementar para ajustarla a sus necesidades. Las clases que proporcionan miembros de `toJSON()` triviales son:

- ByteArray
- Date
- Dictionary
- XML

Puede crear una subclase de la clase ByteArray para anular este método `toJSON()`, o puede redefinir su prototipo. Las clases Date y XML, que se declaran finales, requieren el uso del prototipo de la clase para poder redefinir `toJSON()`. La clase Dictionary se declara dinámica, y se obtiene así libertad adicional a la hora de anular `toJSON()`.

Definición de comportamiento JSON personalizado

Para implementar su propia codificación y descodificación JSON para clases nativas, puede elegir una de las siguientes opciones:

- Definir o anular `toJSON()` en la subclase personalizada de una clase nativa no final
- Definir o redefinir `toJSON()` en el prototipo de la clase
- Definir una propiedad `toJSON` en una clase dinámica
- Utilizar los parámetros `JSON.stringify() replacer` y `JSON.parser() reviver`

Más temas de ayuda

ECMA-262, quinta edición

Definición de toJSON() en el prototipo de una clase integrada

La implementación nativa de JSON en ActionScript imita el mecanismo JSON de ECMAScript definido en ECMA-262, quinta edición. Dado que ECMAScript no admite clases, ActionScript define el comportamiento de JSON en términos de distribución basada en prototipos. Los prototipos son los precursores de las clases de ActionScript 3.0 y permiten herencia simulada, así como incorporaciones y nuevas definiciones de miembros.

ActionScript permite definir o redefinir `toJSON()` en el prototipo de cualquier clase. Este privilegio se aplica incluso a clases marcadas como finales. Cuando se define `toJSON()` en un prototipo de clase, la definición se aplica a todas las instancias de dicha clase dentro del ámbito de la aplicación. Por ejemplo, así puede definir un método `toJSON()` en el prototipo MovieClip:

```
MovieClip.prototype.toJSON = function(k):* {
    trace("prototype.toJSON() called.");
    return "toJSON";
}
```

Cuando la aplicación llame en ese momento a `stringify()` en cualquier instancia de MovieClip, `stringify()` devuelve la salida del método `toJSON()`:

```
var mc:MovieClip = new MovieClip();
var js:String = JSON.stringify(mc); //"prototype toJSON() called."
trace("js: " + js); //"js: toJSON"
```

También puede anular `toJSON()` en clases nativas que definan el método. Por ejemplo, el siguiente código anula `Date.toJSON()`:

```
Date.prototype.toJSON = function (k):* {
    return "any date format you like via toJSON: "+
        "this.time:"+this.time + " this.hours:"+this.hours;
}
var dt:Date = new Date();
trace(JSON.stringify(dt));
// "any date format you like via toJSON: this.time:1317244361947 this.hours:14"
```

Definición o anulación de toJSON() en el nivel de clase

No siempre se necesitan aplicaciones para usar prototipos para redefinir `toJSON()`. También se puede definir `toJSON()` como miembro de una subclase si la clase principal no está marcada como final. Por ejemplo, puede ampliar la clase ByteArray y definir una función `toJSON()` pública:

```
package {

    import flash.utils.ByteArray;
    public class MyByteArray extends ByteArray
    {
        public function MyByteArray() {
        }

        public function toJSON(s:String):*
        {
            return "MyByteArray";
        }

    }
}


var ba:ByteArray = new ByteArray();
trace(JSON.stringify(ba)); //"ByteArray"
var mba:MyByteArray = new MyByteArray(); //"MyByteArray"
trace(JSON.stringify(mba)); //"MyByteArray"
```

Si una clase es dinámica, puede añadir una propiedad `toJSON` a un objeto de dicha clase y asignarle una función del modo siguiente:

```
var d:Dictionary = new Dictionary();
trace(JSON.stringify((d))); // "Dictionary"
d.toJSON = function(){return {c : "toJSON override."};} // overrides existing function
trace(JSON.stringify((d))); // {"c":"toJSON override."}
```

Puede anular, definir o redefinir `toJSON()` en cualquier clase de ActionScript. No obstante, la mayoría de las clases de ActionScript integradas no definen `toJSON()`. Sin embargo, la clase Object no define el método `toJSON` en su prototipo predeterminado ni lo declara como un miembro de clase. Solo unas pocas clases nativas definen el método como una función prototipo. Así, en la mayoría de los casos no es posible anular `toJSON()` en el sentido tradicional.

Las clases nativas que no definen `toJSON()` se serializan en JSON con una implementación interna de JSON. Evite sustituir esta funcionalidad integrada si es posible. Si define un miembro de `toJSON()`, la clase JSON usará su lógica en vez de su propia funcionalidad.

Uso del parámetro replacer de JSON.stringify()

Anular `toJSON()` en el prototipo resulta útil para cambiar el comportamiento de exportación JSON de una clase en toda la aplicación. En algunos casos, no obstante, la lógica de exportación es posible que se aplique solamente a casos especiales en condiciones transitorias. Para poder aceptar cambios con alcance tan limitado, puede utilizar el parámetro `replacer` del método `JSON.stringify()`.

El método `stringify()` aplica la función transferida a través del parámetro `replacer` al objeto que se está codificando. La firma de esta función es similar a la de `toJSON()`:

```
function (k,v):*
```

Al contrario que `toJSON()`, la función `replacer` requiere el valor `v` así como la clave `k`. Esta diferencia es necesaria, ya que `stringify()` está definido en el objeto estático JSON y no en el objeto que se está codificando. Cuando `JSON.stringify()` llama a `replacer(k,v)`, su recorre el objeto de entrada original. El parámetro `this` implícito transferido a la función `replacer` hace referencia al objeto que contiene la clave y el valor. Como `JSON.stringify()` no modifica el objeto de entrada original, dicho objeto permanece intacto en el contenedor que se está recorriendo. De este modo, es posible utilizar el código `this[k]` para consultar la clave en el objeto original. El parámetro `v` aloja el valor que convierte `toJSON()`.

Al igual que `toJSON()`, la función `replacer` puede devolver cualquier tipo de valor. Si `replacer` devuelve una cadena, el motor JSON bloquea el contenido entre comillas y envuelve el contenido bloqueado entre comillas también. Esta envoltura garantiza que `stringify()` recibe un objeto de cadena JSON válido que seguirá siendo una cadena en la siguiente llamada a `JSON.parse()`.

El siguiente código utiliza el parámetro `replacer` y el parámetro implícito `this` para devolver los valores de `time` y `hours` de un objeto Date:

```
JSON.stringify(d, function (k,v):* {
    return "any date format you like via replacer: "+
        "holder[k].time:"+this[k].time + " holder[k].hours:"+this[k].hours;
});
```

Uso del parámetro reviver de JSON.parse()

El parámetro `reviver` del método `JSON.parse()` hace lo contrario que la función `replacer`: convierte a una cadena JSON en un objeto de ActionScript utilizable. El argumento `reviver` es una función que toma dos parámetros y devuelve cualquier tipo:

```
function (k,v):*
```

En esta función, `k` es una clave y `v` es el valor de `k`. Del mismo modo que `stringify()`, `parse()` recorre los pares de clave/valor de JSON y aplica la función `reviver` a cada par (si existe). Un problema potencial es el hecho de que la clase de JSON no produce un nombre de clase de ActionScript de un objeto como salida. Por lo tanto, puede resultar complicado saber qué tipo de objeto se revive. Este problema es especialmente reseñable en objetos anidados. Al diseñar funciones `toJSON()`, `replacer` y `reviver` puede buscar formas de identificar los objetos de ActionScript exportados sin tocar los objetos originales.

Ejemplos de análisis

El siguiente ejemplo muestra una estrategia para revivir objetos analizados desde cadenas JSON. Este ejemplo define dos clases: JSONGenericDictExample y JSONDictionaryExtnExample. La clase JSONGenericDictExample es una clase personalizada de diccionario. Cada registro contiene el nombre de una persona y su cumpleaños, así como un identificador exclusivo. Cada vez que se llama al constructor JSONGenericDictExample, se añade un objeto recién creado a un conjunto estático interno con un entero que incrementa como su identificador. La clase JSONGenericDictExample también define un método `revive()` que extrae la porción entera del miembro de `id` más largo. El método `revive()` utiliza este entero para buscar y devolver el objeto adecuado.

La clase JSONDictionaryExtnExample amplía la clase Dictionary de ActionScript. Sus registros no tienen ninguna estructura definida y pueden contener cualquier dato. Los datos se asignan después de construir un objeto JSONDictionaryExtnExample, no como propiedades definidas por la clase. Los registros de JSONDictionaryExtnExample utilizan objetos JSONGenericDictExample como claves. Cuando se revive un objeto JSONDictionaryExtnExample, la función `JSONGenericDictExample.revive()` utiliza el identificador asociado a JSONDictionaryExtnExample para recuperar el objeto de clave correcto.

Lo que es más importante, el método `JSONDictionaryExtnExample.toJSON()` devuelve una cadena de marcador además del objeto JSONDictionaryExtnExample. Esta cadena identifica la salida JSON como perteneciente a la clase JSONDictionaryExtnExample. Este marcador no deja ninguna duda sobre el tipo de objeto que se está procesando durante `JSON.parse()`.

```
package {
    // Generic dictionary example:
    public class JSONGenericDictExample {
        static var revivableObjects = [];
        static var nextId = 10000;
        public var id;
        public var dname:String;
        public var birthday;

        public function JSONGenericDictExample(name, birthday) {
            revivableObjects[nextId] = this;
            this.id       = "id_class_JSONGenericDictExample_" + nextId;
            this.dname     = name;
            this.birthday = birthday;
            nextId++;
        }
        public function toString():String { return this.dname; }
        public static function revive(id:String):JSONGenericDictExample {
            var r:RegExp = /^id_class_JSONGenericDictExample_([0-9]*)$/;
            var res = r.exec(id);
            return JSONGenericDictExample.revivableObjects[res[1]];
        }
    }
}

package {
    import flash.utils.Dictionary;
    import flash.utils.ByteArray;

    // For this extension of dictionary, we serialize the contents of the
    // dictionary by using toJSON
    public final class JSONDictionaryExtnExample extends Dictionary {
        public function toJSON(k):* {
            var contents = {};
            for (var a in this) {
                contents[a.id] = this[a];
            }

            // We also wrap the contents in an object so that we can
            // identify it by looking for the marking property "class E"
            // while in the midst of JSON.parse.
```

```
            return {"class JSONDictionaryExtnExample": contents};
        }

        // This is just here for debugging and for illustration
        public function toString():String {
            var retval = "[JSONDictionaryExtnExample <";
            var printed_any = false;
            for (var k in this) {
                retval += k.toString() + "=" +
                "[e="+this[k].earnings +
                ",v="+this[k].violations + "], "
                printed_any = true;
            }
            if (printed_any)
                retval = retval.substring(0, retval.length-2);
            retval += ">]"
            return retval;
        }
    }
}
```

En tiempo de ejecución, cuando el siguiente script llama a `JSON.parse()` en un objeto JSONDictionaryExtnExample, la función `reviver` llama a `JSONGenericDictExample.revive()` en cada objeto de JSONDictionaryExtnExample. Esta llamada extrae el identificador que representa a la clave del objeto. La función `JSONGenericDictExample.revive()` utiliza este identificador para recuperar y devolver el objeto JSONDictionaryExtnExample almacenado desde un conjunto estático privado.

```
import flash.display.MovieClip;
import flash.text.TextField;

var a_bob1:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("01/02/1934")));
var a_bob2:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("05/06/1978")));
var a_jen:JSONGenericDictExample = new JSONGenericDictExample("Jen", new
Date(Date.parse("09/09/1999")));

var e = new JSONDictionaryExtnExample();
e[a_bob1] = {earnings: 40, violations: 2};
e[a_bob2] = {earnings: 10, violations: 1};
e[a_jen]  = {earnings: 25, violations: 3};

trace("JSON.stringify(e): " + JSON.stringify(e)); // {"class JSONDictionaryExtnExample":
                        //{"id_class_JSONGenericDictExample_10001":
                        //{"earnings":10,"violations":1},
                        //"id_class_JSONGenericDictExample_10002":
                        //{"earnings":25,"violations":3},
                        //"id_class_JSONGenericDictExample_10000":
                        // {"earnings":40,"violations":2}}}

var e_result = JSON.stringify(e);

var e1 = new JSONDictionaryExtnExample();
var e2 = new JSONDictionaryExtnExample();

// It's somewhat easy to convert the string from JSON.stringify(e) back
// into a dictionary (turn it into an object via JSON.parse, then loop
```

```
// over that object's properties to construct a fresh dictionary).
//
// The harder exercise is to handle situations where the dictionaries
// themselves are nested in the object passed to JSON.stringify and
// thus does not occur at the topmost level of the resulting string.
//
// (For example: consider roundtripping something like
//   var tricky_array = [e1, [[4, e2, 6]], {table:e3}]
// where e1, e2, e3 are all dictionaries.  Furthermore, consider
// dictionaries that contain references to dictionaries.)
//
// This parsing (or at least some instances of it) can be done via
// JSON.parse, but it's not necessarily trivial.  Careful consideration
// of how toJSON, replacer, and reviver can work together is
// necessary.

var e_roundtrip =
    JSON.parse(e_result,
            // This is a reviver that is focused on rebuilding JSONDictionaryExtnExample objects.
                function (k, v) {
                     if ("class JSONDictionaryExtnExample" in v) { // special marker tag;
                         //see JSONDictionaryExtnExample.toJSON().
                        var e = new JSONDictionaryExtnExample();
                        var contents = v["class JSONDictionaryExtnExample"];
                        for (var i in contents) {
                           // Reviving JSONGenericDictExample objects from string
                           // identifiers is also special;
                           // see JSONGenericDictExample constructor and
                           // JSONGenericDictExample's revive() method.
                            e[JSONGenericDictExample.revive(i)] = contents[i];
                        }
                        return e;
                    } else {
                        return v;
                    }
                });

trace("// == Here is an extended Dictionary that has been round-tripped  ==");
trace("// == Note that we have revived Jen/Jan during the roundtrip.    ==");
trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                          //Jen=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jen=[e=25,v=3]>]
trace("Is e_roundtrip a JSONDictionaryExtnExample? " + (e_roundtrip is
JSONDictionaryExtnExample)); //true
trace("Name change: Jen is now Jan");
a_jen.dname = "Jan"

trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                          //Jan=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jan=[e=25,v=3]>]
```

Capítulo 8: Gestión de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un sistema de gestión de eventos permite a los programadores responder a entradas del usuario y eventos del sistema de una forma conveniente. El modelo de eventos de ActionScript 3.0 no solo resulta conveniente, sino que además cumple las normas pertinentes y está bien integrado con la lista de visualización. El nuevo modelo de eventos está basado en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3, una arquitectura de gestión de eventos estándar, y constituye una herramienta de gestión de eventos intuitiva y de grandes prestaciones para los programadores de ActionScript.

El sistema de gestión de eventos de ActionScript 3. interactúa de forma muy estrecha con la lista de visualización. Para conocer los aspectos básicos de la lista de visualización, consulte "Programación de la visualización" en la página 154.

Más temas de ayuda

Paquete flash.events

Especificación de eventos del modelo de objetos de documento (DOM) de nivel 3

Fundamentos de la gestión de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los eventos se pueden considerar como sucesos de cualquier tipo en el archivo SWF que resultan de interés para el programador. Por ejemplo, la mayor parte de los archivos SWF permiten algún tipo de interacción con el usuario, ya sea algo tan sencillo como responder a un clic del ratón o algo mucho más complejo, como aceptar y procesar datos escritos en un formulario. Toda interacción de este tipo entre el usuario y el archivo SWF se considera un evento. Los eventos también pueden producirse sin interacción directa con el usuario, como cuando se terminan de cargar datos desde un servidor o se activa una cámara conectada.

En ActionScript 3.0, cada evento se representa mediante un objeto de evento, que es una instancia de la clase Event o de alguna de sus subclases. Un objeto de evento no solo guarda información sobre un evento concreto, sino que además contiene métodos que facilitan la manipulación del objeto de evento. Por ejemplo, cuando Flash Player o AIR detectan un clic de ratón, crean un objeto de evento (una instancia de la clase MouseEvent) para representar ese evento de clic de ratón en concreto.

Tras crear un objeto de evento, Flash Player o AIR lo *distribuyen*, lo que significa que el objeto de evento se transmite al objeto que es el destino del evento. El objeto que actúa como destino del objeto de evento distribuido se denomina *destino del evento*. Por ejemplo, cuando se activa una cámara conectada, Flash Player distribuye un objeto de evento directamente al destino del evento que, en este caso, es el objeto que representa la cámara. No obstante, si el destino del evento está en la lista de visualización, el objeto de evento se hace pasar por la jerarquía de la lista de visualización hasta alcanzar el destino del evento. En algunos casos, el objeto de evento se "propaga" de vuelta por la jerarquía de la lista de visualización usando la misma ruta. Este recorrido por la jerarquía de la lista de visualización se denomina *flujo del evento*.

Se pueden usar detectores de eventos en el código para detectar los objetos de evento. Los *detectores de eventos* son las funciones o métodos que se escriben para responder a los distintos eventos. Para garantizar que el programa responde a los eventos, es necesario añadir detectores de eventos al destino del evento o a cualquier objeto de la lista de visualización que forme parte del flujo del evento de un objeto de evento.

Cuando se escribe código para un detector de eventos, se suele seguir esta estructura básica (los elementos en negrita son marcadores de posición que se sustituyen en cada caso específico):

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

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

Este código hace dos cosas. En primer lugar, define una función, que es la forma de especificar las operaciones que se llevarán a cabo como respuesta al evento. A continuación, llama al método `addEventListener()` del objeto de origen, básicamente "suscribiendo" la función al evento especificado de modo que se lleven a cabo las acciones de la función cuando ocurra el evento. Cuando el evento se produce finalmente, el destino de evento comprueba la lista de todas las funciones y métodos registrados como detectores de eventos. A continuación llama a cada uno a su vez, pasando el objeto de evento como parámetro.

Es necesario modificar cuatro elementos del código para crear un detector de eventos personalizado. En primer lugar se debe cambiar el nombre de la función por el nombre que se desee usar (es necesario realizar este cambio en dos lugares, donde en el código aparece **eventResponse**). Seguidamente, hay que especificar el nombre de clase adecuado para el objeto de evento distribuido por el evento que se desea detectar (**EventType** en el código) y hay que indicar la constante apropiada para el evento específico (**EVENT_NAME** en el listado). En tercer lugar, es necesario llamar al método `addEventListener()` en el objeto que distribuirá el evento (**eventTarget** en este código). Opcionalmente, se puede cambiar el nombre de la variable utilizada como parámetro de la función (**eventObject** en el código).

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al escribir rutinas de gestión de eventos:

Propagación Sucede para algunos eventos, de forma que un objeto de visualización principal puede responder a eventos distribuidos por sus elementos secundarios.

Fase de propagación Parte del flujo de eventos en la que un evento se propaga hasta los objetos de visualización principales. Esta fase se produce tras las fases de captura y destino.

Fase de captura Parte del flujo de eventos en la que un evento se propaga desde el destino más general hasta el objeto de destino más específico. Esta fase se produce antes de las fases de propagación y destino.

Comportamiento predeterminado Algunos eventos incluyen un comportamiento que ocurre normalmente junto con el evento, denominado comportamiento predeterminado. Por ejemplo, cuando un usuario escribe texto en un campo de texto, se activa un evento de introducción de texto. El comportamiento predeterminado de ese evento es mostrar el carácter que se ha escrito en el campo de texto, si bien se puede sustituir ese comportamiento predeterminado (si, por alguna razón, no se desea que se muestre el carácter escrito).

Distribuir Notificar a los detectores de eventos que se ha producido un evento.

Evento Algo que le sucede a un objeto y que dicho objeto puede comunicar a otros.

Flujo de eventos Cuando los eventos se producen en un objeto de la lista de visualización (un objeto que se muestra en pantalla), todos los objetos que contienen ese objeto reciben la notificación del evento y, a su vez, notifican a sus detectores de eventos. El proceso comienza en el escenario y avanza a través de la lista de visualización hasta el objeto

en el que se ha producido el evento para después volver de nuevo hasta el escenario. Este proceso se conoce como el flujo del evento.

Objeto Event Objeto que contiene información acerca de un evento en concreto y que se envía a todos los detectores cuando se distribuye un evento.

Destino del evento Objeto que realmente distribuye el evento. Por ejemplo, si el usuario hace clic en un botón que se encuentra dentro de un objeto Sprite que, a su vez, está en el escenario, todos esos objetos distribuirán eventos, pero el destino del evento es el objeto en el que se produjo el evento; en este caso, el botón sobre el que se ha hecho clic.

Detector Función u objeto que se ha registrado a sí mismo con un objeto para indicar que debe recibir una notificación cuando se produzca un evento específico.

Fase de destino Punto del flujo de eventos en el que un evento ha alcanzado el destino posible más específico. Esta fase tiene lugar entre las fases de captura y propagación.

Diferencias entre la gestión de eventos en ActionScript 3.0 y en las versiones anteriores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La diferencia principal entre la gestión de eventos en ActionScript 3.0 y en las versiones anteriores radica en que en ActionScript 3.0 hay un único sistema para gestionar eventos, mientras que en las versiones anteriores de ActionScript existían varios. Esta sección comienza con una descripción general de la forma en la que funcionaba la gestión de eventos en las versiones anteriores de ActionScript y pasa luego a examinar el modo en que ha cambiado en ActionScript 3.0.

Gestión de eventos en versiones anteriores de ActionScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las versiones de ActionScript anteriores a la 3.0 ofrecen diversas formas de gestionar los eventos:

- Controladores de eventos `on()` que se pueden colocar directamente en instancias de Button y MovieClip.
- Controladores `onClipEvent()` que se pueden colocar directamente en instancias de MovieClip.
- Propiedades de funciones callback, como `XML.onload` y `Camera.onActivity`.
- Detectores de eventos que pueden registrarse con el método `addListener()`.
- La clase UIEventDispatcher que implementaba parcialmente el modelo de eventos DOM.

Cada uno de estos mecanismos presenta sus propias ventajas y limitaciones. Los controladores `on()` y `onClipEvent()` son fáciles de usar, pero dificultan el mantenimiento posterior de los proyectos, ya que el código colocado directamente en los botones y clips de películas puede ser difícil de encontrar. Las funciones callback también son sencillas de implementar, pero solo permiten una función callback por evento. La implementación de los detectores de eventos es más compleja, ya que no solo es necesario crear un objeto y una función de detector, sino también registrar el detector en el objeto que genera el evento. No obstante, este trabajo adicional permite crear varios objetos detectores y registrarlos todos para el mismo evento.

El desarrollo de componentes para ActionScript 2.0 dio lugar a un nuevo modelo de eventos. Este nuevo modelo, representado por la clase UIEventDispatcher, se basaba en un subconjunto de la especificación de eventos DOM, de modo que, para los desarrolladores que conozcan la gestión de eventos de componentes, la transición al nuevo modelo de eventos de ActionScript 3.0 resultará relativamente sencilla.

Por desgracia, la sintaxis utilizada en los distintos modelos de eventos es igual en algunos casos y diferente en otros. Por ejemplo, en ActionScript 2.0, algunas propiedades, como `TextField.onChanged`, se pueden usar como función callback o como detector de eventos. Sin embargo, la sintaxis para registrar objetos detectores varía dependiendo de si se utiliza una de las seis clases que admiten detectores en la clase UIEventDispatcher. Para las clases Key, Mouse, MovieClipLoader, Selection, Stage y TextField se debe usar el método `addListener()`, mientras que para la gestión de eventos de componentes es necesario utilizar un método llamado `addEventListener()`.

Otra complicación introducida por los distintos modelos de gestión de eventos es que el ámbito de la función de controlador de eventos varía ampliamente dependiendo del mecanismo usado. Dicho de otro modo, el significado de la palabra clave `this` varía entre los distintos sistemas de gestión de eventos.

Gestión de eventos en ActionScript 3.0

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 presenta un único modelo de gestión de eventos que sustituye a todos los mecanismos que existían en las versiones anteriores del lenguaje. El nuevo modelo de eventos se basa en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3. Si bien el formato de archivo SWF no cumple específicamente con el estándar DOM, existen suficientes similitudes entre la lista de visualización y la estructura de DOM como para posibilitar la implementación del modelo de eventos DOM. Los objetos de la lista de visualización son análogos a los nodos de la estructura jerárquica de DOM y los términos *objeto de la lista de visualización* y *nodo* se usan de forma indistinta en este texto.

La implementación de Flash Player y AIR del modelo de eventos DOM incluye un concepto denominado comportamientos predeterminados. Un *comportamiento predeterminado* es una acción que Flash Player o AIR ejecutan como consecuencia normal de determinados eventos.

Comportamientos predeterminados

Los desarrolladores suelen ser los responsables de escribir código que responda a eventos. No obstante, en algunos casos un comportamiento está asociado con tal frecuencia a un evento, que Flash Player o AIR ejecutan automáticamente ese comportamiento a no ser que el desarrollador incluya código para cancelarlo. Flash Player o AIR muestran comportamientos de este tipo de forma automática, por lo que reciben el nombre de comportamientos predeterminados.

Por ejemplo, cuando un usuario escribe texto en un objeto TextField, resulta tan frecuente esperar que el texto se muestre en el objeto TextField que ese comportamiento se incorpora en Flash Player y AIR. Si no se desea que se produzca este comportamiento predeterminado, es posible cancelarlo usando el nuevo sistema de gestión de eventos. Cuando se escribe texto en un objeto TextField, Flash Player o AIR crean una instancia de la clase TextEvent para representar esa entrada de usuario. Si no se desea que Flash Player o AIR muestren el texto en el objeto TextField, es necesario acceder a esa instancia específica de TextEvent y llamar al método `preventDefault()` de la instancia.

Hay algunos comportamientos predeterminados que no se pueden impedir. Por ejemplo, Flash Player y AIR generan un objeto MouseEvent cuando el usuario hace doble clic en una palabra de un objeto TextField. El comportamiento predeterminado, que no se puede impedir, es resaltar la palabra que hay bajo el cursor.

Hay varios tipos de objetos de evento que no tienen asociado ningún comportamiento predeterminado. Por ejemplo, Flash Player distribuye un objeto de evento connect cuando se establece una conexión de red, pero no hay ningún comportamiento predeterminado asociado a él. La documentación de la API para la clase Event y sus subclases relaciona cada tipo de evento y describe el comportamiento predeterminado que tiene asociado, e indica si es posible impedir dicho comportamiento.

Es importante comprender que los comportamientos predeterminados solo están asociados con objetos de eventos distribuidos por Flash Player o AIR y que no existen para objetos de eventos distribuidos mediante programación a través de ActionScript. Por ejemplo, se pueden usar los métodos de la clase EventDispatcher para distribuir un objeto de evento de tipo `textInput`, pero ese objeto de evento no tendrá ningún comportamiento predeterminado asociado. Es decir, Flash Player y AIR no mostrarán un carácter en un objeto TextField como resultado de un evento `textInput` que se haya distribuido mediante programación.

Novedades de los detectores de eventos de ActionScript 3.0

Para los desarrolladores con experiencia en el uso del método `addListener()` de ActionScript 2.0, puede resultar útil señalar las diferencias entre el modelo de detectores de eventos de ActionScript 2.0 y el de ActionScript 3.0. En la siguiente lista se muestran algunas de las diferencias principales entre los dos modelos de eventos:

- Para añadir detectores de eventos en ActionScript 2.0, es necesario usar `addListener()` en algunos casos y `addEventListener()` en otros, mientras que en ActionScript 3.0 siempre se utiliza `addEventListener()`.
- En ActionScript 2.0 no existe el flujo del evento, lo que quiere decir que el método `addListener()` solo se puede llamar en el objeto que difunde el evento, mientras que en ActionScript 3.0, el método `addEventListener()` se puede llamar en cualquier objeto que forme parte del flujo del evento.
- En ActionScript 2.0, los detectores de eventos pueden ser funciones, métodos u objetos, mientras que en ActionScript 3.0 solo las funciones o los métodos pueden ser detectores de eventos.

Flujo de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

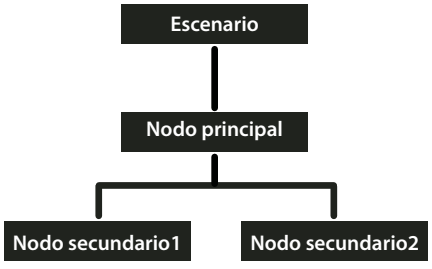
Flash Player o AIR distribuyen objetos de evento siempre que se produce un evento. Si el destino del evento no está en la lista de visualización, Flash Player o AIR distribuyen el objeto de evento directamente al destino del evento. Por ejemplo, Flash Player distribuye el objeto de evento progress directamente a un objeto URLStream. Sin embargo, si el destino del evento está en la lista de visualización, Flash Player distribuye el objeto de evento en la lista de visualización, de modo que recorre la lista hasta llegar al destino del evento.

El *flujo del evento* describe el modo en el que un objeto de evento se desplaza por la lista de visualización. La lista de visualización se organiza en una jerarquía que puede describirse como un árbol. En la parte superior de la jerarquía de la lista de visualización se encuentra el objeto Stage, que es un contenedor de objeto de visualización especial que actúa como raíz de la lista de visualización. El objeto Stage se representa mediante la clase flash.display.Stage y solo es posible acceder a él a través de un objeto de visualización. Todos los objetos de visualización tienen una propiedad llamada `stage` que hace referencia al objeto Stage de esa aplicación.

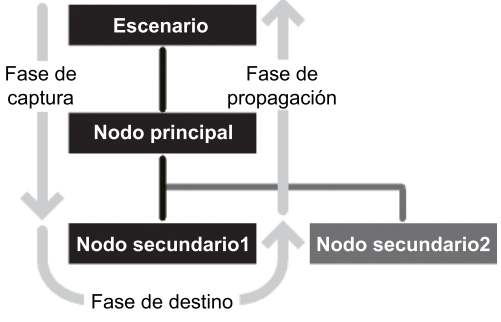
Cuando Flash Player o AIR distribuyen un objeto de evento para un evento relacionado con la lista de visualización, este realiza un viaje de ida y vuelta desde el objeto Stage hasta el *nodo de destino*. La especificación de eventos DOM define el nodo de destino como el nodo que representa el destino del evento. Dicho de otro modo, el nodo de destino es el objeto de la lista de visualización en el que se ha producido el evento. Por ejemplo, si un usuario hace clic en un objeto de lista de visualización denominado `child1`, Flash Player o AIR distribuirán un objeto de evento usando `child1` como nodo de destino.

El flujo del evento se divide conceptualmente en tres partes. La primera parte se llama fase de captura y consta de todos los nodos desde el objeto Stage hasta el elemento principal del nodo de destino. La segunda parte se llama fase de destino y consiste solamente en el nodo de destino. La tercera parte se llama fase de propagación. La fase de propagación consta de los nodos encontrados en el viaje de vuelta desde el elemento principal del nodo de destino hasta el objeto Stage.

Los nombres de las fases cobran más sentido imaginando la lista de visualización como una jerarquía vertical con el objeto Stage en la parte superior, según se muestra en el diagrama siguiente:



Si un usuario hace clic en `Child1 Node`, Flash Player o AIR distribuyen un objeto de evento en el flujo del evento. Como muestra la imagen siguiente, el viaje del objeto empieza en `Stage`, desciende hasta `Parent Node`, luego avanza hasta `Child1 Node` y, finalmente, se propaga de vuelta hasta `Stage` cruzando de nuevo `Parent Node` en su vuelta a `Stage`.



En este ejemplo, la fase de captura consta de `Stage` y `Parent Node` durante el viaje descendente inicial. La fase de destino está compuesta por el tiempo empleado en `Child1 Node`. La fase de propagación consta de `Parent Node` y `Stage`, según se encuentran durante el viaje ascendente de vuelta hasta el nodo raíz.

El flujo del evento contribuye a lograr un sistema de gestión de eventos con mayores prestaciones que el que tenían anteriormente a su disposición los programadores de ActionScript. En las versiones anteriores de ActionScript, el flujo del evento no existe, de modo que los detectores de eventos solo se pueden añadir al objeto que genera el evento. Por contra, en ActionScript 3.0, es posible añadir detectores de eventos no solo a un nodo de destino, sino también a cualquier nodo que pertenezca al flujo del evento.

La posibilidad de añadir detectores de eventos a lo largo del flujo del evento resulta útil cuando un componente de la interfaz de usuario consta de más de un objeto. Por ejemplo, un objeto de botón suele contener un objeto de texto que actúa como etiqueta del botón. Sin la capacidad de añadir un detector al flujo del evento, sería necesario añadir un detector tanto al objeto de botón como al objeto de texto para garantizar que se reciben las notificaciones de eventos de clic que se producen en cualquier punto del botón. No obstante, gracias al flujo del evento es posible colocar un único detector de eventos en el objeto de botón para controlar los eventos de clic que se produzcan tanto en el objeto de texto como en las áreas del objeto de botón que no estén cubiertas por el objeto de texto.

Sin embargo, no todos los objetos participan en las tres fases del flujo del evento. Algunos tipos de eventos, como `enterFrame` e `init`, se distribuyen directamente al nodo de destino y no participan ni en la fase de captura ni en la de propagación. Otros eventos pueden tener como destino objetos que no aparecen en la lista de visualización, como los eventos distribuidos a las instancias de la clase Socket. Estos objetos de evento también van directamente al objeto de destino sin participar en las fases de captura y propagación.

Para conocer el comportamiento de un tipo particular de evento, se puede consultar la documentación de la API o examinar las propiedades del objeto de evento. En la siguiente sección se explica cómo examinar las propiedades del objeto de evento.

Objetos de evento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos de evento tienen una doble finalidad en el nuevo sistema de gestión de eventos. Primero, representan los eventos reales, guardando información acerca de eventos concretos en una serie de propiedades. En segundo lugar, los objetos de evento contienen una serie de métodos que permiten manipular los objetos de evento y afectan al comportamiento del sistema de gestión de eventos.

Para facilitar el acceso a estas propiedades y métodos, la API de Flash Player define una clase Event que constituye la clase base de todos los objetos de evento. La clase Event define una serie fundamental de propiedades y métodos que son comunes a todos los objetos de evento.

Esta sección comienza con una explicación de las propiedades de la clase Event, prosigue con una descripción de los métodos de la clase Event y finaliza con un análisis de las razones por las que existen subclases de la clase Event.

Aspectos básicos de las propiedades de la clase Event

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Event define una serie de propiedades y constantes de solo lectura que proporcionan información importante acerca de un objeto de evento. Las siguientes son especialmente importantes:

- Los tipos de objetos de evento se representan mediante constantes y se almacenan en la propiedad `Event.type`.
- La posibilidad de impedir el comportamiento predeterminado de un evento se representa mediante un valor booleano y se almacena en la propiedad `Event.cancelable`.
- La información del flujo del evento está contenida en las propiedades restantes.

Tipos de objetos de evento

Todos los objetos de evento están asociados a algún tipo de evento. Los tipos de evento se guardan en la propiedad `Event.type` como valores de cadena. Es útil saber de qué tipo es el objeto de evento para que el código pueda distinguir entre objetos de distintos tipos Por ejemplo, el siguiente código especifica que la función de detector `clickHandler()` debe responder a cualquier objeto de evento de clic del ratón que se pase a `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Existen unos veinte tipos de eventos asociados a la clase Event, los cuales se representan con constantes de clase Event, algunas de las cuales se muestran en el siguiente extracto de la definición de la clase Event:

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

Estas constantes proporcionan una forma sencilla de hacer referencia a tipos de eventos específicos. Es aconsejable utilizar estas constantes en lugar de las cadenas a las que representan. Si se escribe incorrectamente el nombre de una constante en el código, el compilador detectará el error, pero si se usan cadenas, un error tipográfico podría pasarse por alto durante la compilación y dar lugar a un comportamiento inesperado difícil de depurar. Por ejemplo, al añadir un detector de eventos, es preferible usar el siguiente código:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

en lugar de:

```
myDisplayObject.addEventListener("click", clickHandler);
```

Información sobre el comportamiento predeterminado

El código puede comprobar si el comportamiento predeterminado para un objeto de evento en particular puede impedirse a través de la propiedad `cancelable`. La propiedad `cancelable` contiene un valor booleano que indica si es posible impedir un comportamiento predeterminado. Para impedir o cancelar el comportamiento predeterminado asociado con algunos eventos se puede utilizar el método `preventDefault()`. Para obtener más información, consulte Cancelación del comportamiento predeterminado de eventos en "Aspectos básicos de los métodos de la clase Event" en la página 136.

Información sobre el flujo del evento

Las demás propiedades de la clase Event contienen información importante acerca de un objeto de evento y su relación con el flujo del evento, según se describe en la siguiente lista:

- La propiedad `bubbles` contiene información acerca de las partes del flujo del evento en el que participa el objeto de evento.
- La propiedad `eventPhase` indica la fase en curso del flujo del evento.
- La propiedad `target` almacena una referencia al destino del evento.
- La propiedad `currentTarget` almacena una referencia al objeto de la lista de visualización que está procesando en ese momento el objeto de evento.

La propiedad bubbles

Se dice que un evento se propaga ("bubbles", en inglés) si su objeto de evento participa en la fase de propagación del flujo del evento, lo que quiere decir que dicho objeto regresa desde el nodo de destino, a través de sus ascendientes, hasta alcanzar el objeto Stage. La propiedad `Event.bubbles` almacena un valor booleano que indica si el objeto de evento participa en la fase de propagación. Dado que todos los eventos que se propagan también participan en las fases de captura y destino, cualquier evento que se propague participa en las tres fases del flujo del evento. Si el valor es `true`, el objeto de evento participa en las tres fases. Si es `false`, el objeto de evento no participa en la fase de propagación.

La propiedad eventPhase

Es posible determinar la fase del evento de un objeto de evento estudiando su propiedad `eventPhase`. La propiedad `eventPhase` contiene un valor entero sin signo que representa una de las tres fases del flujo del evento. La API de Flash Player define una clase EventPhase independiente que contiene tres constantes que se corresponden con los tres valores enteros sin signo, según se muestra en el siguiente extracto de código:

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

Estas constantes corresponden a los tres valores válidos de la propiedad `eventPhase`. Es aconsejable usar estas constantes para hacer que el código sea más legible. Por ejemplo, para asegurarse de que una función denominada `miFunc()` solo se llame si el destino del evento está en la fase de destino, es posible usar el siguiente código para probar dicha condición:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

La propiedad target

La propiedad `target` contiene una referencia al objeto que es el destino del evento. En algunas situaciones esto resulta evidente, como cuando se activa un micrófono, en cuyo caso el destino del objeto de evento es el objeto Microphone. Sin embargo, si el destino se encuentra en la lista de visualización, es necesario tener en cuenta la jerarquía de esta. Por ejemplo, si un usuario hace clic con el ratón en un punto que incluye objetos solapados de la lista de visualización, Flash Player o AIR siempre seleccionan el objeto que se encuentra más lejos del objeto Stage como destino del evento.

En el caso de archivos SWF complejos, especialmente aquellos en los que los botones se decoran sistemáticamente con objetos secundarios menores, la propiedad `target` no puede usarse con demasiada frecuencia, ya que en muchas ocasiones señalará a los objetos secundarios de los botones en lugar de a los propios botones. En estas situaciones, lo habitual es añadir detectores de eventos al botón y usar la propiedad `currentTarget`, ya que esta señala al botón, mientras que la propiedad `target` puede señalar a un elemento secundario del mismo.

La propiedad currentTarget

La propiedad `currentTarget` contiene una referencia al objeto que está procesando en ese momento al objeto de evento. Aunque puede parecer extraño no saber qué nodo está procesando en ese momento al objeto de evento que se está examinando, es necesario recordar que se puede añadir una función de detector a cualquier objeto de visualización en el flujo del evento de ese objeto de evento y que dicha función puede colocarse en cualquier lugar. Además, es posible añadir la misma función de detector a distintos objetos de visualización. A medida que el tamaño y la complejidad de un proyecto crecen, la propiedad `currentTarget` resulta cada vez más útil.

Aspectos básicos de los métodos de la clase Event

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen tres categorías de los métodos de la clase Events:

- Métodos de utilidad, que pueden crear copias de un objeto de evento o convertirlo en una cadena.
- Métodos de flujo del evento, que eliminan objetos de evento del flujo del evento.
- Métodos de comportamiento predeterminado, que impiden que se produzca el comportamiento predeterminado o comprueban si se ha impedido

Métodos de utilidad de la clase Event

Existen dos métodos de utilidad en la clase Event. El método `clone()` permite crear copias de un objeto de evento. El método `toString()` permite generar una representación en forma de cadena de las propiedades de un objeto de evento, junto con sus valores. El sistema de modelos de evento usa ambos métodos internamente, pero están a disposición de los desarrolladores para el uso general.

Los desarrolladores avanzados que deseen crear subclases de la clase Event deben sustituir e implementar versiones de ambos métodos de utilidad para asegurarse de que la subclase de eventos funcionará correctamente.

Detener el flujo del evento

Se puede llamar al método `Event.stopPropagation()` o `Event.stopImmediatePropagation()` para impedir que un objeto de evento siga moviéndose por el flujo del evento. Ambos métodos son casi idénticos y solo se diferencian en que uno permite que se ejecuten los demás detectores de eventos del nodo en curso y el otro no:

- El método `Event.stopPropagation()` impide que el objeto de evento avance hasta el siguiente nodo, pero solo después de permitir que se ejecuten todos los demás detectores de eventos del nodo en curso.
- El método `Event.stopImmediatePropagation()` también impide que el objeto de evento avance hasta el siguiente nodo, pero no permite que se ejecute ningún otro detector de eventos del nodo en curso.

Las llamadas a cualquiera de estos dos métodos no afectan a la aplicación del comportamiento predeterminado asociado a un evento. Es necesario usar los métodos de comportamiento predeterminado de la clase Event para impedir dicho comportamiento.

Cancelación del comportamiento predeterminado de un evento

Para cancelar el comportamiento predeterminado hay dos métodos: `preventDefault()` y `isDefaultPrevented()`. Para cancelar el comportamiento predeterminado asociado con un evento, llame al método `preventDefault()`. mientras que para comprobar si ya se ha llamado a `preventDefault()` en un objeto de evento, es necesario llamar al método `isDefaultPrevented()`, que devuelve el valor `true` si ya se ha llamado al método o `false` en caso contrario.

El método `preventDefault()` solo funcionará si es posible cancelar el comportamiento predeterminado del evento. Se puede comprobar si esto es posible consultando la documentación de la API para ese tipo de evento o usando ActionScript para examinar la propiedad `cancelable` del objeto de evento.

La cancelación del comportamiento predeterminado no afecta al progreso de un objeto de evento en el flujo de eventos. Si desea eliminar un objeto de evento del flujo de eventos, utilice los métodos de flujo de eventos de la clase Event.

Subclases de la clase Event

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para muchos eventos es suficiente el conjunto común de propiedades que se definen en la clase Event. No obstante, otros eventos tienen características únicas que no es posible capturar mediante las propiedades de la clase Event. Para estos eventos, ActionScript 3.0 define varias subclases de la clase Event.

Cada subclase brinda propiedades y tipos de eventos adicionales que son exclusivos para esa categoría de eventos. Por ejemplo, los eventos relacionados con la entrada del ratón tienen algunas características únicas que no es posible capturar mediante las propiedades definidas en la clase Event. La clase MouseEvent amplía la clase Event añadiendo diez propiedades que contienen datos como la ubicación del evento de ratón y si se presionaron teclas específicas durante dicho evento.

Las subclases de Event también contienen constantes que representan los tipos de eventos asociados a la subclase. Por ejemplo, la clase MouseEvent define constantes para varios tipos de eventos de ratón e incluye los tipos de evento `click`, `doubleClick`, `mouseDown` y `mouseUp`.

Tal como se describe en la sección Métodos de utilidad de la clase Event en "Objetos de evento" en la página 133, al crear una subclase de Event es necesario sustituir los métodos `clone()` y `toString()` para ofrecer una funcionalidad específica de la subclase.

Detectores de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los detectores de eventos, también llamados controladores de eventos, son funciones que ejecutan Flash Player y AIR como respuesta a eventos específicos. El proceso de añadir un detector de eventos consta de dos pasos. En primer lugar se debe crear una función o método de clase para que Flash Player o AIR lo ejecuten como respuesta al evento. Esto a veces recibe el nombre de función de detector o función de controlador de eventos. En segundo lugar, es necesario usar el método `addEventListener()` para registrar la función de detector en el destino del evento o en cualquier objeto de la lista de visualización que se encuentre en el trayecto del flujo del evento adecuado.

Creación de funciones de detector

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La creación de funciones de detector es un área en la que el modelo de eventos de ActionScript 3.0 difiere del modelo de eventos DOM. En el modelo de eventos DOM existe una distinción clara entre un detector de eventos y una función de detector: un detector de eventos es una instancia de una clase que implementa la interfaz EventListener, mientras que una función de detector es un método de esa clase denominada `handleEvent()`. En el modelo de eventos DOM, se debe registrar la instancia de la clase que contiene la función de detector en lugar de la función de detector en sí.

En el modelo de eventos de ActionScript 3.0 no hay distinción entre un detector de eventos y una función de detector. ActionScript 3.0 carece de una interfaz EventListener y es posible definir las funciones de detector fuera de una clase o como parte de ella. Además, no es necesario que las funciones de detector se denominen `handleEvent()`, sino que pueden usar cualquier identificador válido. En ActionScript 3.0 se registra el nombre de la función de detector real.

Función de detector definida fuera de una clase

El siguiente código crea un archivo SWF sencillo que muestra una forma cuadrada roja. Una función de detector denominada `clickHandler()`, que no forma parte de ninguna clase, detecta los eventos de clic del ratón sobre el cuadrado rojo.

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

Cuando el usuario interactúa con el archivo SWF resultante haciendo clic en el cuadrado, Flash Player o AIR generan la siguiente salida de traza:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Cabe destacar que el objeto de evento se pasa como argumento a `clickHandler()`. Esto permite a la función de detector examinar el objeto de evento. En este ejemplo se usa la propiedad `type` del objeto de evento para determinar si se trata de un evento de clic.

En el ejemplo también se comprueba el valor de la palabra clave `this`. En este caso, `this` representa el objeto global, lo cual es totalmente lógico, ya que la función se ha definido fuera de todo objeto o clase personalizada.

Función de detector definida como método de clase

El siguiente ejemplo es idéntico al anterior en el que se define la clase ClickExample, excepto en que la función `clickHandler()` se define como un método de la clase ChildSprite:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

Cuando el usuario interactúa con el archivo SWF resultante haciendo clic en el cuadrado rojo, Flash Player o AIR generan la siguiente salida de traza:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

La palabra clave `this` hace referencia a la instancia de ChildSprite denominada `child`. Se trata de un cambio de comportamiento respecto a ActionScript 2.0. Los usuarios que usaban componentes en ActionScript 2.0 sin duda recordarán que, al pasar un método de clase a `UIEventDispatcher.addEventListener()`, el ámbito del método estaba vinculado al componente que difundía el evento en lugar de a la clase en la que estaba definido el método de detector. Dicho de otro modo, si se usara esta técnica en ActionScript 2.0, la palabra clave `this` haría referencia al componente que realiza la difusión en lugar de a la instancia de ChildSprite.

Esto constituía un problema significativo para algunos programadores, ya que implicaba que no podían acceder a otros métodos y propiedades de la clase que contenía el método detector. Como solución, los programadores de ActionScript 2.0 podían usar la clase `mx.util.Delegate` para cambiar el ámbito del método detector. Esto ya no es necesario, ya que ActionScript 3.0 crea un método vinculado cuando se llama a `addEventListener()`. A consecuencia de ello, la palabra clave `this` hace referencia a la instancia de ChildSprite denominada `child` y el programador puede acceder a los demás métodos y propiedades de la clase ChildSprite.

Detector de eventos de uso no recomendado

Existe una tercera técnica, que no se recomienda, en la que se crea un objeto genérico con una propiedad que señala a una función de detector asignada dinámicamente. Se analiza aquí porque se solía utilizar en ActionScript 2.0, aunque no se debe emplear en ActionScript 3.0. Esta técnica no se recomienda, ya que la palabra clave `this` hará referencia al objeto global en lugar de al objeto detector.

El siguiente ejemplo es idéntico al anterior de la clase ClickExample, excepto en que la función de detector se define como parte de un objeto genérico denominado `myListenerObj`:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

El resultado de la traza es el siguiente:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Cabría esperar que `this` hiciera referencia a `myListenerObj` y que la salida de traza fuese `[object Object]` pero, en vez de eso, hace referencia al objeto global. Al pasar el nombre de una propiedad dinámica como un argumento a `addEventListener()`, Flash Player o AIR no pueden crear un método vinculado. Esto se debe a que lo que se transmite como parámetro `listener` no es más que la dirección de memoria de la función de detector y Flash Player y AIR son incapaces de vincular esa dirección de memoria con la instancia de `myListenerObj` .

Administración de detectores de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Es posible administrar las funciones de detector usando los métodos de la interfaz IEventDispatcher. La interfaz IEventDispatcher es la versión de ActionScript 3.0 de la interfaz EventTarget del modelo de eventos DOM. Si bien el nombre IEventDispatcher parece implicar que su objetivo principal es enviar (o distribuir) objetos de evento, en realidad los métodos de esta clase se usan con mucha más frecuencia para registrar detectores de eventos, comprobar su existencia y eliminarlos. La interfaz IEventDispatcher define cinco métodos, según se muestra en el siguiente código:

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

La API de Flash Player implementa la interfaz IEventDispatcher con la clase EventDispatcher, que sirve como clase base de todas las clases que pueden ser destinos de eventos o parte de un flujo de eventos. Por ejemplo, la clase DisplayObject hereda de la clase EventDispatcher. Esto quiere decir que cualquier objeto de la lista de visualización tiene acceso a los métodos de la interfaz IEventDispatcher.

Añadir detectores de eventos

El método `addEventListener()` es el elemento más ampliamente utilizado de la interfaz IEventDispatcher. Se utiliza para registrar las funciones de detector. Los dos parámetros necesarios son `type` y `listener`. Se puede usar el parámetro `type` para especificar el tipo de evento. El parámetro `listener`, por su parte, se emplea para especificar la función de detector que se ejecutará cuando se produzca el evento. El parámetro `listener` puede ser una referencia a una función o a un método de clase.

No utilice paréntesis al especificar el parámetro `listener`. Por ejemplo, la función `clickHandler()` se especifica sin paréntesis en la siguiente llamada al método `addEventListener()`:

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

El parámetro `useCapture` del método `addEventListener()` permite controlar la fase del flujo del evento en la que estará activa el detector. Si `useCapture` se establece en `true`, el detector estará activo durante la fase de captura del flujo del evento. Si `useCapture` se establece en `false`, el detector estará activo durante las fases de destino y de propagación del flujo del evento. Para detectar un evento durante todas las fases del flujo del evento se debe llamar a `addEventListener()` dos veces, una con `useCapture` establecido en `true` y luego otra con `useCapture` establecido en `false`.

El parámetro `priority` del método `addEventListener()` no es parte oficial del modelo de eventos DOM de nivel 3. Se incluye en ActionScript 3.0 para ofrecer una mayor flexibilidad a la hora de organizar los detectores de eventos. Cuando se llama a `addEventListener()`, es posible establecer la prioridad de ese detector de eventos pasando un valor entero como parámetro `priority`. El valor predeterminado es 0, pero se le pueden asignar valores enteros negativos o positivos. Cuanto mayor sea el número, antes se ejecutará el detector de eventos. Los detectores de eventos con la misma prioridad se ejecutan en el orden en que se añadieron, de modo que cuanto antes se añada, antes se ejecutará.

El parámetro `useWeakReference` permite especificar si la referencia a la función de detector es débil o normal. Si este parámetro se establece como `true` se pueden evitar situaciones en las que las funciones de detector permanecen en memoria incluso cuando ya no se necesitan. Flash Player y AIR usan una técnica denominada *eliminación de datos innecesarios* para borrar de la memoria objetos que han dejado de utilizarse. Se considera que un objeto ha dejado de usarse si no existe ninguna referencia a él. El recolector de datos innecesarios descarta las referencias débiles, de modo que una función de detector que solo señala a una referencia débil puede ser eliminada, al considerarse como datos innecesarios.

Eliminación de detectores de eventos

El método `removeEventListener()` sirve para eliminar un detector de eventos que ya no se necesita. Siempre conviene eliminar los detectores que no se vayan a usar más. Los parámetros obligatorios incluyen `eventName` y `listener`, los mismos que para el método `addEventListener()`. Conviene recordar que se pueden detectar eventos durante todas las fases de eventos llamando a `addEventListener()` dos veces, una vez con `useCapture` establecido en `true` y luego otra establecido en `false`. Para eliminar los dos detectores de eventos sería necesario llamar a `removeEventListener()` dos veces, una con `useCapture` establecido en `true` y luego otra establecido en `false`.

Distribución de eventos

Los programadores expertos pueden usar el método `dispatchEvent()` para distribuir un objeto de evento personalizado en el flujo del evento. Este método solo acepta un parámetro, consistente en una referencia a un objeto de evento, que debe ser una instancia de la clase Event o una subclase de esta. Una vez distribuido, la propiedad `target` del objeto de evento se establece en el objeto en el que se llamó a `dispatchEvent()`.

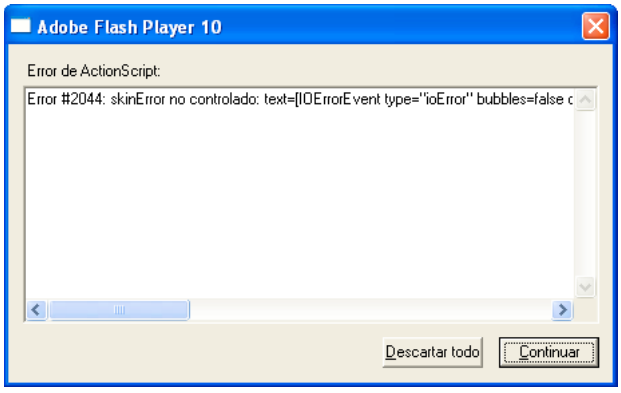
Comprobación de detectores de eventos existentes

Los dos últimos métodos de la interfaz IEventDispatcher ofrecen información útil sobre la existencia de detectores de eventos. El método `hasEventListener()` devuelve `true` si se encuentra un detector de eventos para un tipo de evento específico en un objeto concreto de la lista de visualización. El método `willTrigger()` también devuelve `true` si se encuentra un detector para un objeto concreto de la lista de visualización, pero `willTrigger()` no solo comprueba la existencia de detectores en el objeto de la lista de visualización, sino también en todos los ascendientes del objeto de la lista de visualización para todas las fases del flujo del evento.

Eventos de error sin detectores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las excepciones, y no los eventos, son el principal mecanismo de gestión de errores en ActionScript 3.0, pero la gestión de excepciones no funciona en operaciones asíncronas, como la carga de archivos. Si se produce un error durante una de estas operaciones asíncronas, Flash Player y AIR distribuyen un objeto de evento de error. Si no se crea un detector para el evento de error, las versiones de depuración de Flash Player y AIR mostrarán un cuadro de diálogo con información acerca del error. Por ejemplo, la versión del depurador de Flash Player genera el siguiente cuadro de diálogo que describe el error cuando la aplicación intenta cargar un archivo desde una dirección URL no válida:



La mayor parte de los eventos de error se basan en la clase ErrorEvent y, por lo tanto, tienen una propiedad denominada `text` que se usa para almacenar el mensaje de error que muestra Flash Player o AIR. Las dos excepciones a esta regla son las clases StatusEvent y NetStatusEvent. Ambas clases tienen una propiedad `level` (`StatusEvent.level` y `NetStatusEvent.info.level`). Cuando el valor de la propiedad `level` es "`error`", estos tipos de evento se consideran eventos de error.

Un evento de error no hace que el archivo SWF deje de ejecutarse. Solo se manifestará como un cuadro de diálogo en las versiones de depuración de los plugins de navegadores y de los reproductores autónomos, como un mensaje en el panel de salida del reproductor de edición y como una entrada en el archivo de registro de Adobe Flash Builder. No se manifiesta en las versiones oficiales de Flash Player o AIR.

Ejemplo de gestión de eventos: Reloj con alarma

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo del reloj con alarma consiste en un reloj que permite al usuario especificar una hora a la que sonará una alarma y se mostrará un mensaje. El ejemplo del reloj con alarma se basa en la aplicación SimpleClock de "Trabajo con fechas y horas" en la página 1 El ejemplo ilustra diversos aspectos de la utilización de eventos en ActionScript 3.0 como, por ejemplo:

- Detección y respuesta a un evento
- Notificación a los detectores de un evento
- Creación de un tipo de evento personalizado

Para obtener los archivos de aplicación de Flash Professional para esta muestra, consulte http://www.adobe.com/go/learn_programmingAS3samples_flash_es. Para obtener los archivos de aplicación de Flex para esta muestra, consulte http://www.adobe.com/go/as3examples_es. Los archivos de la aplicación del reloj con alarma se encuentran en la carpeta Samples/AlarmClock. La aplicación consta de los siguientes archivos:

Archivo	Descripción
AlarmClockApp.mxml o AlarmClockApp.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/clock/AlarmClock.as	Una clase que amplía la clase SimpleClock y añade la función de alarma al reloj.
com/example/programmingas3/clock/AlarmEvent.as	Una clase de eventos personalizada (una subclase de flash.events.Event) que actúa como el objeto de evento del evento `alarm` de la clase AlarmClock.
com/example/programmingas3/clock/AnalogClockFace.as	Dibuja una esfera de reloj redonda y las manecillas de hora, minutos y segundos, en función de la hora (descrita en el ejemplo SimpleClock).
com/example/programmingas3/clock/SimpleClock.as	Un componente de la interfaz de reloj con funciones sencillas de control de tiempo (descrito en el ejemplo de SimpleClock).

Información general sobre el reloj con alarma

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para la funcionalidad principal del reloj de este ejemplo, incluido el control del tiempo y la visualización de la esfera del reloj, se vuelve a utilizar el código de la aplicación SimpleClock, que se describe en "Ejemplo de fecha y hora: Un sencillo reloj analógico" en la página 6. La clase AlarmClock amplía la clase SimpleClock del ejemplo añadiendo la funcionalidad necesaria para un reloj con alarma, incluido el ajuste de la hora de la alarma y la notificación cuando suena la alarma.

Los eventos están diseñados para proporcionar notificaciones cuando ocurre algo. La clase AlarmClock expone el evento Alarm, sobre el que los demás objetos pueden realizar detecciones a fin de llevar a cabo las acciones deseadas. Además, la clase AlarmClock usa una instancia de la clase Timer para determinar cuándo hay que activar la alarma. Al igual que la clase AlarmClock, la clase Timer proporciona un evento para notificar a los demás objetos (una instancia de AlarmClock en este caso) cuándo ha transcurrido una determinada cantidad de tiempo. Tal y como ocurre con la mayoría de las aplicaciones de ActionScript, los eventos forman una parte importante de la funcionalidad de la aplicación del ejemplo del reloj con alarma.

Activación de la alarma

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Según se ha mencionado con anterioridad, la única funcionalidad que la clase AlarmClock ofrece realmente está relacionada con la configuración y activación de la alarma. La clase integrada Timer (flash.utils.Timer) proporciona un mecanismo para que los desarrolladores definan código que se ejecutará tras un periodo de tiempo especificado. La clase AlarmClock utiliza una instancia de Timer para determinar cuándo activar la alarma.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

La instancia de Timer definida en la clase AlarmClock se denomina `alarmTimer`. El método `initClock()`, que lleva a cabo las operaciones de configuración necesarias para la instancia de AlarmClock, realiza dos operaciones con la variable `alarmTimer`. En primer lugar, se crea una instancia de la variable con parámetros que indican a la instancia Timer que debe esperar 0 milisegundos y activar su evento de temporizador solo una vez. Tras crear la instancia de `alarmTimer`, el código llama al método `addEventListener()` de la variable para indicar que desea realizar una detección en el evento `timer` de esa variable. Las instancias de Timer funcionan distribuyendo su evento `timer` una vez transcurrida una cantidad de tiempo especificada. La clase AlarmClock necesitará saber cuándo se distribuye el evento `timer` para activar su propia alarma. Al llamar a `addEventListener()`, el código de AlarmClock se registra como detector con `alarmTimer`. Los dos parámetros indican que la clase AlarmClock desea detectar el evento `timer` (indicado por la constante `TimerEvent.TIMER`) y que, cuando este se produzca, debe llamarse al método `onAlarm()` de la clase AlarmClock como respuesta al evento.

Para establecer la alarma, se llama al método `setAlarm()` de la clase AlarmClock del siguiente modo:

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

Este método realiza varias tareas, entre ellas almacenar el mensaje de alarma y crear un objeto Date (`alarmTime`) que representa el instante de tiempo real en el que debe sonar la alarma. En las últimas líneas del método, el temporizador de la variable `alarmTimer` se define y se activa, lo que resulta de especial relevancia para este análisis. En primer lugar, se llama a su método `reset()`, deteniendo el temporizador y reiniciándolo en caso de que ya se estuviese ejecutando. A continuación, se resta la hora actual (representada por la variable `now`) del valor de la variable `alarmTime` para determinar cuántos milisegundos tienen que transcurrir antes de que suene la alarma. La clase Timer no activa su evento `timer` a una hora absoluta, de manera que es esta diferencia relativa de tiempo la que se asigna a la propiedad `delay` de `alarmTimer`. Finalmente, se llama al método `start()` para iniciar el temporizador.

Una vez que ha transcurrido la cantidad especificada de tiempo, `alarmTimer` distribuye el evento `timer`. Dado que la clase AlarmClock ha registrado su método `onAlarm()` como un detector de ese evento, cuando se produzca el evento `timer` se llamará a `onAlarm()`.

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

Un método que se registra como detector de eventos debe definirse con la firma adecuada (es decir, el conjunto de parámetros y el tipo de devolución del método). Para que un método pueda ser detector del evento `timer` de la clase Timer, debe definir un parámetro cuyo tipo de datos sea TimerEvent (flash.events.TimerEvent), una subclase de la clase Event. Cuando la instancia de Timer llama a sus detectores de eventos, pasa una instancia TimerEvent como objeto de evento.

Notificación de la alarma a otros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al igual que la clase Timer, la clase AlarmClock proporciona un evento que permite que otro código reciba notificaciones cuando suena la alarma. Para que una clase pueda usar el marco de gestión de eventos incorporado en ActionScript, debe implementar la interfaz flash.events.IEventDispatcher. Normalmente esto se lleva a cabo ampliando la clase flash.events.EventDispatcher, que proporciona una implementación estándar de IEventDispatcher (o ampliando una de las subclases de EventDispatcher). Tal y como se ha descrito anteriormente, la clase AlarmClock amplía la clase SimpleClock, que (a través de una cadena de herencias) amplía la clase EventDispatcher. Todo esto quiere decir que la clase AlarmClock ya incorpora la funcionalidad adecuada para proporcionar sus propios eventos.

Se puede registrar otro código para que reciba notificaciones del evento `alarm` de la clase AlarmClock llamando al método `addEventListener()` que AlarmClock hereda de EventDispatcher. Cuando una instancia de AlarmClock está lista para notificar a otro código que su evento `alarm` se ha activado, lleva a cabo esta operación llamando al método `dispatchEvent()`, que también se hereda de EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Estas líneas de código están tomadas del método `onAlarm()` de la clase AlarmClock (que se ha mostrado al completo anteriormente). Se llama al método `dispatchEvent()` de la instancia de AlarmClock que, a su vez, notifica a todos los detectores registrados que el evento `alarm` de la instancia de AlarmClock se ha activado. El parámetro que se pasa a `dispatchEvent()` es el objeto de evento que se pasará a los métodos detectores. En este caso es una instancia de la clase AlarmEvent, una subclase de Event creada específicamente para este ejemplo.

Creación de un evento de alarma personalizado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todos los detectores de eventos reciben un parámetro de objeto de evento con información acerca del evento específico que se está activando. En muchos casos, el objeto de evento es una instancia de la clase Event. No obstante, en algunas ocasiones resulta útil proporcionar más información a los detectores de eventos. Una forma habitual de lograr esto es definir una nueva clase (una subclase de la clase Event) y usar una instancia de esa clase como objeto de evento. En este ejemplo, se usa una instancia de AlarmEvent como objeto de evento cuando se distribuye el evento `alarm` de la clase AlarmClock. La clase AlarmEvent, que se muestra aquí, ofrece más información acerca del evento `alarm`, específicamente el mensaje de alarma:

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

La mejor forma de crear una clase de objetos de evento personalizados es definir una clase que amplíe la clase Event, según se muestra en el ejemplo anterior. Para complementar la funcionalidad heredada, la clase AlarmEvent define una propiedad `message` que contiene el texto del mensaje de alarma asociado al evento; el valor de `message` se pasa como un parámetro en el constructor de AlarmEvent. La clase AlarmEvent también define la constante `ALARM`, que se puede utilizar para hacer referencia al evento específico (`alarm`) al llamar al método `addEventListener()` de la clase AlarmClock.

Además de añadir funcionalidad personalizada, cada subclase de Event debe sustituir el método `clone()` heredado como parte del marco de gestión de eventos de ActionScript. Las subclases Event también pueden sustituir el método heredado `toString()` para incluir las propiedades del evento personalizado en el valor que se devuelve al llamar al método `toString()`.

```
    /**
     * Creates and returns a copy of the current instance.
     * @return A copy of the current instance.
     */
    public override function clone():Event
    {
        return new AlarmEvent(message);
    }

    /**
     * Returns a String containing all the properties of the current
     * instance.
     * @return A string representation of the current instance.
     */
    public override function toString():String
    {
        return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase",
"message");
    }
```

El método `clone()` sustituido necesita devolver una nueva instancia de la subclase personalizada de Event con todas las propiedades personalizadas definidas para coincidir con la instancia actual. En el método `toString()` sustituido, el método de utilidad `formatToString()` (heredado de Event) se utiliza para proporcionar una cadena con el nombre del tipo personalizado, además de los nombres y valores de todas sus propiedades.

Capítulo 9: Trabajo con dominios de aplicación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El propósito de la clase ApplicationDomain es almacenar una tabla de definiciones de ActionScript 3.0. Todo el código de un archivo SWF se define como existente en un dominio de aplicación. Los dominios de aplicación se utilizan para hacer particiones de clases en el mismo dominio de seguridad. Esto permite que coexistan varias definiciones de la misma clase y que los elementos secundarios puedan reutilizar definiciones de elementos principales.

Se pueden utilizar dominios de aplicación al cargar un archivo SWF externo programado en ActionScript 3.0 mediante la API de la clase Loader. (Hay que tener en cuenta que no se pueden utilizar dominios de aplicación al cargar una imagen o un archivo SWF programado en ActionScript 1.0 o ActionScript 2.0.) Todas las definiciones de ActionScript 3.0 contenidas en la clase cargada se almacenan en el dominio de aplicación. Al cargar el archivo SWF, se puede especificar que se incluya el archivo en el mismo dominio de aplicación que el del objeto Loader, estableciendo el parámetro `applicationDomain` del objeto LoaderContext en `ApplicationDomain.currentDomain`. Al colocar el archivo SWF cargado en el mismo dominio de aplicación, se puede acceder a sus clases directamente. Esto puede resultar útil si se carga un archivo SWF que contiene medios incorporados, a los que se puede acceder a través de sus nombres de clases asociados, o si se desea acceder métodos del archivo SWF cargado.

En el siguiente ejemplo se supone que se tiene acceso a un archivo Greeter.swf independiente que define un método público denominado welcome():

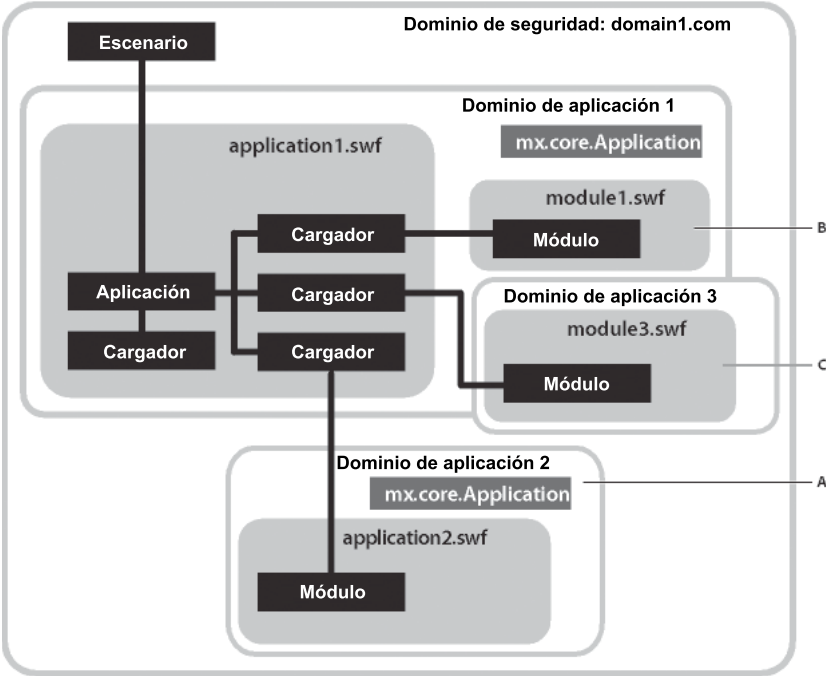
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false,
ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            var myGreeter:Class = ApplicationDomain.currentDomain.getDefinition("Greeter") as
Class;
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Consulte también el ejemplo de la clase ApplicationDomain en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Otras cosas que hay que tener en cuenta al trabajar con dominios de aplicación son las siguientes:

- Todo el código de un archivo SWF se define como existente en un dominio de aplicación. El *dominio* actual es el lugar en el que se ejecuta la aplicación principal. El *dominio del sistema* contiene todos los dominios de la aplicación, incluido el dominio actual, lo que significa que contiene todas las clases de Flash Player.
- Todos los dominios de aplicación tienen asociado un dominio principal, excepto el dominio del sistema. El dominio principal del dominio de aplicación principal es el dominio del sistema. Solo es necesario definir las clases cargadas si su clase principal no las ha definido todavía. No es posible anular una definición de clase cargada con otra definición más reciente.

En el diagrama siguiente se muestra una aplicación que carga contenido de diversos archivos SWF de un solo dominio, domain1.com. En función del contenido que se cargue, se pueden utilizar distintos dominios de aplicación. El texto siguiente describe la lógica que se utiliza para establecer el dominio de aplicación apropiado para cada archivo SWF de la aplicación.



A. *Uso A* ***B.*** *Uso B* ***C.*** *Uso C*

El archivo de aplicación principal es application1.swf. Contiene objetos Loader que cargan contenido de otros archivos SWF. En este escenario, el dominio actual es Application domain 1. Los usos A, B y C ilustran distintas técnicas para establecer el dominio de aplicación apropiado para cada archivo SWF de una aplicación.

Uso A Dividir el archivo SWF secundario creando un elemento secundario del dominio del sistema. En el diagrama, el dominio de aplicación 2 se crea como un elemento secundario del dominio del sistema. El archivo application2.swf se carga en el dominio de aplicación 2 y sus definiciones de clase se dividen de las clases definidas en application1.swf.

Una aplicación de esta técnica es hacer que una aplicación antigua cargue dinámicamente una versión más reciente de la misma aplicación sin conflictos. No hay conflictos porque, aunque se usen los mismos nombres de clase, se dividen en distintos dominios de aplicación.

El código siguiente crea un dominio de aplicación que es un elemento secundario del dominio del sistema y comienza a cargar un archivo SWF que utiliza ese dominio de aplicación:

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

Uso B: Añadir nuevas definiciones de clase a las definiciones de clase actuales. El dominio de aplicación de module1.swf se establece en el dominio actual (dominio de aplicación 1). Esto permite añadir nuevas definiciones de clase al conjunto actual de definiciones de clase de la aplicación. Esto se puede utilizar para una biblioteca compartida en tiempo de ejecución de la aplicación principal. El archivo SWF cargado se trata como una biblioteca remota compartida (RSL). Esta técnica se utiliza para cargar bibliotecas RSL mediante un precargador antes de que se inicie la aplicación.

El código siguiente carga un archivo SWF, estableciendo su dominio de aplicación en el dominio actual:

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

Uso C: Utilizar las definiciones de clase del elemento principal creando un nuevo dominio secundario del dominio actual. El dominio de aplicación de module3.swf es un elemento secundario del dominio actual y el elemento secundario utiliza las versiones del elemento principal de todas las clases. Una aplicación de esta técnica podría ser un módulo de una aplicación de Internet compleja de varias pantallas, cargada como un elemento secundario de la aplicación principal y que utiliza los tipos de la aplicación principal. Si hay la seguridad de que siempre se actualizarán todas las clases para ser compatibles con versiones anteriores y que la aplicación de carga es siempre más reciente que los elementos que carga, los elementos secundarios utilizarán las versiones del elemento principal. Tener un nuevo dominio de aplicación también permite descargar todas las definiciones de clase para eliminar datos innecesarios, si hay la seguridad de que no siguen existiendo referencias al archivo SWF secundario.

Esta técnica permite que los módulos cargados compartan los objetos singleton del cargador y los miembros de clase estáticos.

El código siguiente crea un nuevo dominio secundario del dominio actual y comienza a cargar un archivo SWF que utiliza ese dominio de aplicación:

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

Capítulo 10: Programación de la visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

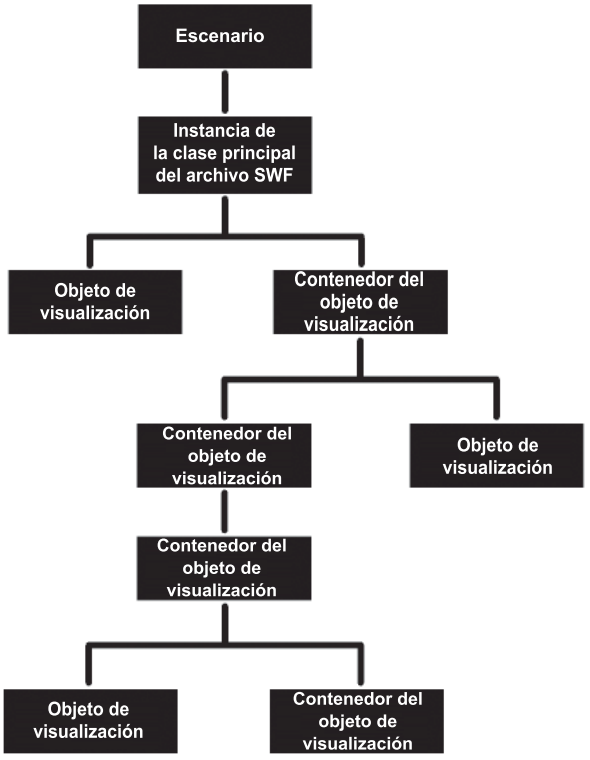
Los elementos visuales se programan en Adobe® ActionScript® 3.0 trabajando con objetos de visualización en el escenario de visualización. Por ejemplo, es posible añadir, mover, eliminar y ordenar objetos de visualización, aplicar filtros y máscaras, dibujar gráficos de mapa de bits y vectoriales y realizar transformaciones tridimensionales utilizando la API de programación de visualización de ActionScript. Las clases principales utilizadas para la programación de visualización forman parte del paquete flash.display.

Nota: *Adobe® AIR™ proporciona el objeto HTMLLoader para procesar y visualizar contenido HTML. HTMLLoader procesa los elementos visuales del DOM de HTML como un solo objeto de visualización. No es posible acceder a los elementos individuales del DOM directamente mediante la jerarquía de la lista de visualización de ActionScript. A estos elementos de DOM se accede utilizando la API de DOM independiente que proporciona HTMLLoader.*

Fundamentos de la programación de la visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cada aplicación creada con ActionScript 3.0 tiene una jerarquía de objetos visualizados, conocida como *lista de visualización*, tal y como se muestra a continuación. La lista de visualización contiene todos los elementos visibles en la aplicación.



Tal y como muestra la ilustración, los elementos en pantalla se ubican en uno o varios de los siguientes grupos:

- Objeto Stage

  El objeto Stage es el contenedor base de los objetos de visualización. Cada aplicación tiene un objeto Stage, que contiene todos los objetos de visualización en pantalla. El objeto Stage es el contenedor de nivel superior y se encuentra arriba del todo en la jerarquía de la lista de visualización:

  Cada archivo SWF tiene una clase de ActionScript asociada, conocida como *la clase principal del archivo SWF*. Cuando se abre un archivo SWF en Flash Player o en Adobe AIR, Flash Player o AIR llama a la función constructora de dicha clase y la instancia que se crea (que es siempre un tipo de objeto de visualización) se añade como elemento secundario del objeto Stage. La clase principal de un archivo SWF siempre amplía la clase Sprite (para obtener más información, consulte "Ventajas de la utilización de la lista de visualización" en la página 160).

  Es posible acceder al objeto Stage a través de la propiedad `stage` de ninguna instancia de DisplayObject. Para obtener más información, consulte "Configuración de las propiedades de Stage" en la página 169.

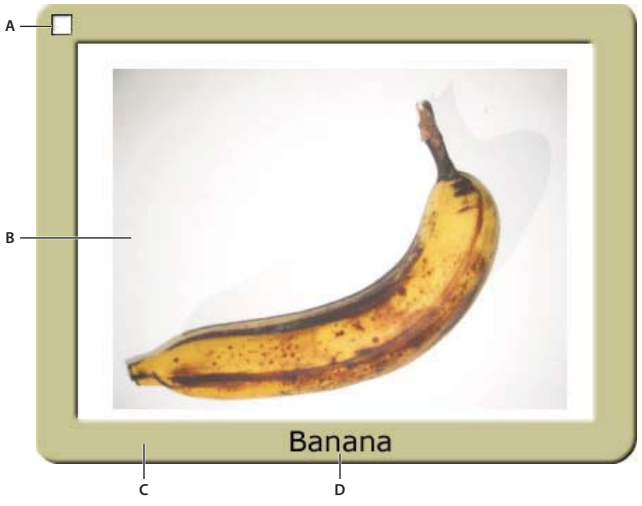
- Objetos de visualización

En ActionScript 3.0, todos los elementos que aparecen en la pantalla de una aplicación son tipos de *objetos de visualización*. El paquete flash.display incluye una clase DisplayObject . Se trata de una clase base ampliada por otras. Estas distintas clases representan tipos diferentes de objetos de visualización, como formas vectoriales, clips de película o campos de texto, entre otros. Para obtener información general de estas clases, consulte "Ventajas de la utilización de la lista de visualización" en la página 160.

- Contenedores de objetos de visualización

  Los contenedores de objetos de visualización son tipos especiales de objetos de visualización que, además de tener su propia representación visual, pueden contener objetos secundarios que también sean objetos de visualización.

  La clase DisplayObjectContainer es una subclase de la clase DisplayObject. Un objeto DisplayObjectContainer puede contener varios objetos de visualización en su *lista deelementos secundarios*. Por ejemplo, la siguiente ilustración muestra un tipo de objeto DisplayObjectContainer, denominado Sprite, que contiene diversos objetos de visualización:



A. *Un objeto SimpleButton. Este tipo de objeto de visualización tiene distintos estados "Arriba", "Abajo" y "Sobre".* ***B.*** *Un objeto Bitmap. En este caso, el objeto Bitmap se cargó de un archivo JPEG externo a través de un objeto Loader.* ***C.*** *Un objeto Shape. El "marco del cuadro" contiene un rectángulo redondeado que se dibuja en ActionScript. Este objeto Shape tiene aplicado un filtro de sombra.* ***D.*** *Un objeto TextField.*

  Cuando se habla de los objetos de visualización, también se hace referencia a los objetos DisplayObjectContainer como *contenedores de objetos de visualización* o simplemente como *contenedores*. Como se ha comentado previamente, el objeto Stage es un contenedor de objeto de visualización.

  Aunque todos los objetos de visualización visibles heredan de la clase DisplayObject, el tipo de cada uno de ellos pertenece a una subclase específica de la clase DisplayObject. Por ejemplo, hay una función constructora para la clase Shape o la clase Video, pero no hay ninguna función constructora para la clase DisplayObject.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecen al programar gráficos de ActionScript:

Alfa Valor del color que representa el grado de transparencia (en realidad, el grado de opacidad) de un color. Por ejemplo, un color con un valor de canal alfa de 60% solo muestra el 60% de su intensidad total y es un 40% transparente.

Gráfico de mapa de bits Gráfico que se define en el equipo como una cuadrícula (filas y columnas) de píxeles de colores. Los gráficos de mapa de bits suelen incluir fotos e imágenes similares.

Modo de fusión Especificación de cómo debe interactuar el contenido de dos imágenes solapadas. Normalmente, una imagen opaca sobre otra imagen simplemente bloquea la imagen de debajo de forma que no esté visible; no obstante, los distintos modos de mezcla hacen que los colores de las imágenes se mezclen de maneras diferentes, por lo que el contenido resultante es una combinación de las dos imágenes.

Lista de visualización Jerarquía de objetos de visualización que Flash Player y AIR representarán como contenido visible en pantalla. El escenario es la raíz de la lista de visualización y todos los objetos de visualización asociados al escenario o a uno de sus elementos secundarios constituyen la lista de visualización, aunque el objeto no se represente realmente (por ejemplo, si está fuera de los límites del escenario).

Objeto de visualización Objeto que representa algún tipo de contenido visual en Flash Player o AIR. Solo se pueden incluir objetos de visualización en la lista de visualización y todas las clases de objetos de visualización son subclases de la clase DisplayObject.

Contenedor de objetos de visualización Tipo especial de objeto de visualización que puede contener objetos de visualización secundarios además de tener (generalmente) su propia representación visual.

Clase principal del archivo SWF Clase que define el comportamiento del objeto de visualización más exterior en un archivo SWF, que conceptualmente es la clase para el mismo archivo SWF. Por ejemplo, en un archivo SWF creado en edición de Flash, la clase principal es la clase de documento. Tiene una "línea de tiempo principal" que contiene todas las demás líneas de tiempo; la clase principal del archivo SWF es la clase de la que la línea de tiempo principal es una instancia.

Enmascaramiento Técnica para ocultar determinadas partes de una imagen (o a la inversa, para permitir únicamente la visualización de partes determinadas de una imagen). Las partes ocultas de la imagen pasan a ser transparentes, por lo que se puede ver el contenido de debajo. El término en inglés ("masking") está relacionado con la cinta de pintor ("masking tape") que se aplica para evitar pintar donde no hay que pintar.

Escenario Contenedor visual que constituye la base o el fondo de todo el contenido visual de un archivo SWF.

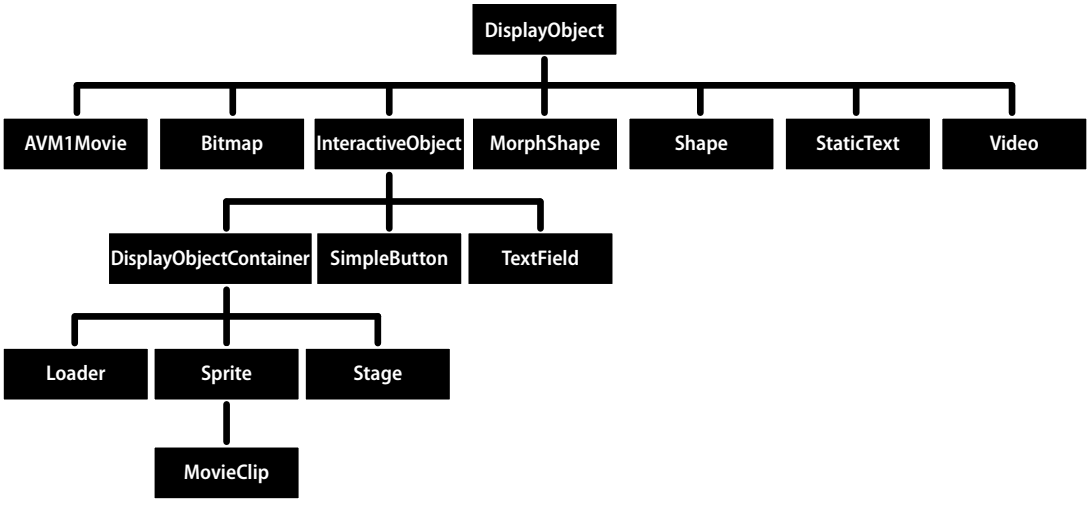
Transformación Ajuste de una característica visual de un gráfico, como girar el objeto, modificar su escala, sesgar o distorsionar su forma, o bien, modificar su color.

Gráfico vectorial Gráfico definido en el equipo como líneas y formas dibujadas con características específicas (como grosor, longitud, tamaño, ángulo y posición).

Clases principales de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El paquete flash.display de ActionScript 3.0 contiene clases de objetos visuales que pueden aparecer en Flash Player o AIR. En la siguiente ilustración se muestran las relaciones entre las subclases de estas clases principales de objetos de visualización.



En la ilustración se muestra la herencia de clases de las clases principales de visualización. Debe tenerse en cuenta que algunas de estas clases, en concreto, StaticText, TextField y Video, no se encuentran en el paquete flash.display pero también heredan de la clase DisplayObject.

Todas las clases que amplían la clase DisplayObject heredan sus métodos y propiedades. Para obtener más información, consulte "Propiedades y métodos de la clase DisplayObject" en la página 163.

Pueden crearse instancias de objetos de las siguientes clases contenidas en el paquete flash.display:

- Bitmap: la clase Bitmap se usa para definir objetos de mapa de bits cargados de archivos externos o representados a través de ActionScript. Para cargar mapas de bits de archivos externos, se puede usar la clase Loader. Se pueden cargar archivos GIF, JPG o PNG. También es posible crear un objeto BitmapData con datos personalizados y crear a continuación un objeto Bitmap que utilice dichos datos. Se pueden usar los métodos de la clase BitmapData para modificar mapas de bits, tanto si se cargan como si se crean en ActionScript. Para obtener más información, consulte "Carga de objetos de visualización" en la página 205 y "Trabajo con mapas de bits" en la página 251.
- Loader: la clase Loader se usa para cargar activos externos (archivos SWF o gráficos). Para obtener más información, consulte "Carga dinámica de contenido de visualización" en la página 204.
- Shape: la clase Shape se usa para crear gráficos vectoriales como rectángulos, líneas, círculos, etc. Para obtener más información, consulte "Uso de la API de dibujo" en la página 229.
- SimpleButton. Un objeto SimpleButton es la representación de ActionScript de un símbolo de botón creado en la herramienta de edición de Flash. Una instancia de SimpleButton dispone de cuatro estados de botón: up (arriba), down (abajo), over (sobre) y hit test (prueba de zona activa) (el área que responde a eventos de teclado y ratón).
- Sprite: un objeto Sprite puede contener gráficos propios y también objetos de visualización secundarios. La clase Sprite amplía la clase DisplayObjectContainer. Para obtener más información, consulte "Trabajo con contenedores de objetos de visualización" en la página 163 y "Uso de la API de dibujo" en la página 229.

- MovieClip: un objeto MovieClip es la representación en ActionScript de un símbolo de clip de película creado en la herramienta de edición de Flash. En la práctica, un MovieClip es similar a un objeto Sprite, excepto en que tiene además una línea de tiempo. Para obtener más información, consulte "Trabajo con clips de película" en la página 337.

Las siguientes clases, que no se encuentran en el paquete flash.display, son subclases de la clase DisplayObject:

- La clase TextField, incluida en el paquete flash.text, es un objeto de visualización para mostrar e introducir texto. Para obtener más información, consulte "Fundamentos de la utilización de texto" en la página 389.
- La clase TextLine, incluida en el paquete flash.text.engine, es el objeto de visualización utilizado para mostrar líneas de texto compuestas por Flash Text Engine y Text Layout Framework. Para obtener más información, consulte "Uso de Flash Text Engine" en la página 416 y "Uso de Text Layout Framework" en la página 446.
- La clase Video, incluida en el paquete flash.media, es el objeto de visualización que se utiliza para mostrar archivos de vídeo. Para obtener más información, consulte "Trabajo con vídeo" en la página 494.

Las siguientes clases del paquete flash.display amplían la clase DisplayObject, pero no es posible crear instancias de las mismas. En lugar de eso, actúan como clases principales de otros objetos de visualización y combinan la funcionalidad común en una sola clase.

- AVM1Movie: la clase AVM1Movie se usa para representar los archivos SWF cargados que se crearon en ActionScript 1.0 y 2.0.
- DisplayObjectContainer: las clases Loader, Stage, Sprite y MovieClip amplían la clase DisplayObjectContainer. Para obtener más información, consulte "Trabajo con contenedores de objetos de visualización" en la página 163.
- InteractiveObject: InteractiveObject es la clase base de todos los objetos y se utiliza para interactuar con el ratón y el teclado. Los objetos SimpleButton, TextField, Loader, Sprite, Stage y MovieClip son subclases de la clase InteractiveObject. Para obtener más información sobre la creación de interacción de teclado y ratón, consulte "Aspectos básicos de la interacción con el usuario" en la página 580.
- MorphShape: estos objetos se crean al crear una interpolación de forma en la herramienta de edición de Flash. No es posible crear instancias de estos objetos con ActionScript pero se puede acceder a ellos desde la lista de visualización.
- Stage: la clase Stage amplía la clase DisplayObjectContainer. Hay una instancia de Stage por aplicación y se sitúa en lo más alto de la jerarquía de la lista de visualización. Para acceder a Stage, debe usarse la propiedad `stage` de cualquier instancia de DisplayObject. Para obtener más información, consulte "Configuración de las propiedades de Stage" en la página 169.

Además, la clase StaticText del paquete flash.text amplía la clase DisplayObject, pero no es posible crear una instancia de ella en el código. Los campos de texto estático se crean únicamente en Flash.

Las siguientes clases no son objetos de visualización ni contenedores de objetos de visualización y tampoco aparecen en la lista de visualización, pero muestran gráficos en el escenario. Estas clases se dibujan en un rectángulo, denominado ventana gráfica, y se sitúan con relación al escenario.

- StageVideo: muestra el contenido de vídeo, utilizado la aceleración de hardware, si es posible. Esta clase se encuentra disponible a partir de Flash Player 10.2 . Para obtener más información, consulte "Uso de la clase StageVideo para la presentación con aceleración por hardware" en la página 534.
- StageWebView: esta clase muestra contenido HTML. Esta clase se encuentra disponible en AIR 2.5. Para obtener más información, consulte "Objetos StageWebView" en la página 1068.

Las siguientes clases fl.display proporcionan una funcionalidad similar a la de las clases flash.display.Loader y LoaderInfo. Utilice estas clases en lugar de sus equivalentes flash.display si está realizando el desarrollo en el entorno de Flash Professional (CS5.5 o posterior). En dicho entorno, esas clases ayudan a resolver problemas que implican TLF con la precarga de RSL. Para obtener más información, consulte "Uso de las clases ProLoader y ProLoaderInfo" en la página 208.

- fl.display.ProLoader: análoga a flash.display.Loader
- fl.display.ProLoaderInfo: análoga a flash.display.LoaderInfo

Ventajas de la utilización de la lista de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 3.0, hay clases independientes para los distintos tipos de objetos de visualización. En ActionScript 1.0 y 2.0, muchos de los objetos del mismo tipo se incluyen en una clase: MovieClip.

Esta individualización de clases y la estructura jerárquica de las listas de visualización presentan las siguientes ventajas:

- Mayor eficacia de representación y disminución del uso de memoria
- Mejor administración de profundidad
- Recorrido completo de la lista de visualización
- Objetos de visualización fuera de la lista
- Creación más sencilla de subclases de objetos de visualización

Mayor eficacia de representación y tamaños de archivo más pequeños

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 1.0 y 2.0, solo era posible dibujar formas en un objeto MovieClip. En ActionScript 3.0, hay clases de objetos de visualización más sencillas en las que se pueden dibujar objetos. Dado que estas clases de objetos de visualización de ActionScript 3.0 no incluyen el conjunto completo de métodos y propiedades que contiene un objeto MovieClip, requieren menos recursos de memoria y de procesador.

Por ejemplo, cada objeto MovieClip incluye propiedades de la línea de tiempo del clip de película, pero un objeto Shape no. Las propiedades para administrar la línea de tiempo pueden consumir muchos recursos de memoria y de procesador. En ActionScript 3.0, el uso del objeto Shape permite mejorar el rendimiento. El objeto Shape tiene una sobrecarga menor que el objeto MovieClip, que es más complejo. Flash Player y AIR no necesitan administrar las propiedades de MovieClip que no se usan, lo cual mejora la velocidad y disminuye la memoria que utiliza el objeto.

Mejor administración de profundidad

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 1.0 y 2.0, la profundidad se administraba a través de un esquema lineal de administración de profundidad y de métodos como `getNextHighestDepth()`.

ActionScript 3.0 incluye la clase DisplayObjectContainer, que tiene más métodos y propiedades útiles para administrar la profundidad de los objetos de visualización.

En ActionScript 3.0, al mover un objeto de visualización a una nueva posición en la lista de elementos secundarios de una instancia de DisplayObjectContainer, los demás elementos secundarios del contenedor de objeto de visualización se vuelven a colocar automáticamente y reciben la asignación de las posiciones adecuadas en el índice de elementos secundarios del contenedor de objeto de visualización.

Además, en ActionScript 3.0 siempre es posible descubrir todos los objetos secundarios de cualquier contenedor de objeto de visualización. Cada instancia de DisplayObjectContainer tiene una propiedad `numChildren`, que muestra el número de elementos secundarios en el contenedor de objeto de visualización. Y como la lista de elementos secundarios de un contenedor de objeto de visualización es siempre una lista con índice, se puede examinar cada objeto de la lista desde la posición 0 hasta la última posición del índice (`numChildren - 1`). Esto no era posible con los métodos y propiedades de un objeto MovieClip en ActionScript 1.0 y 2.0.

En ActionScript 3.0, se puede recorrer fácilmente la lista de visualización de forma secuencial; no hay huecos en los números de índice de una lista de elementos secundarios de un contenedor de objeto de visualización. Recorrer la lista de visualización y administrar la profundidad de los objetos es mucho más sencillo que lo era en ActionScript 1.0 y 2.0. En ActionScript 1.0 y 2.0, un clip de película podía incluir objetos con huecos intermitentes en el orden de profundidad, lo que podía dificultar el recorrido de la lista de objetos. En ActionScript 3.0, cada lista de elementos secundarios de un contenedor de objeto de visualización se almacena en caché internamente como un conjunto, lo que permite realizar búsquedas rápidamente (por índice). Reproducir indefinidamente todos los elementos secundarios de un contenedor de objeto de visualización es también muy rápido.

En ActionScript 3.0, también se puede acceder a los elementos secundarios de un contenedor de objeto de visualización a través del método `getChildByName()` de la clase DisplayObjectContainer.

Recorrido completo de la lista de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 1.0 y 2.0, no era posible acceder a algunos objetos que se dibujaban en la herramienta de edición de Flash como, por ejemplo, las formas vectoriales. En ActionScript 3.0, es posible acceder a todos los objetos de la lista de visualización creados con ActionScript o con la herramienta de edición de Flash. Para obtener información, consulte "Recorrido de la lista de visualización" en la página 167.

Objetos de visualización fuera de la lista

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 3.0, se pueden crear objetos de visualización que no estén incluidos en la lista de visualización visible. Se conocen como objetos de visualización *fuera de la lista.* Un objeto de visualización solo se añade a la lista de visualización visible cuando se llama al método `addChild()` o `addChildAt()` de una instancia de DisplayObjectContainer que ya se haya añadido a la lista de visualización.

Se pueden utilizar los objetos de visualización fuera de la lista para ensamblar objetos de visualización complejos, como los que tienen contenedores de objetos de visualización con varios objetos de visualización. Mantener los objetos de visualización fuera de la lista permite ensamblar objetos complicados sin tener que usar el tiempo de procesamiento para representar estos objetos de visualización. Cuando se necesita un objeto de visualización fuera de la lista, se puede añadir a la lista de visualización. Además, se puede mover un elemento secundario de un contenedor de objeto de visualización dentro y fuera de la lista de visualización, y a cualquier posición que se elija en la lista de visualización.

Creación más sencilla de subclases de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 1.0 y 2.0, a menudo se añadían objetos MovieClip nuevos a un archivo SWF para crear formas básicas o para mostrar mapas de bits. En ActionScript 3.0, la clase DisplayObject incluye numerosas subclases incorporadas, como Shape y Bitmap. Como las clases de ActionScript 3.0 están más especializadas para determinados tipos de objetos, resulta más sencillo crear subclases básicas de las clases incorporadas.

Por ejemplo, para dibujar un círculo en ActionScript 2.0, se podía crear una clase CustomCircle que ampliara la clase MovieClip al crear una instancia de un objeto de la clase personalizada. Sin embargo, esa clase incluiría además varias propiedades y métodos de la clase MovieClip (por ejemplo, `totalFrames`) que no se aplican a la clase. Sin embargo, en ActionScript 3.0 es posible crear una clase CustomCircle que amplíe el objeto Shape y que, como tal, no incluya las propiedades y métodos no relacionados contenidos en la clase MovieClip. El código siguiente muestra un ejemplo de una clase CustomCircle:

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                 yInput:Number,
                                 rInput:Number,
                                 colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

Trabajo con objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una vez comprendidos los conceptos básicos del escenario, los objetos de visualización, los contenedores de objetos de visualización y la lista de visualización, esta sección proporciona información más específica relativa a la utilización de los objetos de visualización en ActionScript 3.0.

Propiedades y métodos de la clase DisplayObject

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todos los objetos de visualización son subclases de la clase DisplayObject y, como tales, heredan las propiedades y métodos de la clase DisplayObject. Las propiedades heredadas son propiedades básicas que se aplican a todos los objetos de visualización. Por ejemplo, cada objeto de visualización tiene una propiedad `x` y una propiedad `y` que especifican la posición del objeto en su contenedor de objeto de visualización.

No se puede crear una instancia de DisplayObject con el constructor de la clase DisplayObject. Se debe crear otro tipo de objeto (un objeto que sea una subclase de la clase DisplayObject), como Sprite, para crear una instancia de un objeto con el operador `new`. Además, si se desea crear una clase personalizada de un objeto de visualización, se debe crear una subclase de una de las subclases del objeto de visualización que tenga una función constructora que pueda utilizarse (por ejemplo, la clase Shape o la clase Sprite). Para obtener más información, consulte la descripción de la clase DisplayObject en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Añadir objetos de visualización a la lista de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se crea una instancia de un objeto de visualización, no aparecerá en pantalla (en el escenario) hasta que se añada la instancia del objeto de visualización a un contenedor de objeto de visualización de la lista de visualización. Por ejemplo, en el código siguiente, el objeto TextField `myText` no sería visible si se omitiera la última línea de código. En la última línea de código, la palabra clave `this` debe hacer referencia a un contenedor de objeto de visualización que ya se haya añadido a la lista de visualización.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

Cuando se añade un elemento visual al escenario, dicho elemento se convierte en un *elemento secundario* del objeto Stage. El primer archivo SWF cargado en una aplicación (por ejemplo, el que se incorpora en una página HTML) se añade automáticamente como elemento secundario del escenario. Puede ser un objeto de cualquier tipo que amplíe la clase Sprite.

Los objetos de visualización que se crean *sin* usar ActionScript (por ejemplo, añadiendo una etiqueta MXML en un archivo MXML de Flex o colocando un elemento en el escenario de Flash Professional) se añaden a la lista de visualización. Aunque estos objetos de visualización no se añadan mediante ActionScript, sí se puede acceder a ellos a través de ActionScript. Por ejemplo, el código siguiente ajusta la anchura de un objeto denominado `button1`, que se añadió en la herramienta de edición (no a través de ActionScript):

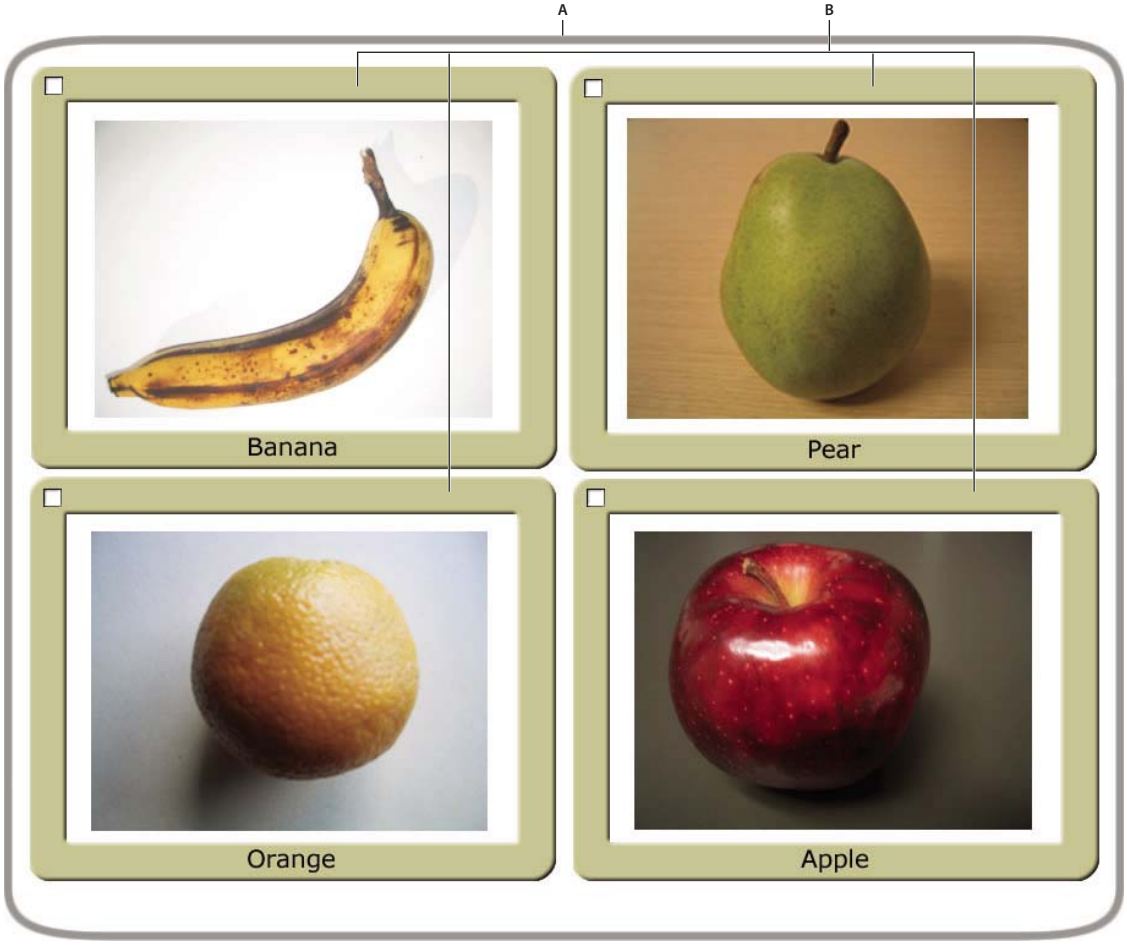
```
button1.width = 200;
```

Trabajo con contenedores de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se elimina un objeto DisplayObjectContainer de la lista de visualización o si se mueve o transforma de algún otro modo, también se elimina, mueve o transforma cada objeto de visualización de DisplayObjectContainer.

Un contenedor de objeto de visualización es por sí mismo un tipo de objeto de visualización; puede añadirse a otro contenedor de objeto de visualización. Por ejemplo, en la imagen siguiente se muestra un contenedor de objeto de visualización, `pictureScreen`, que contiene una forma de contorno y otros cuatro contenedores de objetos de visualización (del tipo PictureFrame):



A. *Una forma que define el borde del contenedor de objeto de visualización pictureScreen* ***B.*** *Cuatro contenedores de objetos de visualización que son elementos secundarios del objeto pictureScreen*

Para que un objeto de visualización aparezca en la lista de visualización, debe añadirse a un contenedor de objeto de visualización incluido en la lista de visualización. Para ello, se utiliza el método `addChild()` o el método `addChildAt()` del objeto contenedor. Por ejemplo, sin la línea final del código siguiente, no se mostraría el objeto `myTextField`:

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

En este ejemplo de código, `this.root` señala al contenedor de objeto de visualización MovieClip que contiene el código. En el código real, se puede especificar un contenedor distinto.

Se debe usar el método `addChildAt()` para añadir el elemento secundario a una posición específica de la lista secundaria del contenedor de objeto de visualización. Estas posiciones del índice basado en cero de la lista de elementos secundarios se refieren a la organización en capas (orden de delante a atrás) de los objetos de visualización. Por ejemplo, observe los tres objetos de visualización siguientes. Cada objeto se creó a partir de una clase personalizada denominada Ball.

La organización en capas de estos objetos de visualización en el contenedor puede ajustarse con el método `addChildAt()`. Por ejemplo, considérese el fragmento de código siguiente:

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

Después de ejecutar este código, los objetos de visualización se colocan del siguiente modo en el objeto DisplayObjectContainer `container`. Observe la organización en capas de los objetos.

Para volver a colocar un objeto en la parte superior de la lista de visualización, simplemente hay que volver a añadirlo a la lista. Por ejemplo, después del código anterior, para mover `ball_A` a la parte superior de la pila, debe utilizarse esta línea de código:

```
container.addChild(ball_A);
```

Este código quita `ball_A` de su ubicación en la lista de visualización de `container` y vuelve a añadirlo a la parte superior de la lista, por lo que finalmente se mueve a la parte superior de la pila.

Puede usarse el método `getChildAt()` para verificar el orden de las capas de los objetos de visualización. El método `getChildAt()` devuelve objetos secundarios de un contenedor basándose en el número de índice que se pasa. Por ejemplo, el código siguiente muestra nombres de objetos de visualización en distintas posiciones de la lista de elementos secundarios del objeto DisplayObjectContainer `container`:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Si se quita un objeto de visualización de la lista de elementos secundarios del contenedor principal, los elementos de niveles superiores de la lista descienden una posición en el índice de elementos secundarios. Por ejemplo, siguiendo con el código anterior, el código siguiente muestra cómo el objeto de visualización que estaba en la posición 2 del objeto DisplayObjectContainer `container` se mueve a la posición 1 si se quita un objeto de visualización situado en un nivel inferior de la lista de elementos secundarios:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Los métodos `removeChild()` y `removeChildAt()` no eliminan completamente una instancia de objeto de visualización. Simplemente la quitan de la lista de elementos secundarios del contenedor. Otra variable podrá seguir haciendo referencia a la instancia. (Para eliminar completamente un objeto, debe usarse el operador `delete`.)

Dado que un objeto de visualización solo tiene un contenedor principal, se puede añadir una instancia de un objeto de visualización a un solo contenedor de objeto de visualización. Por ejemplo, el código siguiente muestra que el objeto de visualización `tf1` solo puede existir en un contenedor (en este caso, un objeto Sprite, que amplía la clase DisplayObjectContainer):

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

Si se añade un objeto de visualización de un contenedor de objeto de visualización a otro contenedor de objeto de visualización, se elimina de la lista de elementos secundarios del primer contenedor de objeto de visualización.

Además de los métodos anteriormente descritos, la clase DisplayObjectContainer define varios métodos para trabajar con objetos de visualización secundarios, entre los que se encuentran:

- `contains()`: determina si un objeto de visualización es un elemento secundario de DisplayObjectContainer.
- `getChildByName()`: recupera un objeto de visualización por el nombre.
- `getChildIndex()`: devuelve la posición de índice de un objeto de visualización.
- `setChildIndex()`: cambia la posición de un objeto de visualización secundario.
- `removeChildren()`: elimina varios objetos de visualización secundarios.

- `swapChildren()`: intercambia el orden de delante a atrás de dos objetos de visualización.
- `swapChildrenAt()`: intercambia el orden de delante a atrás de dos objetos de visualización, especificados por sus valores de índice.

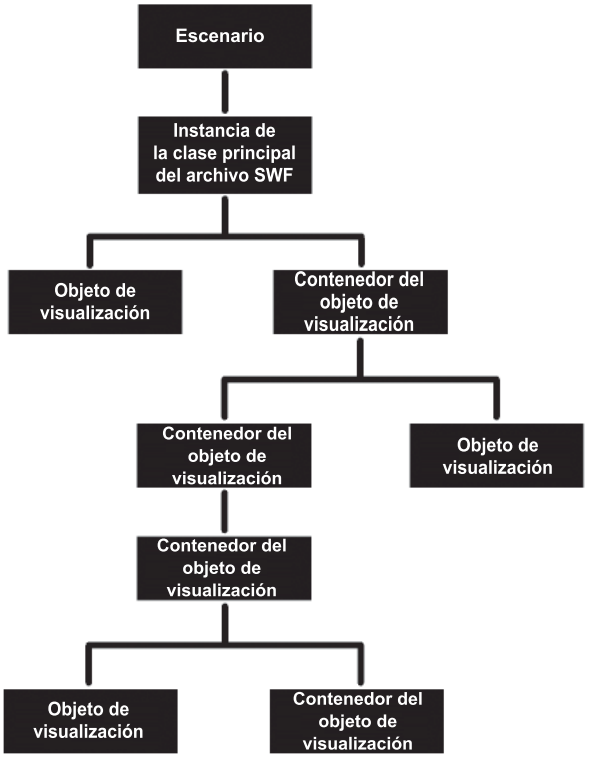
Para obtener más información, consulte las entradas correspondiente en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Cabe recordar que un objeto de visualización que esté fuera de la lista de visualización (es decir, que no esté incluido en un contenedor de objeto de visualización secundario del objeto Stage) se denomina objeto de visualización *fuera de la lista*.

Recorrido de la lista de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Tal como se ha visto, la lista de visualización tiene una estructura de árbol. En la parte superior del árbol está el objeto Stage, que puede contener varios objetos de visualización. Estos objetos de visualización, que son al mismo tiempo contenedores de objetos de visualización, pueden contener otros objetos de visualización o contenedores de objetos de visualización.



La clase DisplayObjectContainer incluye propiedades y métodos para recorrer la lista de visualización, mediante las listas de elementos secundarios de los contenedores de objetos de visualización. Por ejemplo, el código siguiente añade dos objetos de visualización, `title` y `pict`, al objeto `container` (que es un objeto Sprite y la clase Sprite amplía la clase DisplayObjectContainer):

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

El método `getChildAt()` devuelve el elemento secundario de la lista de visualización en una posición de índice específica:

```
trace(container.getChildAt(0) is TextField); // true
```

También se puede acceder a los objetos secundarios por el nombre. Todos los objetos de visualización tienen una propiedad de nombre y, si no se asigna, Flash Player o AIR asignan un valor predeterminado como, por ejemplo, `"instance1"`. Por ejemplo, el código siguiente muestra cómo utilizar el método `getChildByName()` para acceder a un objeto de visualización secundario con el nombre `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

Si se utiliza el método `getChildByName()` el resultado es más lento que si se usa el método `getChildAt()`.

Como un contenedor de objeto de visualización puede contener otros contenedores de objetos de visualización como objetos secundarios en su lista de visualización, se puede recorrer toda la lista de visualización de la aplicación como un árbol. Por ejemplo, en el fragmento de código anterior, cuando finaliza la operación de carga del objeto Loader `pict`, el objeto `pict` tendrá cargado un objeto de visualización secundario, que es el mapa de bits. Para acceder a este objeto de visualización de mapa de bits, se puede escribir `pict.getChildAt(0)`. También se puede escribir `container.getChildAt(0).getChildAt(0)` (porque `container.getChildAt(0) == pict`).

La función siguiente proporciona una salida `trace()` con sangría de la lista de visualización desde un contenedor de objeto de visualización:

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = ""):void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

Adobe Flex

Si utiliza Flex, debe saber que define varias clases del objeto de visualización de componente y estas clases sustituyen los métodos de acceso de la lista de visualización de la clase DisplayObjectContainer. Por ejemplo, la clase Container del paquete mx.core sustituye al método `addChild()` y a otros métodos de la clase DisplayObjectContainer (que amplía la clase Container). En el caso del método `addChild()`, la clase sustituye el método de tal modo que no es posible agregar todos los tipos de objetos de visualización a una instancia de Container en Flex. En este caso, el método sustituido requiere que el objeto secundario que se va a agregar sea un tipo de objeto mx.core.UIComponent.

Configuración de las propiedades de Stage

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Stage sustituye la mayoría de las propiedades y métodos de la clase DisplayObject. Si se llama a una de estas propiedades o métodos sustituidos, Flash Player o AIR emiten una excepción. Por ejemplo, el objeto Stage no tiene propiedades `x` ni `y` porque su posición es fija como el contenedor principal de la aplicación. Las propiedades `x` e `y` hacen referencia a la posición de un objeto de visualización con respecto a su contenedor; como Stage no se incluye en ningún otro contenedor de objeto de visualización, estas propiedades no se aplican.

Nota: *algunas propiedades y métodos de la clase Stage solo están disponibles para los objetos de visualización que se encuentran en el mismo entorno limitado de seguridad que el primer archivo SWF cargado. Para obtener información, consulte "Seguridad del objeto Stage" en la página 1109.*

Control de la velocidad de fotogramas de reproducción

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La propiedad `framerate` de la clase Stage se utiliza para definir la velocidad de fotogramas de todos los archivos SWF cargados en la aplicación. Para obtener más información, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Control del ajuste de escala del escenario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando la parte de la pantalla que representan Flash Player o AIR cambia de tamaño, el motor de ejecución ajusta automáticamente el contenido de Stage para realizar una compensación. La propiedad `scaleMode` de la clase Stage determina cómo se ajustará el contenido del escenario. Esta propiedad se puede establecer en cuatro valores distintos, definidos como constantes en la clase flash.display.StageScaleMode:

- `StageScaleMode.EXACT_FIT` escala el archivo SWF para rellenar las nuevas dimensiones del escenario sin tener en cuenta la relación de aspecto del contenido original. Puede que los factores de escala no sean los mismos para la anchura y la altura, por lo que el contenido puede aparecer comprimido o expandido si cambia la relación de aspecto del escenario.
- `StageScaleMode.SHOW_ALL` escala el archivo SWF para ajustarse por completo a las nuevas dimensiones del escenario sin cambiar la relación de aspecto del contenido. Este modo de escala muestra todo el contenido, pero puede generar bordes de "pantalla ancha", como las barras negras que aparecen al ver una película de pantalla ancha en una televisión estándar.
- `StageScaleMode.NO_BORDER` escala el archivo SWF para ajustarse por completo a las nuevas dimensiones del escenario sin cambiar la relación de aspecto del contenido. Este modo de escala hace un uso completo del área de visualización de escenario, pero puede suponer un recorte.
- `StageScaleMode.NO_SCALE`: no escala el archivo SWF. Si las nuevas dimensiones del escenario son más pequeñas, el contenido se recorta; si son mayores, el espacio añadido está en blanco.

  Solo en el modo de escala `StageScaleMode.NO_SCALE`, las propiedades `Width` y `Height` de la clase Stage, se pueden utilizar para determinar las dimensiones en píxeles reales del escenario cuyo tamaño ha cambiado. (En los otros modos de escala, las propiedades `stageWidth` y `stageHeight` siempre reflejan la anchura y la altura originales del archivo SWF.) Además, si se establece `scaleMode` en `StageScaleMode.NO_SCALE` y se ajusta el tamaño del archivo SWF, se distribuye el evento `resize` de la clase Stage, lo que permite realizar los ajustes necesarios.

Por consiguiente, si establece `scaleMode` en `StageScaleMode.NO_SCALE`, tendrá mayor control sobre el ajuste del contenido de la pantalla al cambiar el tamaño de la ventana. Por ejemplo, en un archivo SWF que contiene un vídeo y una barra de control, puede que desee que la barra de control mantenga el mismo tamaño cuando se cambie el tamaño del escenario y modificar únicamente el tamaño de la ventana de vídeo para adaptarla al cambio de tamaño del escenario. Esto se ilustra en el siguiente ejemplo:

```
// mainContent is a display object containing the main content;
// it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below mainContent) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = mainContent.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;
swfStage.addEventListener(Event.RESIZE, resizeDisplay);

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the main content area
    var newContentHeight:Number = swfHeight - controlBar.height;
    mainContent.height = newContentHeight;
    mainContent.scaleX = mainContent.scaleY;

    // Reposition the control bar.
    controlBar.y = newContentHeight;
}
```

Definición del modo de escala del escenario para las ventanas de AIR

La propiedad `scaleMode` del escenario determina el modo en que el escenario escala y recorta los objetos de visualización secundarios cuando una ventana cambia de tamaño. Únicamente el modo `noScale` se debe utilizar en AIR. En este modo, no se aplica escala al escenario. En cambio, el tamaño del escenario cambia directamente con los límites de la ventana. Los objetos se pueden recortar si se reduce el tamaño de la ventana.

Los modos de escala de escenario se designan para su uso en entornos como, por ejemplo, un navegador web donde no siempre se tiene control sobre la proporción de tamaño o aspecto del escenario. Los modos permiten seleccionar la opción más adecuada cuando el escenario no coincide con la relación de aspecto o tamaño ideal de la aplicación. En AIR, siempre se tiene control del escenario, por lo que en la mayoría de los casos con la nueva distribución del contenido o el ajuste de las dimensiones de la ventana, se obtendrán mejores resultados que activando el ajuste de escala del escenario.

En el navegador y para la ventana de AIR inicial, la relación entre el tamaño de la ventana y el factor de escala inicial se leen desde el archivo SWF cargado. No obstante, cuando se crea un objeto NativeWindow, AIR selecciona una relación arbitraria entre el tamaño de la ventana y el factor de escala de 72:1. De este modo, si la cuenta con 72x72 píxeles, un rectángulo de 10x10 añadido a la ventana se dibuja con el tamaño correcto de 10x10 píxeles. Sin embargo, si la ventana presenta 144x144 píxeles, a un rectángulo de 10x10 píxeles se le aplica una escala de 20x20 píxeles. Si se utiliza `scaleMode` en lugar de `noScale` para un escenario de la ventana, la compensación se puede realizar estableciendo el factor de escala de cualquier objeto de visualización de la ventana en una relación de 72 píxeles en la anchura y altura actual del escenario. Por ejemplo, el siguiente código calcula el factor de escala necesario para un objeto de visualización denominado `client`:

```
if(newWindow.stage.scaleMode != StageScaleMode.NO_SCALE){
client.scaleX = 72/newWindow.stage.stageWidth;
client.scaleY = 72/newWindow.stage.stageHeight;
}
```

Nota: *las ventanas de Flex y HTML establecen automáticamente el valor `scaleMode` del escenario en `noScale`. Al cambiar el valor `scaleMode`, se interrumpen los mecanismos de diseño automático en estos tipos de ventanas.*

Trabajo con el modo de pantalla completa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El modo de pantalla completa permite establecer un escenario de película para rellenar el monitor completo de un espectador sin ningún borde de contenedor o menú. La propiedad `displayState` de la clase Stage se utiliza para activar y desactivar el modo de pantalla completa para un archivo SWF. La propiedad `displayState` puede establecerse en uno de los valores definidos por las constantes de la clase flash.display.StageDisplayState. Para activar el modo de pantalla completa, debe establecerse la propiedad `displayState` en `StageDisplayState.FULL_SCREEN`:

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

Para activar el modo de pantalla completa(novedad en Flash Player 11.3), establezca la propiedad `displayState` en `StageDisplayState.FULL_SCREEN_INTERACTIVE`:

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

En Flash Player, el modo de pantalla completa solo puede iniciarse a través de ActionScript como respuesta a un clic del ratón (incluido el clic con el botón derecho) o una pulsación de tecla. El contenido de AIR que se ejecuta en el entorno limitado de seguridad de la aplicación no requiere que se indique el modo de pantalla completa como respuesta a un gesto del usuario.

Para salir del modo de pantalla completa, debe establecerse la propiedad `displayState` en `StageDisplayState.NORMAL`.

```
stage.displayState = StageDisplayState.NORMAL;
```

Asimismo, el usuario puede optar por salir del modo de pantalla completa cambiando a una ventana diferente o utilizando una de las distintas combinaciones de teclas: tecla Esc (todas las plataformas), Control-W (Windows), Comando-W (Mac) o Alt-F4 (Windows).

Activación del modo de pantalla completa en Flash Player

Para activar el modo de pantalla completa en un archivo SWF incorporado en una página HTML, el código HTML para incorporar Flash Player debe incluir una etiqueta `param` y un atributo `embed` con el nombre `allowFullScreen` y el valor `true`, como en el siguiente código:

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowFullScreen="true" />
</object>
```

En la herramienta de edición de Flash, seleccione Archivo -> Configuración de publicación y en el cuadro de diálogo Configuración de publicación, en la ficha HTML, seleccione la plantilla Solo Flash - Permitir pantalla completa.

En Flex, asegúrese de que la plantilla HTML incluya las etiquetas `<object>` y `<embed>` que son compatibles con la pantalla completa.

Si se utiliza JavaScript en una página web para generar las etiquetas de incorporación de SWF, se debe modificar JavaScript para añadir el atributo/etiqueta de parámetro `allowFullScreen.` Por ejemplo, si la página HTML utiliza la función `AC_FL_RunContent()` (que se usa en páginas HTML generadas en Flash Professional y Flash Builder), se debe añadir el parámetro `allowFullScreen` a dicha llamada de función, del siguiente modo:

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Esto no se aplica a los archivos SWF que se ejecutan en el reproductor Flash Player autónomo.

Nota: *si el modo de ventana (wmode en HTML) se establece en Opaco sin ventanas (opaco) o Transparente sin ventanas (transparente), la ventana de pantalla completa siempre es opaca.*

También existen limitaciones relacionadas con la seguridad al utilizar el modo de pantalla completa de Flash Player en un navegador. Estas limitaciones se describen en el capítulo "Seguridad" en la página 1085.

Activación del modo de pantalla completa en Flash Player 11.3 y versiones posteriores

Flash Player 11.3 y versiones posteriores admite el modo de pantalla completa, que acepta todas las teclas del teclado (salvo **Esc**, que sale del modo de pantalla completa). El modo interactivo de pantalla completa resulta útil para jugar (por ejemplo, para permitir juegos con varios jugadores o controles de teclado WASD en jugadores que disparan en primera persona).

Para activar el modo de pantalla completa en un archivo SWF incorporado en una página HTML, el código HTML para incorporar Flash Player debe incluir una etiqueta `param` y un atributo `embed` con el nombre `allowFullScreenInteractive` y el valor `true`, como el siguiente código:

```
<object>
    ...
    <param name="allowFullScreenInteractive" value="true" />
    <embed ... allowFullScreenInteractive="true" />
</object>
```

En la herramienta de edición de Flash, seleccione Archivo -> Configuración de publicación y en el cuadro de diálogo Configuración de publicación, en la ficha HTML, seleccione la plantilla Solo Flash - Permitir pantalla completa.

En Flash Builder y en Flex, asegúrese de que las plantillas HTML incluyen etiquetas `<object>` y `<embed>` que admitan el modo de pantalla completa.

Si se utiliza JavaScript en una página web para generar las etiquetas de incorporación de SWF, se debe modificar el JavaScript para añadir el atributo/etiqueta `allowFullScreenInteractive param`. Por ejemplo, si la página HTML utiliza la función `AC_FL_RunContent()` (que se usa en páginas HTML generadas en Flash Professional y Flash Builder), se debe añadir el parámetro `allowFullScreenInteractive` a dicha llamada de función, del siguiente modo:

```
AC_FL_RunContent(
    ...
    'allowFullScreenInteractive','true',
    ...
    ); //end AC code
```

Esto no se aplica a los archivos SWF que se ejecutan en el reproductor Flash Player autónomo.

Escala y tamaño del escenario en pantalla completa

Las propiedades `Stage.fullScreenHeight` y `Stage.fullScreenWidth` devuelven la altura y la anchura del monitor que se utiliza cuando se pasa al tamaño de pantalla completa, si ese estado se introduce de forma inmediata. Estos valores pueden ser incorrectos si el usuario tiene la oportunidad de mover el navegador de un monitor a otro después de recuperar estos valores pero antes de pasar al modo de pantalla completa. Si se recuperan estos valores en el mismo controlador de eventos en el que se estableció la propiedad `Stage.displayState` como `StageDisplayState.FULL_SCREEN`, los valores son correctos. Para usuarios de varios monitores, el contenido de SWF solo se expandirá para llenar un monitor. Flash Player y AIR usan una métrica para determinar el monitor que contiene la mayor parte del archivo SWF y utiliza dicho monitor en el modo de pantalla completa. Las propiedades fullScreenHeight y fullScreenWidth solo reflejan el tamaño del monitor que se utiliza para el modo de pantalla completa. Para obtener más información, consulte `Stage.fullScreenHeight` y `Stage.fullScreenWidth` en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

El comportamiento de ajuste de escala del escenario en el modo de pantalla completa es el mismo que en el modo normal; el ajuste de escala se controla con la propiedad `scaleMode` de la clase Stage. Si la propiedad `scaleMode` se establece en `StageScaleMode.NO_SCALE`, las propiedades `stageWidth` y `stageHeight` de Stage cambian para reflejar el tamaño del área de la pantalla ocupada por el archivo SWF (toda la pantalla en este caso); si se visualiza en el navegador, el parámetro HTML controla la configuración.

Se puede utilizar el evento `fullScreen` de la clase Stage para detectar cuándo se activa y desactiva el modo de pantalla completa, y para responder ante ello. Por ejemplo, puede ser que se desee volver a colocar, añadir o quitar elementos de la pantalla al activar o desactivar el modo de pantalla completa, como en el ejemplo:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

Tal y como se muestra en este código, el objeto del evento `fullScreen` es una instancia de la clase flash.events.FullScreenEvent, que incluye una propiedad `fullScreen` que indica si el modo de pantalla completa está activado (`true`) o no (`false`).

Compatibilidad con el teclado en modo de pantalla completa

Cuando Flash Player se ejecuta en un navegador, todo el código ActionScript relacionado con el teclado, como los eventos de teclado y la introducción de texto en instancias de TextField, se desactiva en modo de pantalla completa. Las excepciones (teclas que permanecen activadas) son:

- Determinadas teclas que no afectan a la impresión, como las teclas de flecha, la barra espaciadora o el tabulador
- Métodos abreviados de teclado que cierran el modo de pantalla completa: Esc (Windows y Mac), Control-W (Windows), Comando-W (Mac) y Alt-F4

Estas restricciones no se aplican a contenido de SWF ejecutado en el reproductor autónomo de Flash Player o en AIR. AIR admite un modo interactivo de pantalla completa que permite la entrada del teclado.

Compatibilidad con el ratón en modo de pantalla completa

De forma predeterminada, los eventos de ratón en modo de pantalla completa funcionan del mismo modo que cuando no se está en pantalla completa. Sin embargo, en modo de pantalla completa, si se desea es posible establecer la propiedad `Stage.mouseLock` para activar el bloqueo del ratón. El bloqueo del ratón desactiva el cursor y permite el movimiento ilimitado del ratón.

Nota: *solo es posible activar el bloqueo del ratón en el modo de pantalla completa en las aplicaciones para escritorio. Si se activa cuando no se está en modo de pantalla completa, o en aplicaciones para dispositivos móviles, se emitirá una excepción.*

El bloqueo del ratón se desactiva automáticamente y el cursor del ratón se hace de nuevo visible cuando:

- El usuario sale del modo de pantalla completa mediante el uso de la tecla Escape (todas las plataformas), Control-W (Windows), Comando-W (Mac) o Alt-F4 (Windows).
- La ventana de la aplicación deja de estar seleccionada.
- Cualquier interfaz de configuración es visible, incluidos todos los cuadros de diálogo de privacidad.
- Se visualiza un cuadro de diálogo nativo, por ejemplo, un cuadro de diálogo carga de archivos.

Los eventos asociados al movimiento del ratón, como el evento `mouseMove`, utilizan la clase MouseEvent para representar al objeto de evento. Si está desactivado el bloqueo del ratón, utilice las propiedades `MouseEvent.localX` y `MouseEvent.localY` para determinar la ubicación del ratón. Si está activado el bloqueo del ratón, utilice las propiedades `MouseEvent.movementX` y `MouseEvent.movementY` para determinar la ubicación del ratón. Las propiedades `movementX` y `movementY` contienen cambios de posición del ratón desde el último evento, no las coordenadas absolutas de la ubicación del ratón.

Escala de hardware en modo de pantalla completa

La propiedad `fullScreenSourceRect` de la clase Stage se puede utilizar para establecer que Flash Player o AIR escalen una región específica del escenario al modo de pantalla completa. La escala de hardware en Flash Player y AIR, en caso de estar disponible, se activa en la tarjeta gráfica y de vídeo del equipo y suele mostrar el contenido con mayor rapidez que la escala de software.

Para aprovechar la escala de hardware, debe establecer todo el escenario o parte del mismo en modo de pantalla completa. El siguiente código ActionScript 3.0 establece todo el escenario en modo de pantalla completa:

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Cuando esta propiedad se establece en un rectángulo válido y la propiedad `displayState` se define en modo de pantalla completa, Flash Player y AIR aplican escala al área especificada. El tamaño real del escenario expresado en píxeles dentro de ActionScript no cambia. Flash Player y AIR aplican un límite mínimo para el tamaño del rectángulo de modo que se ajuste al mensaje estándar "Press Esc to exit full-screen mode" (Presione Esc para salir del modo de pantalla completa). Este límite suele ser de 260 x 30 píxeles, pero puede variar según la plataforma y la versión de Flash Player.

La propiedad `fullScreenSourceRect` solo se puede definir cuando Flash Player o AIR no estén en modo de pantalla completa. Para utilizar esta propiedad correctamente, primero debe establecerse y, seguidamente, definir la propiedad `displayState` en modo de pantalla completa.

Para activar la escala, establezca la propiedad `fullScreenSourceRect` en un objeto rectangle.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

Para desactivar la escala, establezca la propiedad `fullScreenSourceRect` en `null`.

```
stage.fullScreenSourceRect = null;
```

Para aprovechar íntegramente las funciones de aceleración de hardware con Flash Player, actívela mediante el cuadro de diálogo Configuración de Flash Player. Para cargar el cuadro de diálogo, haga clic con el botón derecho del ratón (Windows) o presione Control y haga clic (Mac) dentro del contenido de Flash Player en el navegador. Seleccione la ficha Visualización, que es la primera ficha, y active la casilla de verificación: Activar aceleración de hardware.

Modos de ventana directo y de composición con GPU

Flash Player 10 introduce dos modos de ventana, directo y composición por GPU, que se pueden habilitar desde la configuración de publicación de la herramienta de edición de Flash. Estos modos no se admiten en AIR. Para poder utilizar estos modos, es preciso activar la aceleración de hardware para Flash Player.

El modo directo utiliza la ruta más rápida y directa para introducir gráficos en pantalla, lo que resulta apropiado para la reproducción de vídeo.

La composición con GPU utiliza la unidad de procesamiento de gráficos de la tarjeta de vídeo para acelerar la composición. La composición de vídeo es el proceso de organización en capas de varias imágenes para crear una sola imagen de vídeo. Cuando la composición se acelera con la GPU, es posible mejorar el rendimiento de la conversión YUV, la corrección de color, la rotación o la escala, así como la fusión. La conversión YUV hace referencia a la conversión de color de las señales analógicas del compuesto, que se utilizan para la transmisión, en el modelo de color RGB (rojo, verde, azul) que utilizan las pantallas y las cámaras de vídeo. El uso de la GPU para acelerar la composición reduce la demanda del equipo y la memoria que se aplica a la CPU. También supone una reproducción más suavizada para el vídeo de definición estándar.

Sea cuidadoso al implementar estos modos de ventana. El uso de la composición con GPU puede resultar costoso en términos de recursos de CPU y memoria. Si algunas operaciones (por ejemplo, modos de fusión, recorte o enmascaramiento) no se pueden llevar a cabo en la GPU, se realizan mediante el software. Adobe recomienda la limitación a un archivo SWF por página HTML cuando se utilicen estos modos, que no deben se habilitar para banners. La función Probar película de Flash no utiliza la aceleración de hardware, pero puede emplearla mediante la opción Vista previa de publicación.

La definición de una velocidad de fotogramas en el archivo SWF que sea superior a 60, velocidad máxima de actualización de pantalla, no resulta útil. El establecimiento de la velocidad de fotogramas de 50 a 55 permite fotogramas eliminados, lo cual puede suceder por distintos motivos en algunas ocasiones.

Para poder utilizar el modo directo, es preciso disponer de Microsoft DirectX 9 con VRAM de 128 MB en Windows y OpenGL para Apple Macintosh, Mac OS X v10.2 o versiones posteriores. El modo de composición por GPU requiere compatibilidad con Microsoft DirectX 9 y Pixel Shader 2.0 en Windows con 128 MB de VRAM. En Mac OS X y Linux, la composición por GPU requiere OpenGL 1.5 y diversas extensiones de OpenGL (objeto framebuffer, objetos multitexture y shader, lenguaje de sombreado y sombreados de fragmentos).

Puede activar los modos de aceleración `directo` y `gpu` por archivo SWF mediante el cuadro de diálogo de configuración de publicación de Flash, utilizando el menú de aceleración de hardware de la ficha Flash. Si selecciona Ninguno, el modo de ventana vuelve a los valores `predeterminado`, `transparente` u `opaco`, tal y como se especifica en la configuración del modo de ventana en la ficha HTML.

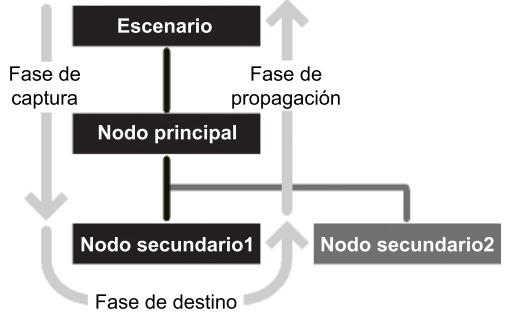
Gestión de eventos de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase DisplayObject hereda de la clase EventDispatcher. Esto significa que cada uno de los objetos de visualización puede participar completamente en el modelo de eventos (descrito en "Gestión de eventos" en la página 127). Cada objeto de visualización puede utilizar su método `addEventListener()`, heredado de la clase EventDispatcher, para detectar un evento concreto, pero solamente si el objeto detector forma parte del flujo de dicho evento.

Cuando Flash Player o AIR distribuyen un objeto de evento, este realiza un viaje de ida y vuelta del escenario al objeto de visualización donde se produjo el evento. Por ejemplo, si un usuario hace clic en un objeto de visualización denominado `child1`, Flash Player distribuye un objeto de evento desde el escenario, y a través de la jerarquía de la lista de visualización, hasta el objeto de visualización `child1`.

El flujo del evento se divide conceptualmente en tres fases, tal y como se ilustra en el diagrama siguiente:



Para obtener más información, consulte "Gestión de eventos" en la página 127.

Un aspecto importante que hay que tener en cuenta cuando se trabaja con eventos de objetos de visualización es el efecto que los detectores de eventos pueden tener sobre la eliminación automática de objetos de visualización de la memoria (recopilados como datos innecesarios) cuando se quitan de la lista de visualización. Si un objeto de visualización tiene objetos suscritos a sus eventos como detectores, dicho objeto de visualización no se eliminará de la memoria aunque se quite de la lista de visualización, porque seguirá teniendo referencias a esos objetos detectores. Para obtener más información, consulte "Administración de detectores de eventos" en la página 141.

Selección de una subclase DisplayObject

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las decisiones importantes que hay que tomar cuando se trabaja con objetos de visualización es elegir qué objeto de visualización se va a utilizar para cada propósito. A continuación se ofrecen algunas directrices que pueden ser útiles en la toma de esta decisión. Estas mismas sugerencias se aplican tanto si se necesita una instancia de una clase como si se elige una clase base para una clase que se esté creando:

- Si no se necesita un objeto que pueda ser un contenedor de otros objetos de visualización (es decir, que solo se necesita uno que actúe como un elemento de pantalla autónomo), debe elegirse una de estas subclases DisplayObject o InteractiveObject, dependiendo del uso que se le vaya a dar:
  - Bitmap para mostrar una imagen de mapa de bits.
  - TextField para añadir texto.
  - Video para mostrar vídeo.
  - Shape para obtener un "lienzo" donde dibujar contenido en pantalla. En concreto, si se desea crear una instancia para dibujar formas en la pantalla y no se va a utilizar como contenedor de otros objetos de visualización, se obtendrán importantes ventajas de rendimiento si se usa Shape en lugar de Sprite o MovieClip.
  - MorphShape, StaticText o SimpleButton para los elementos creados con la herramienta de edición de Flash. (No es posible crear instancias de estas clases mediante programación, pero sí crear variables con estos tipos de datos para hacer referencia a los elementos creados con la herramienta de edición de Flash.)
- Si se necesita una variable para hacer referencia al escenario principal, debe usarse la clase Stage como su tipo de datos.
- Si se necesita un contenedor para cargar un archivo SWF o un archivo de imagen externo, se debe usar una instancia de Loader. El contenido cargado se añadirá a la lista de visualización como un elemento secundario de la instancia de Loader. Su tipo de datos dependerá de la naturaleza del contenido cargado, como se indica a continuación:
  - Una imagen cargada será una instancia de Bitmap.
  - Un archivo SWF cargado, escrito en ActionScript 3.0, será una instancia de Sprite o MovieClip (o una instancia de una subclase de esas clases, según lo especifique el creador de contenido).
  - Un archivo SWF cargado, escrito en ActionScript 1.0 o ActionScript 2.0, será una instancia de AVM1Movie.
- Si se necesita que un objeto actúe como contenedor de otros objetos de visualización, aunque se vaya a dibujar o no en el objeto de visualización mediante ActionScript, debe elegirse una de las subclases de DisplayObjectContainer:
  - Sprite si el objeto se creará solamente con ActionScript, o como la clase base de un objeto de visualización personalizado que se creará y manipulará exclusivamente con ActionScript.
  - MovieClip si se crea una variable para hacer referencia a un símbolo de clip de película creado en la herramienta de edición de Flash.
- Si se crea una clase que se asociará a un símbolo de clip de película en la biblioteca de Flash, se debe elegir una de las siguientes subclases de DisplayObjectContainer como clase base:
  - MovieClip si el símbolo de clip de película asociado tiene contenido en más de un fotograma.
  - Sprite si el símbolo de clip de película asociado tiene contenido solo en el primer fotograma.

Manipulación de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Independientemente del objeto de visualización que se decida utilizar, hay una serie de manipulaciones comunes a todos los objetos de visualización como elementos que se muestran en la pantalla. Por ejemplo, todos ellos pueden colocarse en la pantalla, moverse hacia adelante o hacia atrás en el orden de apilamiento de los objetos de visualización, cambiar de escala, girarse, etc. Dado que todos los objetos de visualización heredan esta funcionalidad de su clase base común (DisplayObject), esta funcionalidad tiene el mismo comportamiento si se manipula una instancia de TextField, una instancia de Video, una instancia de Shape o cualquier otro objeto de visualización. En las secciones siguientes se detallan algunas de estas manipulaciones comunes a los objetos de visualización.

Cambio de posición

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La manipulación más básica de cualquier objeto de visualización es su colocación en la pantalla. Para definir la posición de un objeto de visualización, deben cambiarse las propiedades `x` e `y` del objeto.

```
myShape.x = 17;
myShape.y = 212;
```

El sistema de colocación de objetos de visualización trata el escenario como un sistema de coordenadas cartesiano (el sistema de cuadrícula habitual, con un eje x horizontal y un eje y vertical). El origen del sistema de coordenadas (la coordenada 0,0 donde convergen los ejes x e y) está situado en la esquina superior izquierda del escenario. Tomando esta coordenada como punto de referencia, los valores x son positivos si están a la derecha y negativos si están a la izquierda, mientras que los valores y son positivos si están abajo y negativos si están arriba (al contrario que en los sistemas gráficos típicos). Por ejemplo, las anteriores líneas de código mueven el objeto `myShape` a la coordenada x 17 (17 píxeles a la derecha del origen) y a la coordenada y 212 (212 píxeles por debajo del origen).

De forma predeterminada, cuando se crea un objeto de visualización con ActionScript, las propiedades `x` e `y` se establecen en 0, colocando el objeto en la esquina superior izquierda de su contenido principal.

Cambio de posición con respecto al escenario

Es importante recordar que las propiedades `x` e `y` siempre hacen referencia a la posición del objeto de visualización con respecto a la coordenada 0,0 de los ejes de su objeto de visualización principal. De este modo, en una instancia de Shape (por ejemplo, un círculo) contenida en una instancia de Sprite, al establecer en 0 las propiedades `x` e `y` del objeto Shape, se colocará un círculo en la esquina superior izquierda del objeto Sprite, que no es necesariamente la esquina superior izquierda del escenario. Para colocar un objeto con respecto a las coordenadas globales del escenario, se puede usar el método `globalToLocal()` de cualquier objeto de visualización para convertir las coordenadas globales (escenario) en coordenadas locales (contenedor de objeto de visualización), como en el ejemplo siguiente:

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

También se puede usar el método `localToGlobal()` de la clase DisplayObject para convertir las coordenadas locales en coordenadas del escenario.

Movimiento de objetos de visualización con el ratón

El usuario puede mover los objetos de visualización con el ratón utilizando dos diferentes técnicas en ActionScript. En cualquier caso, se utilizan dos eventos de ratón: cuando se presiona el botón del ratón el objeto recibe la orden de seguir al cursor del ratón y, cuando se suelta el botón, recibe la orden de dejar de seguirlo.

Nota: *Flash Player 11.3 y versiones posteriores, AIR 3.3 y versiones posteriores: también puede utilizar el evento MouseEvent.RELEASE_OUTSIDE para el caso de usuarios que sueltan el botón fuera de los límites del contenedor Sprite.*

La primera técnica consiste en usar el método `startDrag()` y es más sencilla pero también más limitada. Cuando se presiona el botón del ratón, se llama al método `startDrag()` del objeto de visualización que se va a arrastrar. Cuando se suelta el botón del ratón, se llama el método `stopDrag()`. La clase Sprite define estas dos funciones, por lo que el objeto movido debe ser una clase Sprite o una de sus subclases.

```
// This code creates a mouse drag interaction using the startDrag()
// technique.
// square is a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Esta técnica presenta una limitación realmente importante: con `startDrag()` únicamente se puede arrastrar un elemento cada vez. Si se arrastra un objeto de visualización y se llama al método `startDrag()` en otro objeto de visualización, el primer objeto deja de seguir al ratón inmediatamente. Por ejemplo, si se cambia la función `startDragging()` como se muestra aquí, solo se arrastrará el objeto `circle`, en lugar de la llamada al método `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Como consecuencia de poder arrastrar un solo objeto cada vez cuando se usa `startDrag()`, es posible llamar al método `stopDrag()` en cualquier objeto de visualización y detenerlo sea cual sea el objeto que se esté arrastrando.

Si se necesita arrastrar más de un objeto de visualización o evitar la posibilidad de conflictos porque más de un objeto pueda utilizar `startDrag()`, se recomienda usar la técnica de seguimiento del ratón para crear el efecto de arrastre. Con esta técnica, cuando se presiona el botón del ratón, se suscribe una función al evento `mouseMove` del escenario, como detector. Esta función, que se llama cada vez que se mueve el ratón, hace que el objeto arrastrado salte a las coordenadas x, y del ratón. Cuando se suelta el botón del ratón, se quita la suscripción de la función como detector, de modo que ya no se llama cuando se mueve el ratón y el objeto deja de seguir al cursor del ratón. En el código siguiente se muestra esta técnica:

```
// This code moves display objects using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Además de hacer que un objeto de visualización siga al cursor del ratón, suele ser conveniente mover el objeto arrastrado al frente de la pantalla, de modo que parezca que flota sobre todos los demás objetos. Por ejemplo, supongamos que se dispone de dos objetos, un círculo y un cuadrado, y que ambos se pueden mover con el ratón. Si el círculo está por debajo del cuadrado en la lista de visualización, y se hace clic en el círculo y se arrastra de modo que el cursor quede sobre el cuadrado, el círculo parecerá deslizarse detrás del cuadrado, rompiendo así la ilusión de arrastrar y colocar. En lugar de eso, se puede hacer que el círculo se mueva a la parte superior de la lista de visualización cuando se hace clic en él, de modo que el círculo siempre aparezca encima de cualquier otro contenido.

El siguiente código (adaptado del ejemplo anterior) permite que dos objetos de visualización, un círculo y un cuadrado, se muevan con el ratón. Cuando se presiona el botón del ratón sobre cualquiera de los dos, el elemento en cuestión se mueve a la parte superior de la lista de visualización del escenario, de modo que el elemento arrastrado aparece siempre encima. (El código nuevo o cambiado con respecto a la lista anterior aparece en negrita.)

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Para aumentar este efecto, por ejemplo en un juego en el que se mueven fichas o cartas entre distintas pilas, se podría añadir el objeto arrastrado a la lista de visualización del escenario cuando se selecciona y luego añadirlo a otra lista de visualización (por ejemplo, la pila donde se coloca), cuando se suelta el botón del ratón.

Finalmente, para mejorar el efecto, se podría aplicar un filtro de sombra al objeto de visualización al hacer clic en él (al iniciar el arrastre) y quitar la sombra al soltar el objeto. Para obtener información detallada sobre el uso del filtro de sombra y otros filtros de objetos de visualización en ActionScript, consulte "Aplicación de filtros a objetos de visualización" en la página 279.

Desplazamiento lateral y vertical de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si un objeto de visualización es demasiado grande para el área donde se desea mostrar, se puede utilizar la propiedad `scrollRect` para definir el área visualizable del objeto de visualización. Además, si se cambia la propiedad `scrollRect` como respuesta a la entrada del usuario, se puede desplazar el contenido horizontal o verticalmente.

La propiedad `scrollRect` es una instancia de la clase Rectangle, que es una clase que combina los valores necesarios para definir un área rectangular como un solo objeto. Para definir inicialmente el área visualizable del objeto de visualización, se debe crear una nueva instancia de Rectangle y asignarla a la propiedad `scrollRect` del objeto de visualización. Posteriormente, para el desplazamiento vertical u horizontal, se lee la propiedad `scrollRect` en una variable Rectangle independiente y se cambia la propiedad deseada (por ejemplo, la propiedad `x` de la instancia de Rectangle para el desplazamiento horizontal o la propiedad `y` para el desplazamiento vertical). A continuación, se reasigna dicha instancia de Rectangle a la propiedad `scrollRect` para notificar el valor cambiado al objeto de visualización.

Por ejemplo, el código siguiente define el área visualizable de un objeto TextField denominado `bigText` que es demasiado alto para los límites del archivo SWF. Cuando se hace clic en dos botones denominados `up` y `down`, se llama a funciones que desplazan verticalmente el contenido del objeto TextField a través de la modificación de la propiedad `y` de la instancia de Rectangle `scrollRect`.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

Tal y como se ilustra en este ejemplo, cuando se trabaja con la propiedad `scrollRect` de un objeto de visualización, es mejor especificar que Flash Player o AIR deben almacenar en caché el contenido del objeto de visualización como un mapa de bits, usando para ello la propiedad `cacheAsBitmap`. De este modo, Flash Player o AIR no tienen que volver a dibujar todo el contenido del objeto de visualización cada vez que se desplaza verticalmente y, en su lugar, pueden utilizar el mapa de bits almacenado en caché para representar la parte necesaria directamente en pantalla. Para obtener más información, consulte "Almacenamiento en caché de los objetos de visualización" en la página 187.

Manipulación del tamaño y ajuste de escala de los objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Hay dos formas posibles de medir y manipular el tamaño de un objeto de visualización: con las propiedades de dimensión (`width` y `height`) o las propiedades de escala (`scaleX` y `scaleY`).

Cada objeto de visualización tiene una propiedad `width` y una propiedad `height`, definidas inicialmente con el tamaño del objeto en píxeles. La lectura de los valores de estas propiedades permite medir el tamaño del objeto de visualización. También es posible especificar nuevos valores para cambiar el tamaño del objeto, como se indica a continuación:

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Al cambiar la altura (`height`) o anchura (`width`) de un objeto de visualización, se ajusta la escala del objeto, de modo que el contenido del mismo se contrae o expande para ajustarse a la nueva área. Si el objeto de visualización solo contiene formas vectoriales, dichas formas volverán a dibujarse en la nueva escala sin perder la calidad. Los elementos gráficos de mapa de bits del objeto de visualización ajustarán su escala en lugar de dibujarse de nuevo. Así pues, por ejemplo, una foto digital cuya anchura y altura hayan aumentado más allá de las dimensiones reales de la información de píxeles de la imagen aparecerá pixelada y con bordes dentados.

Cuando se cambian las propiedades `width` o `height` de un objeto de visualización, Flash Player y AIR actualizan también las propiedades `scaleX` y `scaleY` del objeto.

Nota: *los objetos TextField son una excepción a este comportamiento de escala. Los campos de texto deben ajustar su tamaño automáticamente para alojar el texto y los tamaños de fuente. Por ello, reajustan sus valores scaleX o scaleY a 1 tras el cambio de tamaño. Sin embargo, si se ajustan los valores scaleX o scaleY de un objeto TextField, los valores de anchura y altura cambian para escalar los valores proporcionados.*

Estas propiedades representan el tamaño relativo del objeto de visualización con respecto a su tamaño original. Las propiedades `scaleX` y `scaleY` utilizan valores fraccionarios (decimales) para representar el porcentaje. Por ejemplo, si se cambia la anchura (`width`) de un objeto de visualización a la mitad de su tamaño original, la propiedad `scaleX` del objeto tendrá el valor `.5`, que significa 50 por ciento. Si se duplica la altura, la propiedad `scaleY` tendrá el valor `2`, que significa 200 por ciento.

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

Los cambios de tamaño no son proporcionales. Dicho de otro modo, si se cambia la altura (`height`) de un cuadrado pero no la anchura (`width`), ya no tendrá las mismas proporciones y ya no será un cuadrado sino un rectángulo. Si se desea cambiar relativamente el tamaño de un objeto de visualización, se pueden definir los valores de las propiedades `scaleX` y `scaleY` para cambiar el tamaño del objeto, en lugar de definir las propiedades `width` o `height`. Por ejemplo, este código cambia la anchura (`width`) del objeto de visualización denominado `square` y luego modifica la escala vertical (`scaleY`) para que coincida con la escala horizontal, de modo que el tamaño del cuadrado siga siendo proporcional.

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

Control de la distorsión durante el ajuste de escala

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Normalmente, cuando se ajusta la escala de un objeto de visualización (por ejemplo, si se expande horizontalmente), la distorsión resultante se extiende por igual en todo el objeto, de modo que cada parte se expande en la misma cantidad. Este es probablemente el comportamiento deseado para los gráficos y elementos de diseño. Sin embargo, a veces es preferible tener el control de las partes del objeto de visualización que se expanden y de las que permanecen invariables. Un ejemplo habitual es un botón que es un rectángulo con bordes redondeados. Con el ajuste de escala normal, las esquinas del botón se expandirán de modo que el radio de la esquina cambie a medida que el botón cambia de tamaño.

Sin embargo, en este caso es preferible tener el control del ajuste de escala, es decir, poder designar las áreas en las que debe ajustarse la escala (los lados rectos y la parte central) y las áreas en las que no, de forma que el ajuste de escala se produzca sin una distorsión visible.

Se puede usar la escala en 9 divisiones (Scale-9) para crear objetos de visualización cuyo ajuste de escala pueda controlarse. Con la escala en 9 divisiones, el objeto de visualización se divide en nueve rectángulos independientes (una cuadrícula de 3 por 3, como la cuadrícula de un tablero de tres en raya). Los rectángulos no son necesariamente del mismo tamaño; el usuario designa la ubicación de las líneas de la cuadrícula. El contenido de los cuatro rectángulos de bordes redondeados (como las esquinas redondeadas de un botón) no se expandirá ni comprimirá cuando se ajuste la escala del objeto de visualización. Los rectángulos superior central e inferior central ajustarán la escala horizontalmente pero no verticalmente, mientras que los rectángulos izquierdo central y derecho central ajustarán la escala verticalmente pero no horizontalmente. El rectángulo central ajustará la escala tanto vertical como horizontalmente.

Teniendo esto en cuenta, cuando se crea un objeto de visualización y se desea que nunca se ajuste la escala de determinado contenido, simplemente hay que asegurarse de que las líneas divisorias de la cuadrícula de escala en 9 divisiones se sitúen de forma que el contenido finalice en uno de los rectángulos de bordes redondeados.

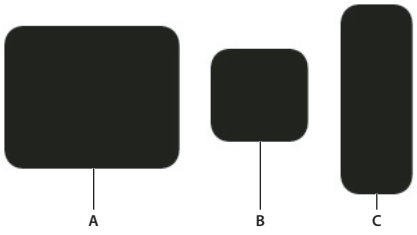
En ActionScript, al establecer un valor de la propiedad `scale9Grid` de un objeto de visualización, se activa la escala en 9 divisiones en el objeto y se define el tamaño de los rectángulos en la cuadrícula Scale-9 del objeto. Se utiliza una instancia de la clase Rectangle como valor de la propiedad `scale9Grid`, del siguiente modo:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Los cuatro parámetros del constructor de Rectangle son la coordenada x, la coordenada y, la anchura y la altura. En este ejemplo, la esquina superior izquierda del rectángulo se sitúa en el punto x: 32, y: 27 en el objeto de visualización denominado `myButton`. El rectángulo tiene una anchura de 71 píxeles y una altura de 64 píxeles (es decir, su borde derecho está en la coordenada x 103 del objeto de visualización y su borde inferior en la coordenada y 92 del objeto de visualización).

El área real contenida en el área definida por la instancia de Rectangle representa el rectángulo central de la cuadrícula Scale-9. Para calcular los demás rectángulos, Flash Player y AIR amplían los lados de la instancia de Rectangle, como se muestra a continuación:

En este caso, a medida que el botón ajusta la escala para expandirse o comprimirse, las esquinas redondeadas no se expanden ni comprimen, pero las demás áreas se ajustan a la escala.



A. *myButton.width = 131;myButton.height = 106;* ***B.*** *myButton.width = 73;myButton.height = 69;* ***C.*** *myButton.width = 54;myButton.height = 141;*

Almacenamiento en caché de los objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El tamaño de sus diseños en Flash irá creciendo, tanto si está creando una aplicación como si realiza complejas animaciones mediante scripts, por lo que deberá tener en cuenta el rendimiento y la optimización. Si tiene contenido que permanece estático (como una instancia de Shape rectangular), Flash Player y AIR no optimizarán el contenido. Por lo tanto, si cambia la posición del rectángulo, Flash Player o AIR redibujan toda la instancia de Shape.

Se pueden almacenar en caché los objetos de visualización especificados para mejorar el rendimiento del archivo SWF. El objeto de visualización es una *superficie*, básicamente una versión de mapa de bits de los datos vectoriales de la instancia, que son datos que no deseará que cambien mucho a lo largo del archivo SWF. Por consiguiente, las instancias para las que está activada la caché no se vuelven a dibujar continuamente mientras se reproduce el archivo SWF, lo que permite que el archivo SWF se represente rápidamente.

Nota: *es posible actualizar los datos vectoriales, momento en el cual se recrea la superficie. Por tanto, los datos vectoriales almacenados en caché en la superficie no tienen por qué permanecer intactos en todo el archivo SWF.*

Al establecer la propiedad `cacheAsBitmap` del objeto de visualización en `true`, el objeto de visualización almacena en caché una representación de mapa de bits de sí mismo. Flash Player o AIR crean un objeto de superficie para la instancia, que es un mapa de bits almacenado en caché, en lugar de los datos vectoriales. Si cambia los límites del objeto de visualización, la superficie se recrea en lugar de modificarse su tamaño. Las superficies pueden anidarse dentro de otras superficies. La superficie secundaria copia el mapa de bits en su superficie principal. Para obtener más información, consulte "Activación de almacenamiento de mapas de bits en caché" en la página 190.

Las propiedades `opaqueBackground` y `scrollRect` de la clase DisplayObject se relacionan con el almacenamiento en caché de mapa de bits a través de la propiedad `cacheAsBitmap`. Aunque estas propiedades son independientes entre sí, las propiedades `opaqueBackground` y `scrollRect` funcionan mejor cuando un objeto se almacena en caché como un mapa de bits; las ventajas de rendimiento de las propiedades `opaqueBackground` y `scrollRect` solo se evidencian cuando se establece `cacheAsBitmap` en `true`. Para obtener más información sobre el desplazamiento del contenido de objetos de visualización, consulte "Desplazamiento lateral y vertical de objetos de visualización" en la página 183. Para obtener más información sobre la configuración de un fondo opaco, consulte "Configuración de un color de fondo opaco" en la página 191.

Para obtener información sobre el enmascaramiento del canal alfa, que requiere que establezca la propiedad `cacheAsBitmap` como `true`, consulte "Enmascarar objetos de visualización" en la página 195.

Cuándo es conveniente activar la caché

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La activación de la caché para un objeto de visualización crea una superficie, lo que presenta algunas ventajas como, por ejemplo, una mayor velocidad de representación de animaciones vectoriales complejas. Existen varias situaciones en las que deseará activar la caché. Podría parecer que siempre es preferible activar la caché para mejorar el rendimiento de los archivos SWF; sin embargo, hay situaciones en las que la activación de la caché no mejora el rendimiento e incluso lo reduce. En esta sección se describen situaciones en las que debe utilizarse la activación de la caché y en las que deben emplearse objetos de visualización normales.

El rendimiento global de los datos almacenados en caché depende de la complejidad de los datos vectoriales de las instancias, de la cantidad de datos que cambie y de si ha establecido la propiedad `opaqueBackground`. Si cambia zonas pequeñas, la diferencia entre el uso de una superficie y el uso de datos vectoriales puede ser insignificante. Es aconsejable probar ambas situaciones antes de implementar la aplicación.

Cuándo es conveniente utilizar la caché de mapa de bits

A continuación se incluyen situaciones típicas en las que pueden apreciarse ventajas significativas al activar la caché de mapa de bits.

- Imagen de fondo compleja Una aplicación que contiene una imagen de fondo compleja y detallada de datos de vectoriales (quizás una imagen en la que aplica el comando Trazar mapa de bits o ilustraciones que ha creado en Adobe Illustrator®). Podrían animarse los caracteres del fondo, lo que ralentizaría la animación porque el fondo necesita la regeneración constante de los datos vectoriales. Para mejorar el rendimiento, se puede establecer la propiedad `opaqueBackground` del objeto de visualización de fondo en `true`. El fondo se representa como mapa de bits y puede redibujarse rápidamente, por lo que la animación se reproduce con mucha mayor velocidad.

- Campo de texto con desplazamiento: aplicación que muestra una gran cantidad de texto en un campo de texto con desplazamiento. Se puede colocar el campo de texto en un objeto de visualización definido como desplazable con límites con desplazamiento (propiedad `scrollRect`). De este modo, se permite un desplazamiento rápido por los píxeles de la instancia especificada. Cuando el usuario se desplaza por la instancia del objeto de visualización, Flash Player o AIR mueven hacia arriba los píxeles desplazados y generan la región recién expuesta en lugar de regenerar todo el campo de texto.
- Sistema de ventanas: aplicación con un complejo sistema de ventanas superpuestas. Cada ventana puede abrirse o cerrarse (por ejemplo, las ventanas de un navegador web). Si se marca cada ventana como una superficie (estableciendo la propiedad `cacheAsBitmap` en `true`), cada ventana se aísla y se almacena en caché. Los usuarios pueden arrastrar las ventanas para que se puedan superponer y cada ventana no necesita regenerar el contenido vectorial.
- Enmascaramiento del canal alfa: si se usa el enmascaramiento del canal alfa, la propiedad `cacheAsBitmap` se debe establecer en `true`. Para obtener más información, consulte "Enmascarar objetos de visualización" en la página 195.

La activación de la caché de mapa de bits en todas estas situaciones mejora el nivel de respuesta e interactividad de la aplicación, pues optimiza los gráficos vectoriales.

Asimismo, si se aplica un filtro a un objeto de visualización, `cacheAsBitmap` se establece automáticamente en `true`, aunque el usuario lo establezca explícitamente en `false`. Si se quitan todos los filtros del objeto de visualización, la propiedad `cacheAsBitmap` recupera el valor que se estableció por última vez.

Cuándo es conveniente evitar utilizar la caché de mapa de bits

El uso de esta función en circunstancias inadecuadas, puede afectar de forma negativa al rendimiento del archivo SWF. Cuando utilice la caché de mapa de bits, recuerde las siguientes directrices:

- No haga un uso abusivo de las superficies (objetos de visualización para los que está activada la caché). Cada superficie utiliza más memoria que un objeto de visualización normal, lo que significa que solo deberá utilizar las superficies cuando necesite mejorar el rendimiento de la representación.

  Un mapa de bits almacenado en caché utiliza bastante más memoria que un objeto de visualización normal. Por ejemplo, si una instancia de Sprite en el escenario tiene un tamaño de 250 por 250 píxeles, al almacenarse en caché podría utilizar 250 KB en lugar de 1 KB cuando se trata de una instancia de Sprite normal (no almacenada en caché).
- Evite aplicar zoom a las superficies almacenadas en caché. Si utiliza en exceso la caché de mapa de bits, se consume una gran cantidad de memoria (consulte el apartado anterior), especialmente si aplica el zoom para acercar el contenido.
- Utilice superficies para instancias de objetos de visualización que sean principalmente estáticas (sin animación). Puede arrastrar o mover la instancia, pero el contenido de la instancia no debe incluir demasiada animación ni cambiar mucho. (La animación o el cambio de contenido son más frecuentes en las instancias de MovieClip que contienen animación o en las instancias de Video.) Por ejemplo, si gira o transforma una instancia, esta cambia entre la superficie y los datos vectoriales, lo que dificulta el procesamiento y afecta de forma negativa al archivo SWF.
- Si mezcla superficies con datos vectoriales, aumentará la cantidad de procesamiento que deben llevar a cabo Flash Player y AIR (y, a veces, el equipo). Agrupe las superficies tanto como sea posible; por ejemplo, cuando cree aplicaciones de gestión de ventanas.

- No almacene en caché los objetos cuyos gráficos cambien frecuentemente. Cada vez que se aplica la escala, el sesgo, la rotación del objeto de visualización, el cambio del parámetro alfa o la transformación de color, el movimiento de los objetos de visualización secundarios o el dibujo utilizando la propiedad de gráficos, la caché de mapa de bits se vuelve a dibujar. Si esto sucede en cada fotograma, el motor de ejecución debe dibujar el objeto en un mapa de bits y, a continuación, copiar ese mapa de bits en el escenario, lo que implica un trabajo adicional si se compara con solo dibujar el objeto no almacenado en caché en el escenario. El inconveniente menor de rendimiento del almacenamiento en caché frente a la frecuencia de actualización depende de la complejidad y el tamaño del objeto de visualización y solo se puede determinar probando el contenido específico.

Activación de almacenamiento de mapas de bits en caché

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para activar la caché de mapa de bits para un objeto de visualización, debe establecerse su propiedad `cacheAsBitmap` en `true`:

```
mySprite.cacheAsBitmap = true;
```

Una vez establecida la propiedad `cacheAsBitmap` en `true`, es posible observar que el objeto de visualización ajusta automáticamente los píxeles a coordenadas enteras. Al probar el archivo SWF, debería apreciarse un aumento considerable en la velocidad de representación de imágenes vectoriales complejas.

Si se dan una o varias de las siguientes condiciones, no se crea ninguna superficie (mapa de bits almacenado en caché) aunque `cacheAsBitmap` se haya establecido en `true`:

- El mapa de bits tiene una altura o una anchura superior a 2880 píxeles.
- No se puede asignar el mapa de bits (se produce un error de memoria insuficiente).

Matrices de transformación de mapa de bits almacenadas en caché

Adobe AIR 2.0 y posterior (perfil móvil)

En las aplicaciones de AIR para dispositivos móviles, es necesario establecer la propiedad `cacheAsBitmapMatrix` siempre que se defina la propiedad `cacheAsBitmap`. El establecimiento de esta propiedad permite aplicar un mayor rango de transformaciones al objeto de visualización sin activar la representación.

```
mySprite.cacheAsBitmap = true;
mySprite.cacheAsBitmapMatrix = new Matrix();
```

Cuando se establece esta propiedad de matriz, se puede aplicar la siguiente transformación adicional al objeto de visualización sin volver a almacenar el objeto en caché:

- Movimiento o conversión sin ajuste a píxeles
- Girar
- Escala
- Sesgo
- Cambio de alfa (transparencia de entre 0 y 100%)

Estas transformaciones se aplican directamente al mapa de bits almacenado en caché.

Configuración de un color de fondo opaco

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede configurar un fondo opaco en un objeto de visualización. Por ejemplo, cuando el archivo SWF tiene un fondo que contiene complejos gráficos vectoriales, se puede establecer la propiedad `opaqueBackground` en un color especificado (normalmente el mismo color del escenario). El color se especifica en forma de número (normalmente un valor de color hexadecimal). El fondo se trata entonces como un mapa de bits, lo que ayuda a optimizar el rendimiento.

Cuando se establece la propiedad `cacheAsBitmap` en `true` y la propiedad `opaqueBackground` en un color especificado, la propiedad `opaqueBackground` permite que el mapa de bits interno sea opaco y se represente más rápido. Si no se establece `cacheAsBitmap` en `true`, la propiedad `opaqueBackground` añade una forma cuadrada vectorial opaca al fondo del objeto de visualización. No crea un mapa de bits automáticamente.

En el ejemplo siguiente se muestra cómo configurar el fondo de un objeto de visualización para optimizar el rendimiento:

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

En este caso, el color de fondo del objeto Shape denominado `myShape` se establece en rojo (`0xFF0000`). Suponiendo que la instancia de Shape contiene un dibujo de un triángulo verde, en un escenario con un fondo blanco, se mostraría como un triángulo verde con color rojo en el espacio vacío del recuadro de delimitación de la instancia de Shape (el rectángulo que abarca completamente Shape).

Por supuesto, este código tendría más sentido si se utilizara con un escenario que tuviera un fondo de color rojo sólido. Con un fondo de otro color, dicho color se especificaría en su lugar. Por ejemplo, en un archivo SWF con un fondo blanco, la propiedad `opaqueBackground` probablemente se establecería en `0xFFFFFF`, o blanco puro.

Aplicación de modos de mezcla

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los modos de mezcla implican combinar los colores de una imagen (la imagen base) con los de otra imagen (la imagen de mezcla) para producir una tercera imagen, que es la que se muestra realmente en la pantalla. Cada valor de píxel de una imagen se procesa con el correspondiente valor de píxel de la otra imagen para producir un valor de píxel para esa misma posición en el resultado.

Cada objeto de visualización tiene una propiedad `blendMode` que puede establecerse en uno de los siguientes modos de mezcla. Se trata de constantes definidas en la clase BlendMode. Como alternativa, se pueden usar los valores de String (entre paréntesis) que son los valores reales de las constantes.

- `BlendMode.ADD` (`"add"`): suele utilizarse para crear un efecto de disolución de aclarado animado entre dos imágenes.
- `BlendMode.ALPHA` (`"alpha"`): suele utilizarse para aplicar la transparencia del primer plano al fondo. (No se admite en procesamiento de GPU.)
- `BlendMode.DARKEN` (`"darken"`): suele utilizarse para superponer el tipo. (No se admite en procesamiento de GPU.)

- `BlendMode.DIFFERENCE` (`"difference"`): suele utilizarse para crear colores más vivos.
- `BlendMode.ERASE` (`"erase"`): se suele utilizar para recortar (borrar) parte del fondo utilizando el alfa de primer plano. (No se admite en procesamiento de GPU.)
- `BlendMode.HARDLIGHT` (`"hardlight"`): suele utilizase para crear efectos de sombreado. (No se admite en procesamiento de GPU.)
- `BlendMode.INVERT` (`"invert"`): se utiliza para invertir el fondo.
- `BlendMode.LAYER` (`"layer"`): se utiliza para forzar la creación de un búfer temporal para la composición previa de un determinado objeto de visualización. (No se admite en procesamiento de GPU.)
- `BlendMode.LIGHTEN` (`"lighten"`): suele utilizarse para superponer el tipo. (No se admite en procesamiento de GPU.)
- `BlendMode.MULTIPLY` (`"multiply"`): suele utilizarse para crear efectos de sombras y profundidad.
- `BlendMode.NORMAL` (`"normal"`): se utiliza para especificar que los valores de píxeles de la imagen mezclada sustituyan a los de la imagen base.
- `BlendMode.OVERLAY` (`"overlay"`): se suele utilizar para crear efectos de sombreado. (No se admite en procesamiento de GPU.)
- `BlendMode.SCREEN` (`"screen"`): se suele emplear para crear resaltados y destellos de lentes.
- `BlendMode.SHADER` (`"shader"`): se utiliza para especificar que se usa un sombreado de Pixel Bender para crear un efecto de fusión personalizado. Para obtener más información sobre el uso de sombreados, consulte "Trabajo con sombreados de Pixel Bender" en la página 314. (No se admite en procesamiento de GPU.)
- `BlendMode.SUBTRACT` (`"subtract"`): suele utilizarse para crear un efecto de disolución de oscurecimiento animado entre dos imágenes.

Ajuste de los colores de DisplayObject

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar los métodos de la clase ColorTransform (flash.geom.ColorTransform) para ajustar el color de un objeto de visualización. Cada objeto de visualización tiene una propiedad `transform`, que es una instancia de la clase Transform y que contiene información sobre diversas transformaciones que se aplican al objeto de visualización (como la rotación, los cambios en la escala o posición, entre otros). Además de esta información sobre las transformaciones geométricas, la clase Transform también incluye una propiedad `colorTransform`, que es una instancia de la clase ColorTransform y que permite realizar ajustes de color en el objeto de visualización. Para acceder a la información sobre transformación de color de un objeto de visualización, se puede utilizar código como el siguiente:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

Una vez creada una instancia de ColorTransform, se pueden leer sus valores de propiedades para saber cuáles son las transformaciones de color que ya se han aplicado o se pueden definir dichos valores para realizar cambios de color en el objeto de visualización. Para actualizar el objeto de visualización después de haber realizado cambios, es necesario reasignar la instancia de ColorTransform a la propiedad `transform.colorTransform`.

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

Configuración de los valores de color a través del código

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La propiedad `color` de la clase ColorTransform puede utilizarse para asignar un valor específico de color rojo, verde, azul (RGB) al objeto de visualización. En el ejemplo siguiente se usa la propiedad `color` para cambiar a azul el color del objeto de visualización denominado `square`, cuando el usuario haga clic en un botón denominado `blueBtn`:

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Cuando se cambia el color de un objeto de visualización a través de la propiedad `color`, se cambia completamente el color de todo el objeto, independientemente de que el objeto tuviera varios colores. Por ejemplo, si hay un objeto de visualización que contiene un círculo verde con texto negro encima, al definir como una sombra de rojo la propiedad `color` de la instancia de ColorTransform asociada a dicho objeto, todo el objeto, círculo y texto, se volverá rojo (de modo que el texto ya no podrá distinguirse del resto del objeto).

Modificación de efectos de color y brillo a través del código

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Supongamos que tiene un objeto de visualización con varios colores (por ejemplo, una foto digital) y no desea modificar los colores de todo el objeto, sino únicamente ajustar el color de un objeto de visualización basándose en los colores existentes. En esta situación, la clase ColorTransform incluye una serie de propiedades de multiplicador y desplazamiento que pueden utilizarse para realizar este tipo de ajuste. Las propiedades de multiplicador, denominadas `redMultiplier`, `greenMultiplier`, `blueMultiplier` y `alphaMultiplier`, funcionan como filtros fotográficos de color (o gafas de sol de color), ya que amplifican o reducen determinados colores en el objeto de visualización. Las propiedades de desplazamiento (`redOffset`, `greenOffset`, `blueOffset` y `alphaOffset`) pueden utilizarse para añadir cantidades adicionales de un determinado color al objeto o para especificar el valor mínimo que puede tener un determinado color.

Estas propiedades de multiplicador y desplazamiento son idénticas a la configuración avanzada de color disponible para los símbolos de clip de película en la herramienta de edición de Flash que se muestra al elegir Avanzado en el menú emergente Color del inspector de propiedades.

En el código siguiente se carga una imagen JPEG y se aplica una transformación de color que modifica los canales rojo y verde a medida que el puntero del ratón se mueve por los ejes x e y. En este caso, como no se especifica ningún valor de desplazamiento, el valor de cada uno de los canales de color mostrados en pantalla será un porcentaje del valor de color original de la imagen, lo que significa que la mayoría de rojo y verde mostrado en cualquier píxel será la cantidad original de rojo o verde de dicho píxel.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

Rotación de objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos de visualización pueden girarse a través de la propiedad `rotation`. Se puede leer este valor para saber si se ha girado un objeto. Para girar el objeto, se puede establecer esta propiedad en un número (en grados) que representa la cantidad de rotación que se aplica al objeto. Por ejemplo, en la siguiente línea de código se gira el objeto `square` 45 grados (un octavo de vuelta):

```
square.rotation = 45;
```

Para girar un objeto de visualización también se puede utilizar una matriz de transformación; esto se describe en "Trabajo con la geometría" en la página 217.

Aparición o desaparición progresiva de objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede controlar la transparencia de un objeto de visualización para hacerlo transparente (parcialmente o en su totalidad) o cambiar la transparencia para hacer que el objeto aparezca o desaparezca progresivamente. La propiedad `alpha` de la clase DisplayObject define la transparencia (o, mejor dicho, la opacidad) de un objeto de visualización. La propiedad `alpha` puede definirse como cualquier valor entre 0 y 1, siendo 0 completamente transparente y 1 completamente opaco. Por ejemplo, con las siguientes líneas de código, el objeto denominado `myBall` se hace parcialmente transparente (en un 50 %) cuando se hace clic en él con el ratón:

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

También se puede modificar la transparencia de un objeto de visualización mediante los ajustes de color disponibles a través de la clase ColorTransform. Para obtener más información, consulte "Ajuste de los colores de DisplayObject" en la página 192.

Enmascarar objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar un objeto de visualización como una máscara para crear un agujero a través del cual se ve el contenido de otro objeto de visualización.

Definición de una máscara

Para indicar que un objeto de visualización será la máscara de otro objeto de visualización, es preciso definir el objeto de máscara como la propiedad `mask` del objeto de visualización que se va a enmascarar:

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

El objeto de visualización con máscara aparece bajo todas las zonas opacas (no transparentes) del objeto de visualización que actúa como máscara. Por ejemplo, el código siguiente crea una instancia de Shape que contiene un cuadrado rojo de 100 por 100 píxeles y una instancia de Sprite que contiene un círculo azul con un radio de 25 píxeles. Cuando se hace clic en un círculo, se define como la máscara del cuadrado, de forma que la única parte del cuadrado que aparece visible es la que queda cubierta por la parte sólida del círculo. Dicho de otro modo, solo se ve un círculo rojo.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

El objeto de visualización que actúa como máscara puede arrastrarse, animarse, cambiar de tamaño automáticamente y usar formas independientes en una sola máscara. No es necesario añadir el objeto de visualización de máscara a la lista de visualización. Sin embargo, si se desea ajustar la escala del objeto de máscara cuando se ajusta la escala del escenario, o permitir la interacción del usuario con la máscara (por ejemplo, el arrastre o cambio de tamaño controlado por el usuario), el objeto de máscara debe añadirse a la lista de visualización. El índice z real (de delante a atrás) de los objetos de visualización no es relevante, siempre y cuando el objeto de máscara se añada a la lista de visualización. (El objeto de máscara solo aparecerá en la pantalla como una máscara.) Si el objeto de máscara es una instancia de MovieClip con varios fotogramas, reproduce todos ellos en su línea de tiempo, tal y como haría si no actuara como una máscara. Para eliminar una máscara, debe establecerse la propiedad `mask` en `null`:

```
// remove the mask from mySprite
mySprite.mask = null;
```

No puede utilizar una máscara para enmascarar otra máscara ni definir la propiedad `alpha` de un objeto de visualización de máscara. En un objeto de visualización que se usa como máscara, solo se pueden utilizar los rellenos; los trazos se pasan por alto.

AIR 2

Si un objeto de visualización enmascarado se almacena en caché estableciendo las propiedades `cacheAsBitmap` y `cacheAsBitmapMatrix`, la máscara debe ser un objeto secundario del objeto de visualización enmascarado. Del mismo modo, si el objeto de visualización enmascarado es un descendiente de un contenedor del objeto de visualización que está almacenado en caché, tanto la máscara como el objeto de visualización deben ser descendentes de dicho contenedor. Si el objeto enmascarado es un descendiente de varios contenedores del objeto de visualización almacenado en caché, la máscara debe ser un descendiente del contenedor almacenado en caché más cercano al objeto enmascarado en la lista de visualización.

Enmascaramiento de fuentes de dispositivo

Se puede utilizar un objeto de visualización para enmascarar un texto definido en una fuente de dispositivo. Cuando se utiliza un objeto de visualización para enmascarar texto definido en una fuente de dispositivo, el recuadro de delimitación rectangular de la máscara se utiliza como la forma de máscara. Es decir, si se crea una máscara de objeto de visualización no rectangular para el texto de la fuente de dispositivo, la máscara que aparecerá en el archivo SWF tendrá la forma del recuadro de delimitación rectangular y no la de la máscara en sí.

Enmascaramiento del canal alfa

Se admite el enmascaramiento del canal alfa si tanto la máscara como los objetos de visualización enmascarados utilizan la caché de mapa de bits, como se muestra a continuación:

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Una aplicación de enmascaramiento del canal alfa consiste, por ejemplo, en usar un filtro del objeto de máscara independientemente del filtro aplicado al objeto de visualización enmascarado.

En el siguiente ejemplo se carga un archivo de imagen externo en el escenario. Dicha imagen (mejor dicho, la instancia de Loader en la que se ubica) será el objeto de visualización que se enmascare. Se dibuja un óvalo con degradado (de color negro sólido en el centro y con transparencia progresiva en los bordes) sobre la imagen; será la máscara alfa. Ambos objetos de visualización tienen activada la caché de mapa de bits. El óvalo se define como una máscara de la imagen y luego se convierte en un elemento que se puede arrastrar.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

Animación de objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La animación es el proceso de mover algo o, alternativamente, hacer que algo cambie con el tiempo. La animación mediante scripts es una parte fundamental de los videojuegos, que a menudo se utiliza para pulir y añadir útiles indicaciones de interacción a otras aplicaciones.

La idea fundamental en la que se basa la animación mediante scripts es la necesidad de realizar un cambio y de dividir dicho cambio en incrementos temporales. Repetir algo en ActionScript es tan sencillo como usar una sentencia de bucle común. Sin embargo, es necesario reproducir todas las repeticiones de un bucle para poder actualizar la visualización. Para crear animación mediante scripts, es necesario escribir código ActionScript que repita alguna acción a lo largo del tiempo y que además actualice la pantalla cada vez que se ejecuta.

Por ejemplo, imagine que desea crear una sencilla animación de una bola que se desplaza a través de la pantalla. ActionScript incluye un mecanismo sencillo que permite seguir el paso del tiempo y actualizar la pantalla de forma correspondiente, lo que significa que se puede escribir código que mueva la bola un poco cada vez hasta llegar a su destino. La pantalla se actualiza después de cada movimiento, de forma que el movimiento a través del escenario es visible para el espectador.

Desde un punto de vista práctico, tiene sentido sincronizar la animación mediante scripts con la velocidad de fotogramas del archivo SWF (es decir, hacer que una animación cambie cada vez que se muestra un nuevo fotograma), ya que define la frecuencia con la que Flash Player o AIR actualiza la pantalla. Cada objeto de visualización tiene un evento `enterFrame` que se distribuye según la velocidad de fotogramas del archivo SWF: un evento por fotograma. La mayoría de los desarrolladores que crean animación mediante scripts utilizan el evento `enterFrame` para crear acciones que se repiten a lo largo del tiempo. Se puede escribir un código que detecte el evento `enterFrame` y mueva la bola animada una cantidad determinada en cada fotograma y, al actualizarse la pantalla (en cada fotograma), redibuje la bola en la nueva ubicación, creando movimiento.

Nota: *otra forma de repetir una acción a lo largo del tiempo consiste en utilizar la clase Timer. Una instancia de Timer activa una notificación de evento cada vez que pasa un tiempo determinado. Se puede escribir código que lleve a cabo la animación mediante la gestión del evento timer de la clase Timer, si se define un intervalo de tiempo pequeño (alguna fracción de un segundo). Para obtener más información sobre el uso de la clase Timer, consulte "Control de intervalos de tiempo" en la página 4.*

En el ejemplo siguiente, se crea en el escenario una instancia de Sprite circular, denominada `circle`. Cuando el usuario hace clic en el círculo, se inicia una secuencia de animación mediante scripts que provoca la desaparición progresiva de `circle` (se reduce su propiedad `alpha`) hasta que es completamente transparente:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

Cuando el usuario hace clic en el círculo, se suscribe la función `fadeCircle()` como detector del evento `enterFrame`, lo que significa que se inicia su llamada una vez en cada fotograma. Esa función hace que `circle` desaparezca progresivamente cambiando su propiedad `alpha`, de modo que, una vez en cada fotograma, el valor de `alpha` del círculo se reduce en 0,05 (5 por ciento) y se actualiza la pantalla. Finalmente, cuando el valor de `alpha` es 0 (`circle` es completamente transparente), se elimina la función `fadeCircle()` como detector de eventos y finaliza la animación.

Este mismo código podría utilizarse, por ejemplo, para crear un movimiento de animación en lugar de la desaparición progresiva. Si se sustituye otra propiedad de `alpha` en la función que actúa como detector del evento `enterFrame`, esa propiedad se animará. Por ejemplo, si se cambia esta línea

```
circle.alpha -= .05;
```

por este código

```
circle.x += 5;
```

se animará la propiedad `x`, haciendo que el círculo se mueva hacia la derecha a través del escenario. La condición que finaliza la animación puede cambiarse para que la animación termine (es decir, se quite la suscripción del detector de `enterFrame`) al llegar a la coordenada x especificada.

Orientación del escenario

AIR 2.0 y posterior

Los dispositivos móviles suelen reorientar la interfaz de usuario para mantener la visualización en orientación vertical cuando el usuario gira el dispositivo. Si se activa la orientación automática en la aplicación, el dispositivo mantiene la visualización orientada correctamente, pero depende del usuario el asegurar que el contenido se vea correctamente cuando cambie la relación de aspecto del escenario. Si se desactiva la orientación automática, la pantalla del dispositivo permanece fija a no ser que se modifique manualmente la orientación.

Las aplicaciones de AIR se ejecutan en una serie de diferentes dispositivos móviles y sistemas operativos. El comportamiento de orientación subyacente puede variar según los sistemas operativos e incluso en función de los distintos dispositivos en el mismo sistema operativo. Una sencilla estrategia de diseño, que funciona bien en todos los dispositivos y sistemas operativos, consiste en activar la orientación automática y detectar los eventos `resize` del escenario para determinar cuándo es necesario actualizar el diseño de la aplicación.

Como alternativa, si la aplicación solo admite la relación de aspecto vertical u horizontal, se puede desactivar la orientación automática y establecer la relación de aspecto admitida en el descriptor de la aplicación de AIR. Esta estrategia de diseño ofrece un comportamiento coherente y selecciona la "mejor" orientación para la relación de aspecto seleccionada. Por ejemplo, si especifica la relación de aspecto horizontal, la orientación elegida será la adecuada para dispositivos con modo horizontal y teclado emergente.

Indicación de la relación de aspecto y la orientación del escenario actual

La orientación se obtiene en relación con la posición normal del dispositivo. En la mayoría de los dispositivos, existe una posición clara vertical. Esta posición se considera la orientación *predeterminada*. Las demás orientaciones posibles son: *girado a la izquierda*, *girado a la derecha* y *al revés*. La clase StageOrientation define las constantes de cadena que se utilizan al establecer o comparar los valores de orientación.

La clase Stage define dos propiedades que indican la orientación:

- Stage.deviceOrientation: indica la orientación física del dispositivo respecto a la posición predeterminada.

  Nota: *la propiedad deviceOrientation no siempre está disponible cuando la aplicación se inicia en primer lugar o cuando el dispositivo se encuentra en posición horizontal. En estos casos, la orientación del dispositivo se muestra como desconocida.*

- Stage.orientation: indica la orientación del escenario respecto a la posición predeterminada. Cuando se activa la orientación automática, el escenario gira en dirección contraria al dispositivo para permanecer en posición vertical. De este modo, las posiciones derecha e izquierda indicadas por la propiedad `orientation` son las contrarias de las que se indican mediante la propiedad `deviceOrientation`. Por ejemplo, si `deviceRotation` indica *girado a la derecha*, `la orientación` indica *girado a la izquierda*.

La relación de aspecto del escenario se puede obtener simplemente comparando la achura y altura actuales del escenario:

```
var aspect:String = this.stage.stageWidth >= this.stage.stageHeight ?
StageAspectRatio.LANDSCAPE : StageAspectRatio.PORTRAIT;
```

Orientación automática

Si la orientación automática está activada y el usuario gira su dispositivo, el sistema operativo reorienta toda la interfaz de usuario,incluyendo la barra de tareas del sistema y la aplicación. Como resultado, la relación de aspecto del escenario cambia de vertical a horizontal, o viceversa. Cuando cambia la relación de aspecto, las dimensiones del escenario también cambian.

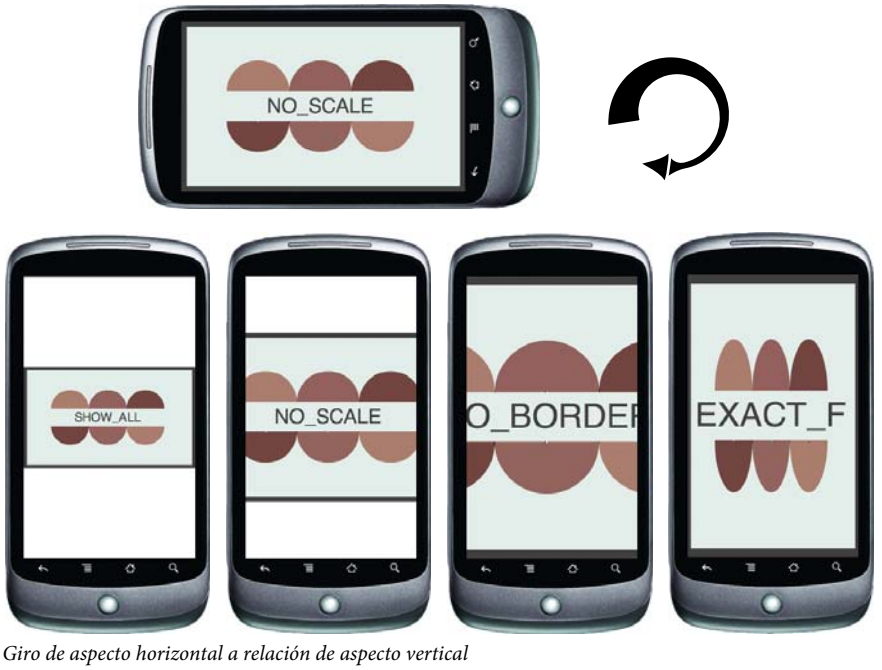
Active o desactive la orientación automática en tiempo de ejecución, estableciendo la propiedad `autoOrients` del escenario en `true` o `false`. Es posible establecer el valor inicial de esta propiedad en el descriptor de la aplicación de AIR con el elemento `<autoOrients>`. (Se debe tener en cuenta que antes de AIR 2.6, `autoOrients` es una propiedad de solo lectura y solo se puede establecer en el descriptor de la aplicación.)

Si especifica una relación de aspecto horizontal o vertical y también activa la orientación automática, AIR limitará la orientación automática a la relación de aspecto especificada.

Cambios de dimensión del escenario

Cuando cambian las dimensiones del escenario, el contenido del escenario se escala y se puede volver a situar tal y como se especifica mediante las propiedades `scaleMode` y `align` del objeto Stage. En la mayoría de los casos, al confiar en el comportamiento automático proporcionado por la configuración de `scaleMode` de Stage, no se obtienen buenos resultados. Es necesario volver a diseñar o dibujar los gráficos y componentes para que admitan varias relaciones de aspecto. (Proporcionar una lógica de diseño flexible también significa que la aplicación funcionará mejor en diversos dispositivos con diferentes tamaños de pantalla y relaciones de aspecto.)

En la siguiente ilustración se muestran los efectos de las diferentes configuraciones de `scaleMode` al girar un dispositivo móvil común:



Giro de aspecto horizontal a relación de aspecto vertical

La ilustración muestra el comportamiento del ajuste de escala que se produce al girar de una relación de aspecto horizontal a una relación de aspecto vertical con diferentes modos de escala. El giro de posición vertical a horizontal produce un conjunto similar de efectos.

Eventos de cambio de orientación

El objeto Stage distribuye dos tipos de eventos que se pueden utilizar para detectar y reaccionar frente a los cambios de orientación. Ambos eventos `resize` y `orientationChange` del escenario se distribuyen cuando se activa la orientación automática.

El evento *resize* es el más adecuado cuando en la orientación automática la pantalla debe permanecer en posición vertical. Cuando el escenario distribuya un evento `resize`, vuelva a diseñar o dibujar el contenido, según sea necesario. El evento `resize` solo se distribuye cuando el modo de escala del escenario se establece en `noScale`.

El evento `orientationChange` solo se puede utilizar para detectar cambios de orientación. El evento `orientationChange` solo se distribuye cuando se activa la orientación automática.

Nota: *en algunas plataformas móviles, el escenario distribuye un evento `orientationChanging` que se puede cancelar antes de distribuir los eventos resize u orientationChange. Debido a que el evento no se admite en todas las plataformas, se debe evitar basarse en el mismo.*

Orientación manual

AIR 2.6 y posterior

Es posible controlar la orientación del escenario con los métodos `setOrientation()` o `setAspectRatio()` del objeto Stage.

Definición de la orientación del escenario

Puede definir la orientación del escenario en tiempo de ejecución con el método `setOrientation()` del objeto Stage. Utilice las constantes de cadena definidas por la clase StageOrientation para especificar la orientación que desee:

```
this.stage.setOrientation( StageOrientation.ROTATED_RIGHT );
```

No todos los dispositivos y sistemas operativos admiten todas las orientaciones posibles. Por ejemplo, la programación de Android 2.2 no permite elegir mediante programación la orientación girada a la izquierda en dispositivos con orientación estándar vertical ni permite la orientación invertida en ningún caso. La propiedad `supportedOrientations` del escenario ofrece una lista con las orientaciones que pueden transferirse al método `setOrientation()`:

```
var orientations:Vector.<String> = this.stage.supportedOrientations;
for each( var orientation:String in orientations )
{
    trace( orientation );
}
```

Definición de la relación de aspecto del escenario

Si lo que más le importa es la relación de aspecto del escenario, puede definirla en vertical o en horizontal. Puede definir la relación de aspecto en el descriptor de la aplicación de AIR o, en tiempo de ejecución a través del método `setAspectRatio()` del objeto Stage:

```
this.stage.setAspectRatio( StageAspectRatio.LANDSCAPE );
```

El motor de ejecución seleccionará una de las dos orientaciones posibles para la relación de aspecto especificada. Puede no coincidir con la orientación actual del dispositivo. Por ejemplo, la orientación predeterminada se elige con orientación normal como preferencia (AIR 3.2 y versiones anteriores) y la orientación adecuada para el teclado virtual se selecciona con orientación contraria a la preferente.

(AIR 3.3 y versiones posteriores) A partir de AIR 3.3 (SWF versión 16), también es posible utilizar la constante `StageAspectRatio.ANY`. Si `Stage.autoOrients` se establece en `true` y se llama a `setAspectRatio(StageAspectRatio.ANY)`, la aplicación puede reorientar todas las orientaciones (horizontal-izquierda, horizontal-derecha, vertical y vertical-al revés). Otra novedad de AIR 3.3 es que la relación de aspecto se mantiene y la rotación adicional del dispositivo se limita a la orientación especificada.

Ejemplo: Definición de la orientación del escenario para coincidir con la del dispositivo

El siguiente ejemplo muestra una función que actualiza la orientación del escenario para que coincida con la orientación actual del dispositivo. La propiedad `deviceOrientation` del escenario indica la orientación física del dispositivo, incluso si está desactivada la orientación automática.

```
function refreshOrientation( theStage:Stage ):void
{
    switch ( theStage.deviceOrientation )
    {
        case StageOrientation.DEFAULT:
            theStage.setOrientation( StageOrientation.DEFAULT );
            break;
        case StageOrientation.ROTATED_RIGHT:
            theStage.setOrientation( StageOrientation.ROTATED_LEFT );
            break;
        case StageOrientation.ROTATED_LEFT:
            theStage.setOrientation( StageOrientation.ROTATED_RIGHT );
            break;
        case StageOrientation.UPSIDE_DOWN:
            theStage.setOrientation( StageOrientation.UPSIDE_DOWN );
            break;
        default:
            //No change
    }
}
```

El cambio de orientación es asíncrono. Es posible detectar el evento `orientationChange` distribuido por el escenario para detectar la finalización del cambio. Si un dispositivo no admite una orientación, la llamada `setOrientation()` falla sin emitir un error.

Carga dinámica de contenido de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En una aplicación de ActionScript 3.0, se puede cargar cualquiera de los siguientes activos de visualización externos:

- Un archivo SWF creado en ActionScript 3.0: este archivo puede ser un objeto Sprite, MovieClip o de cualquier clase que amplíe Sprite. En aplicaciones de AIR en iOS, solo se pueden cargar archivos SWF que no contienen código de bytes de ActionScript. Esto significa que los archivos SWF que contienen datos incorporados, como imágenes o sonido, se podrán cargar, pero no los archivos SWF que contengan código ejecutable.
- Un archivo de imagen: por ejemplo, archivos JPG, PNG y GIF.
- Un archivo SWF AVM1: un archivo SWF escrito en ActionScript 1.0 o 2.0. (no se admite en aplicaciones de AIR para móviles)

Estos activos se cargan mediante la clase Loader.

Carga de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos Loader se usan para cargar archivos SWF y archivos de gráficos en una aplicación. La clase Loader es una subclase de la clase DisplayObjectContainer. Un objeto Loader solo puede contener un objeto de visualización secundario en su lista de visualización: el objeto de visualización que representa el archivo SWF o archivo de gráficos que se carga. Cuando se añade un objeto Loader a la lista de visualización, como en el código siguiente, también se añade a la lista de visualización el objeto de visualización secundario cargado, una vez que se ha cargado:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

Cuando se carga el archivo SWF o la imagen, se puede mover el objeto de visualización cargado a otro contenedor de objeto de visualización, como el objeto DisplayObjectContainer `container` que se muestra en este ejemplo:

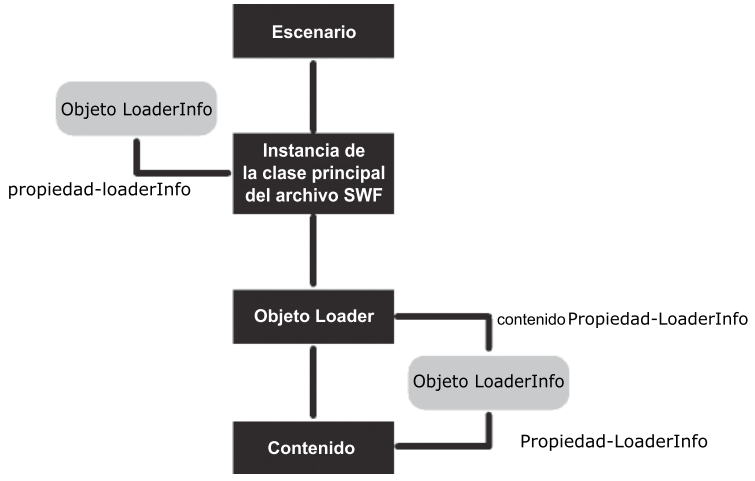
```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

Supervisión del progreso de carga

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se empieza a cargar el archivo, se crea un objeto LoaderInfo. Un objeto LoaderInfo proporciona información tal como el progreso de carga, los URL del cargador y el contenido cargado, el número total de bytes del medio, y la anchura y la altura nominal del medio. Un objeto LoaderInfo también distribuye eventos para supervisar el progreso de la carga.

El siguiente diagrama muestra los diferentes usos del objeto LoaderInfo para la instancia de la clase principal del archivo SWF, para un objeto Loader y para un objeto cargado por el objeto Loader:



Se puede acceder al objeto LoaderInfo como una propiedad tanto del objeto Loader como del objeto de visualización cargado. En cuanto comienza la carga, se puede acceder al objeto LoaderInfo a través de la propiedad `contentLoaderInfo` del objeto Loader. Cuando finaliza la carga del objeto de visualización, también es posible acceder al objeto LoaderInfo como una propiedad del objeto de visualización cargado, a través de la propiedad `loaderInfo` del objeto de visualización. La propiedad `loaderInfo` del objeto de visualización hace referencia al mismo objeto LoaderInfo al que se refiere la propiedad `contentLoaderInfo` del objeto Loader. Dicho de otro modo, un objeto LoaderInfo se comparte entre un objeto cargado y el objeto Loader que lo ha cargado (es decir, entre el contenido cargado y el cargador).

Para acceder a las propiedades del contenido cargado, es necesario añadir un detector de eventos al objeto LoaderInfo, como se muestra en el siguiente código:

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

Para obtener más información, consulte “Gestión de eventos” en la página 127.

Especificación del contexto de carga

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se carga un archivo externo en Flash Player o AIR con el método `load()` o `loadBytes()`, se puede especificar opcionalmente un parámetro `context`. Este parámetro es un objeto LoaderContext.

La clase LoaderContext incluye tres propiedades que permiten definir el contexto de uso del contenido cargado:

- `checkPolicyFile`: utilice esta propiedad solo si carga un archivo de imagen (no un archivo SWF). Si establece esta propiedad en `true`, Loader comprueba el servidor de origen de un archivo de política (consulte "Controles de sitio web (archivos de política)" en la página 1095). Solo es necesaria en el contenido procedente de dominios ajenos al del archivo SWF que contiene el objeto Loader. Si el servidor concede permisos al dominio de Loader, el código ActionScript de los archivos SWF del dominio de Loader puede acceder a los datos de la imagen cargada y, por lo tanto, se puede usar el comando `BitmapData.draw()` para acceder a los datos de la imagen cargada.

  Tenga en cuenta que un archivo SWF de otros dominios ajenos al del objeto Loader puede llamar a `Security.allowDomain()` para permitir el uso de un dominio específico.

- `securityDomain`: utilice esta propiedad solo si carga un archivo SWF (no una imagen). Esta propiedad se especifica en un archivo SWF de un dominio ajeno al del archivo que contiene el objeto Loader. Si se especifica esta opción, Flash Player comprueba la existencia de un archivo de política y, en caso de que exista uno, los archivos SWF de los dominios permitidos en el archivo de política entre dominios pueden reutilizar los scripts del contenido SWF cargado. Se puede especificar `flash.system.SecurityDomain.currentDomain` como este parámetro.

- `applicationDomain`: utilice esta propiedad solamente si carga un archivo SWF escrito en ActionScript 3.0 (no una imagen ni un archivo SWF escritos en ActionScript 1.0 o 2.0). Al cargar el archivo, se puede especificar que se incluya en el mismo dominio de aplicación del objeto Loader; para ello, hay que establecer el parámetro `applicationDomain` en `flash.system.ApplicationDomain.currentDomain`. Al colocar el archivo SWF cargado en el mismo dominio de aplicación, se puede acceder a sus clases directamente. Esto puede ser muy útil si se carga un archivo SWF que contiene medios incorporados, a los que se puede tener acceso a través de sus nombres de clase asociados. Para obtener más información, consulte "Trabajo con dominios de aplicación" en la página 150.

A continuación se muestra un ejemplo de comprobación de un archivo de política durante la carga de un mapa de bits de otro dominio:

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

A continuación se muestra un ejemplo de comprobación de un archivo de política durante la carga de un archivo SWF de otro dominio, con el fin de colocar el archivo en el mismo entorno limitado de seguridad que el objeto Loader. Además, el código añade las clases del archivo SWF cargado al mismo dominio de aplicación que el del objeto Loader:

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Para obtener más información, consulte la clase LoaderContext en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Carga de archivos SWF en AIR para iOS

Adobe AIR 3.6 y posterior, solo iOS

Con dispositivos iOS, hay restricciones de carga y compilación en tiempo de ejecución. Dichas restricciones implican ciertas diferencias en la tarea de carga de archivos SWF externos en la aplicación:

- Todos los archivos SWF que contienen código ActionScript deben incluirse en el paquete de la aplicación. No es posible cargar desde un origen externo archivos SWF que contengan código. Como parte del empaquetado de la aplicación, se compila el código ActionScript de todos los archivos SWF en el paquete de la aplicación como código nativo para dispositivos iOS.
- No puede cargar, descargar y volver a cargar un archivo SWF. Si lo intenta, se producirá un error.
- La carga en memoria y posterior descarga se realiza igual que en plataformas de escritorio. Al cargar un archivo SWF y luego descargarlo, todos los elementos visuales que contenga se descargan de la memoria. Pero no es el caso de las referencias de clase de ActionScript que pueda haber en el SWF cargado, que se mantienen en la memoria y son accesible en el código ActionScript.
- Todos los archivos SWF cargados deben cargarse en el mismo dominio de aplicación que el SWF principal. Dado que este comportamiento no es el predeterminado, por cada archivo SWF que cargue debe crear un objeto LoaderContext especificando el dominio de aplicación principal y pasar el objeto LoaderContext a la llamada a método Loader.load(). Si intenta cargar un SWF en un dominio de aplicación que no sea el de la aplicación de SWF principal, se producirá un error. Esto ocurre incluso si el SWF cargado contiene únicamente elementos visuales (sin código ActionScript).

  En el ejemplo siguiente se muestra el código que usar para cargar un archivo SWF del paquete de la aplicación en el dominio de la aplicación de SWF principal:

```
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("swfs/SecondarySwf.swf");
var loaderContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain,
null);
loader.load(url, loaderContext);
```

Si un archivo SWF contiene únicamente elementos (no código), se puede cargar desde el paquete de la aplicación o a través de una red. En ambos casos es preciso cargar el SWF en el dominio de la aplicación principal.

Para versiones de AIR anteriores a AIR 3.6, durante el proceso de compilación se quita el código de todos los archivos SWF a excepción del de la aplicación principal. En tiempo de ejecución se pueden incluir en el paquete de la aplicación archivos SWF que contengan elementos visuales pero no código. Si intenta cargar un SWF con código ActionScript, se producirá un error. De ser así aparecerá en la aplicación un cuadro de diálogo de ActionScript sin compilar.

Véase también
Packaging and loading multiple SWFs in AIR apps on iOS

Uso de las clases ProLoader y ProLoaderInfo

Flash Player 9 y versiones posteriores, Adobe AIR 1.0 y versiones posteriores y requiere Flash Professional CS5.5

Para ayudar con la precarga de la biblioteca remota compartida (RSL), Flash Professional CS5.5 introduce las clases fl.display.ProLoader y fl.display.ProLoaderInfo. Estas clases imitan a las clases flash.display.Loader y flash.display.LoaderInfo pero presentan un comportamiento de carga más coherente.

En concreto, ProLoader ayuda a cargar archivos SWF que utilizan Text Layout Framework (TLF) con precarga de RSL. En tiempo de ejecución, los archivos SWF que precargan otros archivos SWF o SWZ como, por ejemplo, TLF, requieren un archivo envolvente SWF solo interno. La capa adicional de complejidad impuesta por el archivo envolvente SWF puede tener un comportamiento no deseado. ProLoader soluciona esta complejidad para cargar estos archivos como si fueran archivos normales SWF. La solución utilizada por la clase ProLoader es transparente para el usuario y no requiere un control especial en ActionScript. Asimismo, ProLoader carga el contenido SWF normal correctamente.

En Flash Professional CS5.5 y posterior, puede reemplazar con seguridad todos los usos de la clase Loader con la clase ProLoader. A continuación, exporte su aplicación a Flash Player 10.2 o posterior para que ProLoader pueda acceder a la funcionalidad de ActionScript necesaria. También puede utilizar ProLoader mientras se utilizan versiones anteriores de Flash Player que admiten ActionScript 3.0. Sin embargo, se aprovechan completamente las funciones de ProLoader solo con Flash Player 10.2 o superior. Utilice siempre ProLoader cuando use TLF en Flash Professional CS5.5 o posterior. ProLoader no es necesario en entornos distintos a Flash Professional.

Importante: *para los archivos SWF publicados en Flash Professional CS5.5 y posterior, utilice siempre las clases fl.display.ProLoader y fl.display.ProLoaderInfo en lugar de flash.display.Loader y flash.display.LoaderInfo.*

Problemas abordados por la clase ProLoader

La clase ProLoader aborda problemas para los que la clase heredada Loader no estaba diseñada para gestionar. Estos problemas proceden de la precarga RSL de bibliotecas TLF. Concretamente, se aplican a los archivos SWF que utilizan un objeto Loader para cargar otros archivos SWF. Entre los problemas tratados se incluyen los siguientes:

- **La creación de scripts entre el archivo de carga y el archivo cargado no se comporta como estaba previsto.** La clase ProLoader establece automáticamente el archivo SWF en carga como elemento principal del archivo SWF cargado. De este modo, las comunicaciones del archivo SWF de carga van directamente al archivo SWF cargado.
- **La aplicación SWF debe gestionar activamente el proceso de carga.** Esto requiere la implementación de eventos adicionales como, por ejemplo, `added`, `removed`, `addedToStage` y `removedFromStage`. Si la aplicación se destina a Flash Player 10.2 o posterior, ProLoader elimina la necesidad de este trabajo adicional.

Actualización de código para utilizar con ProLoader en lugar de Loader

Como ProLoader imita a la clase Loader, es posible alternar fácilmente entre las dos clases en el código. El siguiente ejemplo muestra cómo actualizar el código existente para utilizar la nueva clase:

```
import flash.display.Loader;
import flash.events.Event;
var l:Loader = new Loader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

Este código se puede actualizar para usar ProLoader del modo siguiente:

```
import fl.display.ProLoader;
import flash.events.Event;
var l:ProLoader = new ProLoader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

Ejemplo de objeto de visualización: SpriteArranger

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación de ejemplo SpriteArranger se basa en la aplicación de ejemplo de formas geométricas que se describe por separado en *Aprendizaje de ActionScript 3.0*.

La aplicación de ejemplo SpriteArranger ilustra diversos conceptos relacionados con los objetos de visualización:

- Ampliación de las clases de objetos de visualización
- Añadir objetos a la lista de visualización
- Organización en capas de los objetos de visualización y utilización de contenedores de objetos de visualización
- Respuesta a eventos de objetos de visualización
- Uso de propiedades y métodos de objetos de visualización

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación SpriteArranger se encuentran en la carpeta Examples/SpriteArranger. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SpriteArranger.mxml o SpriteArranger.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/SpriteArranger/CircleSprite.as	Una clase que define un tipo de objeto Sprite que representa un círculo en pantalla.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	Una clase que define el lienzo, que es un contenedor de objeto de visualización que contiene objetos GeometricSprite.
com/example/programmingas3/SpriteArranger/SquareSprite.as	Una clase que define un tipo de objeto Sprite que representa un cuadrado en pantalla.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	Una clase que define un tipo de objeto Sprite que representa un triángulo en pantalla.
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Una clase que amplía el objeto Sprite y que se usa para definir una forma en pantalla. CircleSprite, SquareSprite y TriangleSprite amplían esta clase.

Archivo	Descripción
com/example/programmingas3/geometricshapes/IGeometricShape.as	La interfaz base que define los métodos que implementarán todas las clases de formas geométricas.
com/example/programmingas3/geometricshapes/IPolygon.as	Una interfaz que define los métodos que implementarán las clases de formas geométricas que tienen varios lados.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Un tipo de forma geométrica que tiene lados de igual longitud, simétricamente ubicados alrededor del centro de la forma.
com/example/programmingas3/geometricshapes/Circle.as	Un tipo de forma geométrica que define un círculo.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Una subclase de RegularPolygon que define un triángulo con todos los lados de la misma longitud.
com/example/programmingas3/geometricshapes/Square.as	Una subclase de RegularPolygon que define un rectángulo con los cuatro lados de la misma longitud.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Una clase que contiene un "método de fábrica" para crear formas a partir de un tipo y tamaño de forma.

Definición de las clases de SpriteArranger

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación SpriteArranger permite al usuario añadir varios objetos de visualización al "lienzo" en pantalla.

La clase DrawingCanvas define un área de dibujo, un tipo de contenedor de objeto de visualización, en el que el usuario puede añadir formas en pantalla. Estas formas en pantalla son instancias de una de las subclases de la clase GeometricSprite.

La clase DrawingCanvas

En Flex, todos los objetos de visualización secundarios agregados a un objeto Container deben ser de una clase que descienda de la clase mx.core.UIComponent. Esta aplicación añade una instancia de la clase DrawingCanvas como elemento secundario de un objeto mx.containers.VBox, tal y como se define en el código MXML del archivo SpriteArranger.mxml. Esta herencia se define en la declaración de la clase DrawingCanvas del siguiente modo:

```
public class DrawingCanvas extends UIComponent
```

La clase UIComponent hereda de las clases DisplayObject, DisplayObjectContainer y Sprite y el código de la clase DrawingCanvas emplea métodos y propiedades para estas clases.

La clase DrawingCanvas amplía la clase Sprite y su herencia se define en la declaración de la clase DrawingCanvas de la manera siguiente:

```
public class DrawingCanvas extends Sprite
```

La clase Sprite es una subclase de las clases DisplayObjectContainer y DisplayObject, y la clase DrawingCanvas utiliza los métodos y las propiedades de dichas clases.

El método constructor `DrawingCanvas()` configura un objeto Rectangle, `bounds`, que se utiliza para dibujar el contorno del lienzo. A continuación, llama al método `initCanvas()`, del siguiente modo:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Tal y como se muestra en el ejemplo siguiente, el método `initCanvas()` define diversas propiedades del objeto DrawingCanvas que se pasaron como argumentos a la función constructora:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

El método `initCanvas()` llama entonces al método `drawBounds()`, que dibuja el lienzo a través de la propiedad `graphics` de DrawingCanvas. La propiedad `graphics` se hereda de la clase Shape.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

Los siguientes métodos adicionales de la clase DrawingCanvas se llaman en función de las interacciones del usuario con la aplicación:

- Los métodos `addShape()` y `describeChildren()`, que se describen en "Añadir objetos de visualización al lienzo" en la página 213
- Los métodos `moveToBack()`, `moveDown()`, `moveToFront()` y `moveUp()`, descritos en "Reorganización de las capas de los objetos de visualización" en la página 215
- El método `onMouseUp()`, descrito en "Selección mediante clic y arrastre de objetos de visualización" en la página 214

La clase GeometricSprite y sus subclases

Cada objeto de visualización que puede añadir el usuario al lienzo es una instancia de una de las siguientes subclases de la clase GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

La clase GeometricSprite amplía la clase flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

La clase GeometricSprite incluye varias propiedades comunes para todos los objetos GeometricSprite. Estas propiedades se definen en la función constructora según los parámetros pasados a la función. Por ejemplo:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

La propiedad `geometricShape` de la clase GeometricSprite define una interfaz IGeometricShape, que define las propiedades matemáticas de la forma, pero no las propiedades visuales. Las clases que implementan la interfaz IGeometricShape se definen en la aplicación de ejemplo GeometricShapes en *Aprendizaje de ActionScript 3.0*.

La clase GeometricSprite define el método `drawShape()`, que se refina en las definiciones de sustitución de cada una de las subclases de GeometricSprite. Para obtener más información, consulte la siguiente sección "Añadir objetos de visualización al lienzo".

La clase GeometricSprite también proporciona los siguientes métodos:

- Los métodos `onMouseDown()` y `onMouseUp()`, que se describen en "Selección mediante clic y arrastre de objetos de visualización" en la página 214
- Los métodos `showSelected()` y `hideSelected()`, descritos en "Selección mediante clic y arrastre de objetos de visualización" en la página 214

Añadir objetos de visualización al lienzo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando el usuario hace clic en el botón Añadir forma, la aplicación llama al método `addShape()` de la clase DrawingCanvas. Crea una instancia de una nueva clase GeometricSprite mediante una llamada a la función constructora adecuada de una de las subclases de GeometricSprite, como se muestra en el siguiente ejemplo:

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

Cada método constructor llama al método `drawShape()`, que utiliza la propiedad `graphics` de la clase (heredada de la clase Sprite) para dibujar el gráfico vectorial adecuado. Por ejemplo, el método `drawShape()` de la clase CircleSprite incluye el siguiente código:

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

La penúltima línea de la función `addShape()` define la propiedad `alpha` del objeto de visualización (heredada de la clase DisplayObject), de forma que cada objeto de visualización que se añade al lienzo es ligeramente transparente, lo que permite que el usuario vea los objetos que hay detrás.

La línea final del método `addChild()` añade el nuevo objeto de visualización a la lista de elementos secundarios de la instancia de la clase DrawingCanvas, que ya se encuentra en la lista de visualización. Como resultado, el nuevo objeto de visualización aparece en el escenario.

La interfaz de la aplicación incluye dos campos de texto: `selectedSpriteTxt` y `outputTxt`. Las propiedades de texto de estos campos se actualizan con la información relativa a los objetos GeometricSprite que se han añadido al lienzo o que ha seleccionado el usuario. La clase GeometricSprite gestiona esta tarea de notificación de información mediante la sustitución del método `toString()`, del siguiente modo:

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

La propiedad `shapeType` se establece en el valor adecuado en el método constructor de cada subclase de GeometricSprite. Por ejemplo, el método `toString()` podría devolver el siguiente valor de una instancia de CircleSprite recién añadida a la instancia de DrawingCanvas:

```
Circle of size 50 at 0, 0
```

El método `describeChildren()` de la clase DrawingCanvas reproduce indefinidamente la lista de elementos secundarios del lienzo, utilizando la propiedad `numChildren` (heredada de la clase DisplayObjectContainer) para definir el límite del bucle `for`. Genera una cadena con el valor de cada elemento secundario, del siguiente modo:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

La cadena resultante se utiliza para establecer la propiedad `text` del campo de texto `outputTxt`.

Selección mediante clic y arrastre de objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando el usuario hace clic en una instancia de GeometricSprite, la aplicación llama al controlador de eventos `onMouseDown()`. Como se muestra a continuación, se define este controlador de eventos para detectar los eventos de clic del ratón en la función constructora de la clase GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

A continuación, el método `onMouseDown()` llama al método `showSelected()` del objeto GeometricSprite. Si es la primera vez que se ha llamado a este método en el objeto, el método crea un nuevo objeto Shape denominado `selectionIndicator` y utiliza la propiedad `graphics` del objeto Shape para dibujar un rectángulo de resaltado rojo, del siguiente modo:

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

Si no es la primera vez que se llama al método `onMouseDown()`, el método simplemente establece la propiedad `visible` del objeto Shape `selectionIndicator`(heredada de la clase DisplayObject), del siguiente modo:

```
this.selectionIndicator.visible = true;
```

El método `hideSelected()` oculta el objeto Shape `selectionIndicator` del objeto seleccionado previamente estableciendo su propiedad `visible` en `false`.

El método del controlador de eventos `onMouseDown()` también llama al método `startDrag()` (heredado de la clase Sprite), que incluye el siguiente código:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

De esta forma, el usuario puede arrastrar el objeto seleccionado por el lienzo, dentro de los límites definidos por el rectángulo `boundsRect`.

Cuando el usuario suelta el botón del ratón, se distribuye el evento `mouseUp`. El método constructor de DrawingCanvas configura el siguiente detector de eventos:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Este detector de eventos se define en el objeto DrawingCanvas y no en cada uno de los objetos GeometricSprite. El motivo de ello es que, al arrastrar el objeto GeometricSprite, este podría acabar detrás de otro objeto de visualización (otro objeto GeometricSprite) al soltar el botón del ratón. El objeto de visualización en primer plano recibiría el evento de soltar el ratón pero no así el objeto de visualización que arrastra el usuario. Al añadir el detector al objeto DrawingCanvas, se garantiza un control perpetuo del evento.

El método `onMouseUp()` llama al método `onMouseUp()` del objeto GeometricSprite, que a su vez llama al método `stopDrag()` del objeto GeometricSprite.

Reorganización de las capas de los objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La interfaz de usuario de la aplicación incluye botones Move Back, Move Down, Move Up y Move to Front para mover atrás, abajo, arriba y al frente, respectivamente. Cuando el usuario hace clic en uno de estos botones, la aplicación llama al método correspondiente de la clase DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` o `moveToFront()`. Por ejemplo, el método `moveToBack()` incluye el siguiente código:

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

El método utiliza el método `setChildIndex()` (heredado de la clase DisplayObjectContainer) para colocar el objeto de visualización en la posición de índice 0 en la lista de elementos secundarios de la instancia de DrawingCanvas (`this`).

El método `moveDown()` funciona de forma similar, excepto en que reduce en 1 la posición de índice del objeto de visualización en la lista de elementos secundarios de la instancia de DrawingCanvas:

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Los métodos `moveUp()` y `moveToFront()` funcionan de modo similar a los métodos `moveToBack()` y `moveDown()`.

Capítulo 11: Trabajo con la geometría

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El paquete flash.geom contiene clases que definen objetos geométricos, como puntos, rectángulos y matrices de transformación. Estas clases se usan para definir las propiedades de los objetos que se aplican en otras clases.

Más temas de ayuda

flash.geom, paquete

Fundamentos de geometría

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El paquete flash.geom contiene clases que definen objetos geométricos, como puntos, rectángulos y matrices de transformación. Estas clases no siempre proporcionan funcionalidad; sin embargo, se utilizan para definir las propiedades de los objetos que se usan en otras clases.

Todas las clases de geometría se basan en la idea de que las ubicaciones de la pantalla se representan como un plano bidimensional. La pantalla se trata como un gráfico plano con un eje horizontal (x) y un eje vertical (y). Cualquier ubicación (o *punto*) de la pantalla se puede representar como un par de valores x e y (las *coordenadas* de dicha ubicación).

Todos los objetos de visualización, incluyendo el escenario, cuentan con su propio *espacio de coordenadas*. El espacio de coordenadas es un gráfico propio del objeto para representar las ubicaciones de objetos de visualización secundarios, dibujos, etc. El *origen* está en la ubicación de la coordenada 0, 0 (donde coinciden los ejes x e y), y se sitúa en la esquina superior izquierda del objeto de visualización. Aunque esta ubicación de origen siempre es válida para el escenario, no lo es necesariamente para otros objetos de visualización. Los valores del eje x aumentan hacia la derecha y disminuyen hacia la izquierda. En las ubicaciones situadas a la izquierda del origen, la coordenada x es negativa. No obstante, a diferencia de los sistemas de coordenadas tradicionales, los valores de coordenadas del motor de ejecución de Flash del eje y aumentan en la parte inferior de la pantalla y disminuyen en la parte superior. Los valores superiores al origen presentan una coordenada y negativa. Puesto que la esquina superior izquierda del escenario es el origen de su espacio de coordenadas, la mayoría de los objetos del escenario tienen una coordenada x mayor que 0 y menor que la anchura del escenario. Y el mismo objeto cuenta con una coordenada y mayor que 0 y menor que la altura del escenario.

Se pueden utilizar instancias de la clase Point para representar puntos individuales de un espacio de coordenadas. Además, se puede crear una instancia de Rectangle para representar una región rectangular en un espacio de coordenadas. Los usuarios avanzados pueden utilizar una instancia de Matrix para aplicar varias transformaciones o transformaciones complejas a un objeto de visualización. Muchas transformaciones simples, como la rotación o los cambios de posición y de escala se pueden aplicar directamente a un objeto de visualización mediante las propiedades de dicho objeto. Para obtener más información sobre la aplicación de transformaciones con propiedades de objetos de visualización, consulte “Manipulación de objetos de visualización” en la página 178.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos de geometría importantes:

Coordenadas cartesianas Las coordenadas se suelen escribir como un par de números (como 5, 12 o 17, -23). Los dos números son la coordenada x y la coordenada y, respectivamente.

Espacio de coordenadas Gráfico de coordenadas incluido en un objeto de visualización, en el que se sitúan sus elementos secundarios.

Origen Punto de un espacio de coordenadas en el que el eje x converge con el eje y. Este punto tiene las coordenadas 0, 0.

Punto Ubicación individual en un espacio de coordenadas. En el sistema de coordenadas bidimensional utilizado en ActionScript, la ubicación a lo largo de los ejes x e y (coordenadas del punto) define el punto.

Punto de registro En un objeto de visualización, el origen (coordenada 0, 0) de su espacio de coordenadas.

Escala Tamaño de un objeto en relación con su tamaño original. Ajustar la escala de un objeto significa cambiar su tamaño estirándolo o encogiéndolo.

Trasladar Cambiar las coordenadas de un punto de un espacio de coordenadas a otro.

Transformación Ajuste de una característica visual de un gráfico, como girar el objeto, modificar su escala, sesgar o distorsionar su forma, o bien, modificar su color.

Eje X Eje horizontal del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Eje Y Eje vertical del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Uso de objetos Point

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto Point define un par cartesiano de coordenadas. Representa una ubicación en un sistema de coordenadas bidimensional, en el que *x* representa el eje horizontal e *y* representa el eje vertical.

Para definir un objeto Point, se establecen sus propiedades `x` e `y` del siguiente modo:

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

Cálculo de la distancia entre dos puntos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede usar el método `distance()` de la clase Point para calcular la distancia entre dos puntos en un espacio de coordenadas. Por ejemplo, el código siguiente calcula la distancia entre los puntos de registro de dos objetos de visualización, `circle1` y `circle2`, en el mismo contenedor de objetos de visualización:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

Traslación de los espacios de coordenadas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si dos objetos de visualización se encuentran en diferentes contenedores, es posible que se ubiquen en distintos espacios de coordenadas. Se puede usar el método `localToGlobal()` de la clase DisplayObject para trasladar las coordenadas al mismo espacio de coordenadas (global), es decir, el del escenario. Por ejemplo, el código siguiente calcula la distancia entre los puntos de registro de dos objetos de visualización, `circle1` y `circle2`, en diferentes contenedores de objetos de visualización:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Asimismo, para calcular la distancia del punto de registro de un objeto de visualización denominado `target` con respecto a un punto específico del escenario, se puede usar el método `localToGlobal()` de la clase DisplayObject:

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

Desplazamiento de un objeto de visualización con un ángulo y una distancia especificados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede usar el método `polar()` de la clase Point para desplazar un objeto de visualización una distancia específica con un ángulo determinado. Por ejemplo, el código siguiente desplaza el objeto `myDisplayObject` 100 píxeles por 60°:

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Otros usos de la clase Point

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos Point se pueden usar con los métodos y las propiedades siguientes:

Clase	Métodos o propiedades	Descripción
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()` `getObjectsUnderPoint()`	Se usa para devolver una lista de objetos bajo un punto en un contenedor de objetos de visualización.
BitmapData	`hitTest()`	Se usa para definir el píxel en el objeto BitmapData, así como el punto en el que se busca una zona activa.

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Se usa para definir las posiciones de los rectángulos que definen las operaciones.

Matrix	`deltaTransformPoint()` `transformPoint()`	Se usa para definir los puntos para los que se desea aplicar una transformación.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		Se usa para definir estas propiedades.

Uso de objetos Rectangle

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto Rectangle define un área rectangular. Tiene una posición, definida por las coordenadas *x* e *y* de su esquina superior izquierda, una propiedad `width` y una propiedad `height`. Estas propiedades se pueden definir para un objeto Rectangle invocando a la función constructora `Rectangle()`, del siguiente modo:

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Cambio de tamaño y posición de objetos Rectangle

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen varias formas de cambiar el tamaño y la posición de los objetos Rectangle.

Se puede cambiar la posición del objeto Rectangle directamente modificando sus propiedades `x` e `y`. Esta cambio no tiene ningún efecto sobre la anchura o la altura del objeto Rectangle.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Tal y como se muestra en el siguiente código, cuando se cambia la propiedad `left` o `top` de un objeto Rectangle, el rectángulo cambia de posición. Las propiedades `x` e `y` del rectángulo coinciden con las propiedades `left` y `top`, respectivamente. Sin embargo, no se modifica la posición de la esquina inferior izquierda del objeto Rectangle y cambia su tamaño.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

Asimismo, tal como se muestra en el ejemplo siguiente, si se cambia la propiedad `bottom` o `right` del objeto Rectangle, no se verá modificada la posición de su esquina superior izquierda. El rectángulo cambia de tamaño de forma correspondiente:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

También se puede cambiar la posición de un objeto Rectangle con el método `offset()`, del siguiente modo:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

El método `offsetPt()` funciona de un modo similar, con la excepción de que utiliza un objeto Point como parámetro, en lugar de valores de desplazamiento *x* e *y*.

También se puede cambiar el tamaño de un objeto Rectangle con el método `inflate()`, que incluye dos parámetros, `dx` y `dy`. El parámetro `dx` representa el número de píxeles que los lados izquierdo y derecho del rectángulo se desplazan respecto al centro. El parámetro `dy` representa el número de píxeles que la parte superior e inferior del rectángulo se desplazan respecto al centro:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

El método `inflatePt()` funciona de un modo similar, con la excepción de que utiliza un objeto Point como parámetro, en lugar de valores `dx` y `dy`.

Búsqueda de uniones e intersecciones de objetos Rectangle

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `union()` se usa para buscar la región rectangular formada por los límites de dos rectángulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

El método `intersection()` se usa para buscar la región rectangular formada por el área solapada de dos rectángulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

El método `intersects()` se usa para saber si dos rectángulos presentan un punto de intersección. Asimismo, se puede usar el método `intersects()` para conocer si en una región determinada del escenario hay un objeto de visualización. Por ejemplo, en el código siguiente, se considera que el espacio de coordenadas del contenedor de objetos de visualización que incluye el objeto `circle` es el mismo que el del escenario. En el ejemplo se muestra cómo utilizar el método `intersects()` para determinar si un objeto de visualización, `circle`, crea una intersección con regiones especificadas del escenario, definidas por los objetos Rectangle `target1` y `target2`:

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

Asimismo, se puede utilizar el método `intersects()` para saber si se superponen los rectángulos de delimitación de dos objetos de visualización. Se puede utilizar el método `getRect()` de la clase DisplayObject para incluir el espacio adicional que puedan añadir los trazos de un objeto de visualización a la región de delimitación.

Utilización alternativa de objetos Rectangle

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos Rectangle se utilizan en los métodos y las propiedades siguientes:

Clase	Métodos o propiedades	Descripción
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `drawWithQuality()`, `encode()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` y `threshold()`	Se usa como el tipo de algunos parámetros para definir una región del objeto BitmapData.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Se usa como el tipo de datos de la propiedad o el tipo de datos que se devuelve.
PrintJob	`addPage()`	Se usa para definir el parámetro `printArea`.
Sprite	`startDrag()`	Se usa para definir el parámetro `bounds`.
TextField	`getCharBoundaries()`	Se usa como el tipo de valor devuelto.
Transform	`pixelBounds`	Se usa como el tipo de datos.

Uso de objetos Matrix

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Matrix represena una matriz de transformación que determina cómo asignar puntos entre espacios de coordenadas. Es posible realizar varias transformaciones gráficas en un objeto de visualización mediante la definición de las propiedades de un objeto Matrix y aplicar dicho objeto Matrix a la propiedad `matrix` de un objeto Transform. Por último, se puede aplicar dicho objeto Transform como la propiedad `transform` del objeto de visualización. Estas funciones de transformación son la traslación (cambio de posición *x* e *y*), giro, cambio de escala y sesgo.

Aunque se puede definir una matriz ajustando directamente las propiedades (`a`, `b`, `c`, `d`, `tx`, `ty`) de un objeto Matrix, resulta más fácil utilizar el método `createBox()`. Este método incluye parámetros que permiten definir directamente los efectos de ajuste de escala, rotación y traslación del objeto Matrix resultante. Por ejemplo, el código siguiente crea un objeto Matrix que ajusta la escala de un objeto con un factor de 2 horizontalmente y 3 verticalmente, lo gira 45 grados y lo mueve (es decir, lo traslada) 10 píxeles a la derecha y 20 píxeles hacia abajo:

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

También se pueden ajustar los efectos de escala, rotación y traslación de un objeto Matrix con los métodos `scale()`, `rotate()` y `translate()`. Estos métodos se combinan con los valores del objeto Matrix existente. Por ejemplo, en el código siguiente se establece un objeto Matrix que ajusta la escala de un objeto con un factor de 4 y lo gira 60 grados, ya que los métodos `scale()` y `rotate()` se llaman dos veces.

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Para aplicar una transformación de sesgo a un objeto Matrix, debe ajustarse su propiedad `b` o `c`. Al ajustar la propiedad `b`, se sesga la matriz verticalmente y al ajustar la propiedad `c`, se sesga la matriz horizontalmente. En el código siguiente, se sesga el objeto Matrix `myMatrix` verticalmente con un factor de 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Se puede aplicar una transformación de objeto Matrix a la propiedad `transform` de un objeto de visualización. Por ejemplo, el código siguiente aplica una transformación de matriz a un objeto de visualización denominado `myDisplayObject`:

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

La primera línea define un objeto Matrix en la matriz de transformación existente que utiliza el objeto de visualización `myDisplayObject` (la propiedad `matrix` de la propiedad `transformation` del objeto de visualización `myDisplayObject`). De este modo, los métodos de la clase Matrix que se llamen tendrán un efecto acumulativo sobre la posición, la escala y la rotación del objeto de visualización.

Nota: *la clase ColorTransform también está incluida en el paquete flash.geometry. Esta clase se utiliza para establecer la propiedad* `colorTransform` *de un objeto Transform. Puesto que no aplica ningún tipo de transformación geométrica, no se describe aquí detalladamente. Para obtener más información, consulte la clase ColorTransform en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.*

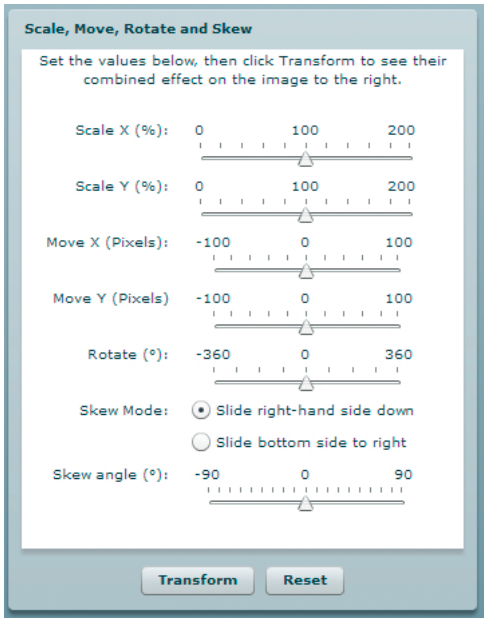
Ejemplo de geometría: Aplicación de una transformación de matriz a un objeto de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación de ejemplo DisplayObjectTransformer muestra varias funciones de la utilización de la clase Matrix para transformar un objeto de visualización, entre las que se incluyen:

- Girar el objeto de visualización
- Ajustar la escala del objeto de visualización
- Trasladar el objeto de visualización (cambiar su posición)
- Sesgar el objeto de visualización

La aplicación proporciona una interfaz para ajustar los parámetros de la transformación de matriz del modo siguiente:



Cuando el usuario hace clic en el botón Transformar, la aplicación aplica la transformación correspondiente.

El objeto de visualización original, y el mismo objeto con una rotación de -45 y un ajuste de escala del 50%

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación DisplayObjectTransformer se encuentran en la carpeta Samples/DisplayObjectTransformer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
DisplayObjectTransformer.mxml o DisplayObjectTransformer.fla	El archivo de aplicación principal en Flash (FLA) o Flex (MXML)
com/example/programmingas3/geometry/MatrixTransformer.as	Una clase que contiene métodos para aplicar transformaciones de matriz.
img/	Un directorio que contiene archivos de imagen de ejemplo que utiliza la aplicación.

Definición de la clase MatrixTransformer

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase MatrixTransformer incluye métodos estáticos que aplican transformaciones geométricas de objetos Matrix.

Método transform()

El método `transform()` incluye parámetros para cada uno de los valores siguientes:

- `sourceMatrix`: matriz de entrada que es transformada por el método.
- `xScale` e `yScale`: factor de escala *x* e *y*.
- `dx` y `dy`: magnitudes de traslación *x* e *y*, representadas en píxeles.
- `rotation`: magnitud de rotación, expresada en grados.
- `skew`: factor de sesgo, como porcentaje.
- `skewType`: dirección del sesgo, ya sea `"right"` (derecha) o `"left"` (izquierda).

El valor devuelto es la matriz resultante.

El método `transform()` llama a los siguientes métodos estáticos de la clase:

- `skew()`
- `scale()`
- `translate()`

- `rotate()`

Cada uno devuelve la matriz de origen con la transformación aplicada.

Método skew()

El método `skew()` sesga la matiz ajustando las propiedades `b` y `c` de la misma. Un parámetro opcional, `unit`, determina las unidades que se utilizan para definir el ángulo de sesgo y, si es necesario, el método convierte el valor `angle` en radianes:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Se crea y se ajusta un objeto `skewMatrix` para aplicar la transformación de sesgo. Inicialmente, es la matriz de identidad:

```
var skewMatrix:Matrix = new Matrix();
```

El parámetro `skewSide` determina el lado en el que se aplica el sesgo. Si se establece en `"right"`, el código siguiente establece la propiedad `b` de la matriz:

```
skewMatrix.b = Math.tan(angle);
```

De lo contrario, el lado inferior se sesga ajustando la propiedad `c` de la matriz, del siguiente modo:

```
skewMatrix.c = Math.tan(angle);
```

El sesgo resultante se aplica a la matriz existente mediante la concatenación de las dos matrices, como se muestra en el ejemplo siguiente:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

Método scale()

Tal y como se muestra en el ejemplo, el método `scale()` ajusta primero el factor de escala si se proporciona como porcentaje y luego utiliza el método `scale()` del objeto de matriz:

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

Método translate()

El método `translate()` aplica simplemente los factores de traslación `dx` y `dy` llamando al método `translate()` del objeto de matriz:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

Método rotate()

El método `rotate()` convierte el factor de rotación de entrada en radianes (si se proporciona en grados o gradientes), y luego llama al método `rotate()` del objeto de matriz:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Llamada al método MatrixTransformer.transform() desde la aplicación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación contiene una interfaz de usuario para obtener los parámetros de transformación del usuario. A continuación transmite dichos valores, junto con la propiedad `matrix` de la propiedad `transform` del objeto de visualización, al método `Matrix.transform()`:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

La aplicación aplica el valor devuelto a la propiedad `matrix` de la propiedad `transform` del objeto de visualización, con lo que se activa la transformación:

```
img.content.transform.matrix = tempMatrix;
```

Capítulo 12: Uso de la API de dibujo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si bien las imágenes e ilustraciones importadas tienen una gran relevancia, la funcionalidad conocida como la API de dibujo, que permite dibujar líneas y formas en ActionScript, ofrece la libertad de iniciar una aplicación con el equipo en lo que sería el equivalente a un lienzo en blanco en el que se pueden crear las imágenes que se deseen. La capacidad de crear gráficos propios abre un amplísimo abanico de posibilidades para las aplicaciones. Gracias a las técnicas que se tratarán en este capítulo, es posible crear un programa de dibujo, diseñar obras de arte animadas e interactivas o desarrollar mediante programación elementos de interfaz propios, entre muchas otras posibilidades.

Más temas de ayuda

flash.display.Graphics

Aspectos básicos de la API de dibujo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La funcionalidad integrada en ActionScript que permite crear gráficos vectoriales (líneas, curvas, formas, rellenos y degradados) y mostrarlos en la pantalla mediante ActionScript recibe el nombre de API de dibujo. La clase flash.display.Graphics es la encargada de proporcionar esta funcionalidad. Con ActionScript es posible dibujar en cualquier instancia de Shape, Sprite o MovieClip utilizando la propiedad `graphics` definida en cada una de esas clases. (La propiedad `graphics` de cada una de esas clases es en realidad una instancia de la clase Graphics.)

A la hora de empezar a dibujar mediante código, la clase Graphics ofrece varios métodos que facilitan el trazado de formas comunes, como círculos, elipses, rectángulos y rectángulos con esquinas redondeadas. Todas ellas se pueden dibujar como líneas vacías o como formas rellenas. Cuando sea necesaria una funcionalidad más avanzada, la clase Graphics también dispone de métodos para trazar líneas y curvas cuadráticas de Bézier, que se pueden emplear junto con las funciones trigonométricas de la clase Math para crear cualquier forma imaginable.

Los motores de ejecución de Flash (como Flash Player 10 y Adobe AIR 1.5 y versiones posteriores) incorporan una API adicional para dibujo, que permite dibujar mediante programación formas completas con un único comando. Un vez que se haya familiarizado con la clase Graphics y las tareas incluidas en "Fundamentos de la utilización de la API de dibujo", continúe con "Utilización avanzada de la API de dibujo" en la página 243 para obtener más información sobre estas funciones de la API de dibujo.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán mientras se utiliza la API de dibujo:

Punto de anclaje Uno de los dos puntos finales de una curva cuadrática de Bézier.

Punto de control Punto que define la dirección y la curvatura de una curva cuadrática de Bézier. La línea curva nunca alcanza el punto de control, no obstante, se curva como si fuese arrastrada hacia este.

Espacio de coordenadas Gráfico de coordenadas incluido en un objeto de visualización, en el que se sitúan sus elementos secundarios.

Relleno Parte interior sólida de una forma que tiene una línea rellena con color o la totalidad de una forma que no tiene contorno.

Degradado Color que está formado por la transición gradual de un color a otro u otros (a diferencia de un color sólido).

Punto Ubicación individual en un espacio de coordenadas. En el sistema de coordenadas bidimensional utilizado en ActionScript, el punto se define por su ubicación en el eje x y en el eje y (las coordenadas del punto).

Curva cuadrática de Bézier Tipo de curva definida por una fórmula matemática concreta. En este tipo de curva, la forma se calcula basándose en las posiciones de los puntos de ancla (los extremos de la curva) y el punto de control que define la dirección y la curvatura del trazo.

Escala Tamaño de un objeto en relación con su tamaño original. Ajustar la escala de un objeto significa cambiar su tamaño estirándolo o encogiéndolo.

Trazo Parte del contorno de una forma que tiene una línea rellena con un color o las líneas de una forma sin relleno.

Trasladar Cambiar las coordenadas de un punto de un espacio de coordenadas a otro.

Eje X Eje horizontal del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Eje Y Eje vertical del sistema de coordenadas bidimensional que se utiliza en ActionScript.

Clase Graphics

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todos los objetos Shape, Sprite y MovieClip tienen una propiedad `graphics`, que es una instancia de la clase Graphics. La clase Graphics incluye propiedades y métodos para dibujar líneas, rellenos y formas. Cuando se necesita un objeto de visualización que se va a utilizar únicamente como un lienzo para dibujar contenido, se puede utilizar una instancia de Shape. Las instancias de Shape funcionan mejor que otros objetos de visualización a la hora de dibujar porque no tienen la sobrecarga de la funcionalidad adicional con la que cuentan las clases Sprite y MovieClip. En caso de que se desee un objeto de visualización en el que se pueda dibujar contenido gráfico y que, al mismo tiempo, pueda contener otros objetos de visualización, se puede utilizar una instancia de Sprite. Para más información sobre la forma de determinar qué objeto de visualización debe usarse para las distintas tareas, consulte "Selección de una subclase DisplayObject" en la página 177.

Dibujo de líneas y curvas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todos los dibujos que se pueden realizar con una instancia de Graphics se basan en trazados básicos con líneas y curvas. En consecuencia, todos los dibujos con ActionScript deben llevarse a cabo utilizando la misma serie de pasos:

- Definir los estilos de línea y relleno.
- Establecer la posición inicial de dibujo.
- Dibujar líneas, curvas y formas (moviendo opcionalmente el punto de dibujo).
- En caso de ser necesario, finalizar creando un relleno.

Definición de los estilos de línea y relleno

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para dibujar con la propiedad `graphics` de una instancia de Shape, Sprite o MovieClip, en primer lugar es necesario definir el estilo (tamaño y color de línea, color de relleno) que se utilizará al dibujar. Al igual que cuando se utilizan las herramientas de dibujo de Adobe® Flash® Professional u otra aplicación de dibujo, al utilizar ActionScript se puede dibujar con o sin trazo y con o sin color de relleno. La apariencia del trazo se especifica mediante los métodos `lineStyle()` o `lineGradientStyle()`. Para crear una línea sólida se emplea el método `lineStyle()`. Al llamar a este método, los valores más comunes que se especificarán son los primeros tres parámetros: grosor de línea, color y alfa. Por ejemplo, la siguiente línea de código indica al objeto Shape llamado `myShape` que dibuje líneas de 2 píxeles de grosor, de color rojo (0x990000) y de una opacidad del 75%:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

El valor predeterminado del parámetro alfa es de 1,0 (100%), de modo que se puede omitir el parámetro si se desea una línea completamente opaca. El método `lineStyle()` también acepta dos parámetros más para las sugerencias de píxeles y el modo de escala. Para obtener más información sobre el uso de estos parámetros, consulte la descripción del método `Graphics.lineStyle()` en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Para crear una línea con gradiente se emplea el método `lineGradientStyle()`. Este método se describe en "Creación de líneas y rellenos degradados" en la página 234.

Para crear una forma con relleno es necesario llamar a los métodos `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` o `beginShaderFill()` antes de empezar a dibujar. El más básico de ellos, el método `beginFill()`, acepta dos parámetros: el color de relleno y (opcionalmente) un valor de alfa para el color de relleno. Por ejemplo, para dibujar una forma con un relleno verde sólido, se puede usar el siguiente código (suponiendo que se vaya a dibujar un objeto llamado `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

Llamar a cualquier método de relleno termina de forma implícita cualquier relleno antes de iniciar uno nuevo. Llamar a cualquier método que especifique un estilo de trazo sustituye al trazo anterior, pero no modifica el relleno especificado previamente (y viceversa).

Una vez indicado el estilo de línea y las propiedades de relleno, el siguiente paso es especificar el punto de inicio del dibujo. La instancia de Graphics tiene un punto de dibujo, como la punta de un lápiz sobre la superficie de un papel. La siguiente acción de dibujo empezará allá donde se encuentre el punto de dibujo. Inicialmente, los objetos Graphics tienen su punto de dibujo en el punto 0,0 del espacio de coordenadas del objeto en el que se está dibujando. Para empezar a dibujar en un punto diferente es necesario llamar primero al método `moveTo()` antes de llamar a cualquiera de los métodos de dibujo. Esto es análogo a levantar la punta del lápiz del papel y moverla a una nueva posición.

Una vez que el punto de dibujo está colocado en su lugar, se empieza a dibujar utilizando una serie de llamadas a los métodos de dibujo `lineTo()` (para dibujar líneas rectas) y `curveTo()` (para dibujar curvas).

Mientras se dibuja es posible llamar al método `moveTo()` en cualquier momento para mover el punto de dibujo a una nueva posición sin dibujar.

Durante el dibujo, si se ha especificado un color de relleno, es posible cerrar el relleno llamando al método `endFill()`. Si no se ha dibujado una forma cerrada (dicho de otro modo, si en el momento en que se llama a `endFill()` el punto de dibujo no se encuentra en el punto de inicio de la forma), al llamar al método `endFill()`, el motor de ejecución de Flash cerrará automáticamente la forma dibujando una línea recta desde el punto de dibujo actual hasta la ubicación especificada en la llamada a `moveTo()` más reciente. Si se ha iniciado un relleno y no se ha llamado a `endFill()`, al llamar a `beginFill()` (o a cualquier otro método de relleno) se cerrará el relleno actual y se iniciará uno nuevo.

Dibujo de líneas rectas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al llamar al método `lineTo()`, el objeto Graphics dibuja una línea recta desde el punto de dibujo actual hasta las coordenadas especificadas en los dos parámetros de la llamada al método usando el estilo de línea que se haya indicado. Por ejemplo, la siguiente línea de código coloca el punto de dibujo en las coordenadas 100, 100 y, a continuación, dibuja una línea hasta el punto 200, 200:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

El siguiente ejemplo dibuja triángulos de color rojo y verde con una altura de 100 píxeles:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

Dibujo de curvas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `curveTo()` dibuja una curva cuadrática de Bézier, es decir, un arco que conecta dos puntos (llamados puntos de ancla) y se inclina hacia un tercero (llamado punto de control). El objeto Graphics utiliza la posición de dibujo actual como primer punto de ancla. Al llamar al método `curveTo()`, se pasan cuatro parámetros: las coordenadas x e y, seguidas por las coordenadas x e y del segundo punto de ancla. Por ejemplo, el siguiente código dibuja una curva que empieza en el punto 100, 100 y termina en el punto 200, 200. Dado que el punto de control se encuentra en 175, 125, la curva que se genera se mueve hacia la derecha y luego hacia abajo:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

El siguiente ejemplo dibuja objetos circulares rojos y verdes con una altura y una anchura de 100 píxeles. Hay que tener en cuenta que, debido a la naturaleza de la ecuación cuadrática de Bézier, no son círculos perfectos:

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

Dibujo de formas mediante los métodos incorporados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Por comodidad, a la hora de dibujar formas comunes, como círculos, elipses, rectángulos y rectángulos con esquinas redondeadas, ActionScript 3.0 dispone de métodos para dibujar esas formas automáticamente. Estos son los métodos `drawCircle()`, `drawEllipse()`, `drawRect()` y `drawRoundRect()` de la clase Graphics. Estos métodos se pueden usar en lugar de `lineTo()` y `curveTo()`. Sin embargo, hay que tener en cuenta que sigue siendo necesario especificar los estilos de línea y relleno antes de llamar a estos métodos.

A continuación, se vuelve a usar el ejemplo del dibujo de cuadrados rojos, verdes y azules con una altura y una anchura de 100 píxeles. Este código utiliza el método `drawRect()` y, además, especifica que el color de relleno tiene un valor alfa del 50% (0.5):

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

En un objeto Sprite o MovieClip, el contenido de dibujo creado con la propiedad `graphics` siempre aparece detrás de todos los objetos de visualización secundarios contenidos en el objeto. Además, el contenido de la propiedad `graphics` no es un objeto de visualización independiente, de modo que no aparece en la lista de un elemento secundario de un objeto Sprite o MovieClip. Por ejemplo, el siguiente objeto Sprite tiene un círculo dibujado con su propiedad `graphics` y un objeto TextField en su lista de objetos de visualización secundarios:

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Hay que tener en cuenta que el objeto TextField aparece sobre el círculo dibujado con el objeto graphics.

Creación de líneas y rellenos degradados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El objeto graphics también puede dibujar trazos y rellenos con degradados en lugar de con colores sólidos. Los trazos con degradado se crean mediante el método `lineGradientStyle()`, mientras que para los rellenos degradados se usa el método `beginGradientFill()`.

Ambos métodos aceptan los mismos parámetros. Los cuatro primeros son necesarios: tipo, colores, alfas y proporciones. Los cuatro restantes son opcionales, pero resultan útiles para llevar a cabo una personalización avanzada.

- El primer parámetro especifica el tipo de degradado que se va a crear. Los valores aceptables son `GradientType.LINEAR` o `GradientType.RADIAL`.
- El segundo parámetro especifica el conjunto de valores de color que se usará. En un degradado lineal, los colores se ordenarán de izquierda a derecha. En un degradado radial, se organizarán de dentro a fuera. El orden de los colores del conjunto representa el orden en el que los colores se dibujarán en el degradado.
- El tercer parámetro especifica los valores de transparencia alfa de los colores correspondientes al parámetro anterior.
- El cuarto parámetro especifica las proporciones, o el énfasis que cada color tiene dentro del gradiente. Los valores válidos se encuentran entre 0 y 255. Estos valores no representan una altura o anchura, sino una posición dentro del degradado; 0 representa el inicio del degradado y 255 el final. El conjunto de proporciones debe aumentar secuencialmente y tener el mismo número de entradas que los conjuntos de color y de alfa especificados en el segundo y tercer parámetros.

Si bien el quinto parámetro, la matriz de transformación, es opcional, se suele utilizar con frecuencia, ya que constituye una forma fácil y potente de controlar la apariencia del degradado. Este parámetro acepta una instancia de Matrix. La forma más sencilla de crear un objeto Matrix para el degradado es usar el método `createGradientBox()` de la clase Matrix.

Definición de un objeto Matrix para su utilización con un degradado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `beginGradientFill()` y `lineGradientStyle()` de la clase flash.display.Graphics se utilizan para definir los degradados que se utilizarán en las formas. Cuando se define un degradado, se proporciona una matriz como uno de los parámetros de estos métodos.

La forma más fácil de definir la matriz consiste en utilizar el método `createGradientBox()` de la clase Matrix, que crea una matriz que se usa para definir el degradado. La escala, la rotación y la posición del degradado se definen con los parámetros que se pasan al método `createGradientBox()`. El método `createGradientBox()` acepta los siguientes parámetros:

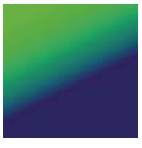
- Anchura del cuadro del degradado: la anchura (en píxeles) en que se extenderá el degradado
- Altura del cuadro del degradado: la altura (en píxeles) sobre la que se extenderá el degradado
- Rotación del cuadro de degradado: la rotación (en radianes) que se aplicará al degradado
- Traslación horizontal: el desplazamiento horizontal (en píxeles) que se aplicará al degradado
- Traslación vertical: el desplazamiento vertical (en píxeles) que se aplicará al degradado

Por ejemplo, en el caso de un degradado con las características siguientes:

- `GradientType.LINEAR`
- Dos colores, verde y azul, con el conjunto `ratios` establecido en `[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

En los ejemplos siguientes, se muestran los degradados en los que el parámetro `rotation` del método `createGradientBox()` difiere del modo indicado, pero el resto de valores no cambia:

`width = 100;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`	

`width = 100;`
`height = 100;`
`rotation = Math.PI/4; // 45°`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 100;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`	

En el ejemplo siguiente se muestran los efectos sobre un degradado lineal de verde a azul, en el que los parámetros `rotation`, `tx` y `ty` del método `createGradientBox()` difieren del modo indicado, pero el resto de valores no cambia:

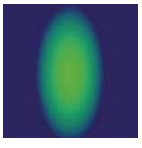
`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`	

`width = 50;`
`height = 100;`
`rotation = 0`
`tx = 50;`
`ty = 0;`	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 50;`	

Los parámetros `width`, `height`, `tx` y `ty` del método `createGradientBox()` también afectan al tamaño y la posición de un relleno con degradado *radial*, tal como se muestra en el ejemplo siguiente:

`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 25;`
`ty = 0;`
	

En el siguiente código se muestra el último degradado radial ilustrado:

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                           colors,
                           alphas,
                           ratios,
                           matrix,
                           spreadMethod,
                           interp,
                           focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

No hay que olvidar que la anchura y altura del relleno degradado vienen dadas por la anchura y la altura de la matriz de degradado en lugar de por la anchura o la altura dibujadas usando el objeto Graphics. Al dibujar con el objeto Graphics, se dibuja lo que existe en esas coordenadas en la matriz de degradado. Incluso si se usa uno de los métodos de forma de un objeto Graphics, como `drawRect()`, el degradado no se estira hasta adaptarse al tamaño de la forma dibujada, sino que el tamaño del degradado debe especificarse en la propia matriz de degradado.

El siguiente ejemplo ilustra la diferencia visual entre las dimensiones de la matriz de degradado y las dimensiones del dibujo:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

Este código dibuja tres degradados con el mismo estilo de relleno especificado una distribución idéntica de rojo, verde y azul. Los degradados se dibujan utilizando el método `drawRect()` con anchuras de 50, 100 y 150 píxeles respectivamente. La matriz de degradado especificada en el método `beginGradientFill()` se crea con una anchura de 100 píxeles. Esto quiere decir que el primer degradado abarcará solo la mitad del espectro del degradado, el segundo lo englobará totalmente y el tercero lo incluirá por completo y además tendrá 50 píxeles más de azul extendiéndose hacia la derecha.

El método `lineGradientStyle()` funciona de forma similar a `beginGradientFill()` excepto en que, además de definir el degradado, es necesario especificar el grosor del trazo utilizando el método `lineStyle()` antes de dibujar. El siguiente código dibuja un cuadrado con un trazo degradado rojo, verde y azul:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Para más información sobre la clase Matrix, consulte “Uso de objetos Matrix” en la página 223.

Uso de la clase Math con los métodos de dibujo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos Graphics pueden dibujar círculos y cuadrados, aunque también pueden trazar formas más complejas, en especial cuando los métodos de dibujo se utilizan junto con las propiedades y métodos de la clase Math. La clase Math contiene constantes comunes de tipo matemático, como `Math.PI` (aproximadamente 3,14159265...), una constante que expresa la proporción entre el perímetro de un círculo y su diámetro. Asimismo, contiene métodos de funciones trigonométricas, como `Math.sin()`, `Math.cos()` y `Math.tan()` entre otros. Cuando se dibujan formas utilizando estos métodos y constantes se crean efectos visuales de mayor dinamismo, especialmente si se usan junto con mecanismos de repetición o recursión.

En muchos métodos de la clase Math es necesario que las unidades empleadas en las medidas circulares sean radianes en lugar de grados. La conversión entre estos dos tipos de unidades es uno de los usos más frecuentes de la clase Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

El siguiente ejemplo crea una onda de seno y una onda de coseno para poner de manifiesto la diferencia entre los métodos `Math.sin()` y `Math.cos()` para un valor dado.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

Animación con la API de dibujo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las ventajas de crear contenido con la API de dibujo es que no existe la limitación de colocar el contenido una sola vez. Los elementos dibujados se pueden modificar, conservando y modificando las variables utilizadas para dibujar. Se puede lograr un efecto de animación cambiando las variables y redibujando, ya sea a lo largo de una serie de fotogramas o mediante un temporizador.

Por ejemplo, el siguiente código modifica la pantalla con cada fotograma que pasa (detectando el evento `Event.ENTER_FRAME`), incrementa el número de grados e indica al objeto de Graphics que debe limpiar la pantalla y redibujarla con la posición actualizada.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

Se puede experimentar modificando los valores de inicialización de las variables `currentDegrees`, `radius` y `satelliteRadius` al comienzo del código para lograr unos resultados completamente diferentes. Por ejemplo, se puede reducir la variable radius o incrementar la variable totalSatellites. Esto no es más que un ejemplo de la forma en la que la API de dibujo puede crear una presentación visual cuya complejidad oculta la sencillez de su creación.

Ejemplo de API de dibujo: Generador visual algorítmico

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo del generador visual algorítmico dibuja dinámicamente en el escenario varios "satélites", o círculos que se mueven en una órbita circular. Algunas de las funciones exploradas son:

- Utilizar la API de dibujo para dibujar una forma básica con apariencias dinámicas
- Conectar la interacción del usuario con las propiedades que se usan en un dibujo
- Producir una animación borrando y redibujando el escenario en cada fotograma

El ejemplo de la subsección anterior animaba un "satelite" solitario mediante el evento `Event.ENTER_FRAME`. En este ejemplo se amplía el ejemplo anterior creando un panel de control con varios controles deslizantes que actualizan inmediatamente la presentación visual de varios satélites. En este ejemplo se formaliza el código en clases externas y se rodea con un bucle el código de creación del satélite, almacenando una referencia a cada satélite en un conjunto `satellites`.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación se encuentran en la carpeta Samples/AlgorithmicVisualGenerator. Esta carpeta contiene los siguientes archivos:

Archivo	Descripción
AlgorithmicVisualGenerator.fla	El archivo principal de la aplicación en Flash (FLA).
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	La clase que proporciona la funcionalidad principal de la aplicación, incluido el dibujo de satélites en el escenario y la respuesta a los eventos del panel de control para actualizar las variables que afectan al dibujo de satélites.
com/example/programmingas3/algorithmic/ControlPanel.as	Una clase que gestiona la interacción con el usuario mediante varios controles deslizantes y distribuye eventos cuando esto ocurre.
com/example/programmingas3/algorithmic/Satellite.as	Una clase que representa al objeto de visualización que gira en una órbita alrededor de un punto central y contiene propiedades relacionadas con su estado de dibujo actual.

Configuración de los detectores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En primer lugar, la aplicación crea tres detectores. El primero de ellos detecta un evento distribuido desde el panel de control que hace que resulte necesario volver a crear los satélites. El segundo detecta los cambios en el tamaño del escenario del archivo SWF. El tercero detecta el paso de los fotogramas en el archivo SWF y redibuja la pantalla utilizando la función `doEveryFrame()`.

Creación de los satélites

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una vez configurados esos detectores, se llama a la función `build()`. Esta función llama en primer lugar a la función `clear()`, que vacía el conjunto `satellites` y borra todo lo que hubiera dibujado anteriormente en el escenario. Esto es necesario, ya que la función `build()` puede volverse a llamar siempre que el panel de control envíe un evento para ello como, por ejemplo, cuando se modifique la configuración del color. En tal caso es necesario eliminar y volver a crear los satélites.

A continuación, la función crea los satélites, configurando las propiedades iniciales necesarias para su creación, como la variable `position`, que se inicia en una posición aleatoria de la órbita, y la variable `color`, que en este ejemplo no cambia una vez creado el satélite.

A medida que se crea cada satélite, se añade una referencia a ellos en el conjunto `satellites`. Cuando se llame a la función `doEveryFrame()`, se actualizarán todos los satélites de este conjunto.

Actualización de la posición de los satélites

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La función `doEveryFrame()` es el núcleo del proceso de animación de la aplicación. En cada fotograma se llama a esta función, a una velocidad igual a la velocidad de fotogramas del archivo SWF. Dado que las variables del dibujo varían ligeramente, se produce el efecto de animación.

En primer lugar, la función borra todo lo que hubiese dibujado previamente y redibuja el fondo. A continuación, el bucle itera para cada contenedor de satélite e incrementa la propiedad `position` de cada satélite y actualiza las propiedades `radius` y `orbitRadius`, que pueden haber cambiado a causa de la interacción del usuario con el panel de control. Finalmente, el satélite se actualiza y se coloca en su nueva posición llamando al método `draw()` de la clase Satellite.

Hay que tener presente que el contador, `i`, solo se incrementa hasta la variable `visibleSatellites`. Esto se debe a que, si el usuario ha limitado la cantidad de satélites que se muestran en el panel de control, los satélites restantes del bucle no se muestran, sino que se ocultan. Esto ocurre en un bucle que sigue inmediatamente al bucle encargado del dibujo.

Cuando se completa la función doEveryFrame(), el número de `visibleSatellites` se actualiza en posición a lo largo de la pantalla.

Respuesta a la interacción con el usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La interacción con el usuario se produce a través del panel de control, que gestiona la clase ControlPanel. Esta clase define un detector, junto con los valores mínimos, máximos y predeterminados individuales de cada control deslizante. Cuando el usuario mueve alguno de los controles deslizantes, se llama a la función `changeSetting()`. Esta función actualiza las propiedades del panel de control. Si el cambio requiere que se redibuje la pantalla, se envía un evento que se gestiona en el archivo principal de la aplicación. Cuando cambia la configuración del panel de control, la función `doEveryFrame()` dibuja cada uno de los satélites con las variables actualizadas.

Personalización avanzada

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Este ejemplo constituye tan solo un esquema básico de la forma en que se pueden generar efectos visuales mediante la API de dibujo. Emplea relativamente pocas líneas de código para crear una experiencia interactiva que, en apariencia, es muy compleja y, aún así, este ejemplo puede ampliarse mediante pequeños cambios. A continuación se exponen algunas ideas:

- La función `doEveryFrame()` puede incrementar el valor del color de los satélites.
- La función `doEveryFrame()` puede reducir o aumentar el radio del satélite a lo largo del tiempo.
- El radio del satélite no tiene por qué ser circular. Se puede usar la clase Math para se muevan siguiendo, por ejemplo, una onda sinusoidal.
- Los satélites pueden emplear un mecanismo de detección de colisiones con otros satélites.

Se puede utilizar la API de dibujo como alternativa para crear efectos visuales en el entorno de edición de Flash, dibujando formas básicas en tiempo de ejecución. Asimismo, se puede usar para crear efectos visuales de una variedad y extensión imposibles de alcanzar manualmente. Mediante la API de dibujo y un poco de matemáticas, el autor puede dar vida a un sinnúmero de creaciones sorprendentes.

Utilización avanzada de la API de dibujo

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Flash Player 10, Adobe AIR 1.5 y motores de ejecución de Flash posteriores admiten un conjunto avanzado de funciones de dibujo. Las mejoras de la API de dibujo amplían los métodos de dibujo de las versiones anteriores, por lo que se pueden establecer conjuntos de datos para generar formas, modificarlas en tiempo de ejecución y crear efectos tridimensionales. Las mejoras de la API de dibujo consolidan los métodos existentes como comandos alternativos. Estos comandos aprovechan los conjuntos vectoriales y las clases de enumeración para ofrecer conjuntos de datos para métodos de dibujo. El uso de conjuntos vectoriales permite que las formas más complejas se representen con rapidez y que los desarrolladores puedan modificar los valores del conjunto mediante programación para la representación de formas dinámicas en tiempo de ejecución.

Las funciones de dibujo incluidas en Flash Player 10 se describen en las siguientes secciones: "Trazados de dibujo" en la página 244, "Definición de reglas de trazado" en la página 245, "Uso de clases de datos gráficos" en la página 247 y "Uso de drawTriangles()" en la página 250.

A continuación se enumeran tareas habituales que se realizan con la API de dibujo avanzado de ActionScript:

- Uso de objetos Vector para almacenar datos para métodos de dibujo.
- Definición de trazados para dibujar formas mediante programación en una sola operación.
- Definición de reglas de dirección de trazado para determinar cómo se rellenan las formas superpuestas.
- Lectura de contenido de gráficos vectoriales de un objeto de visualización para, por ejemplo, serializar y guardar datos de gráficos, generar una hoja de Sprite en tiempo de ejecución o dibujar una copia del contenido de gráficos vectoriales
- Uso de triángulos y métodos de dibujo para efectos tridimensionales.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que se utilizan en esta sección:

- Vector: conjunto de valores del mismo tipo de datos. Un objeto Vector puede almacenar un conjunto de valores que los métodos de dibujo utilizan para crear líneas y formas con un solo comando. Para obtener más información sobre los objetos Vector, consulte "Conjuntos indexados" en la página 27.
- Trazado: un trazado se compone de uno o varios segmentos rectos o curvos. El principio y el final de cada segmento están marcados con coordenadas, que funcionan como grapas que mantienen un cable en su lugar. Un trazado puede estar cerrado (por ejemplo, un círculo), o abierto, con marcados puntos finales (por ejemplo, una línea ondulante).
- Dirección de trazado: la dirección de un trazado interpretada por el procesador; positiva (en el sentido de las agujas del reloj) o negativa (en el sentido contrario a las agujas del reloj).
- GraphicsStroke: clase para definir el estilo de línea. Aunque el "trazo" no se incluye entre las mejoras de la API de dibujo, el uso de una clase para designar un estilo de línea con su propia propiedad de relleno sí supone una mejora. Se puede ajustar dinámicamente un estilo de línea utilizando la clase GraphicsStroke.
- Objetos Fill: objetos creados utilizando clases de visualización como flash.display.GraphicsBitmapFill y flash.display.GraphicsGradientFill, que se transmiten al comando de dibujo `Graphics.drawGraphicsData()`. Los objetos Fill y los comandos de dibujo mejorados introducen un enfoque de programación más orientado a objetos en la réplica de `Graphics.beginBitmapFill()` y `Graphics.beginGradientFill()`.

Trazados de dibujo

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La sección sobre dibujo de líneas y curvas (consulte "Dibujo de líneas y curvas" en la página 230) introdujo los comandos para dibujar una sola línea (`Graphics.lineTo()`) o curva (`Graphics.curveTo()`) y posteriormente mover la línea a otro punto (`Graphics.moveTo()`) para realizar una forma. Los métodos `Graphics.drawPath()` y `Graphics.drawTriangles()` aceptan un conjunto de objetos que representan los mismos comandos de dibujo que un parámetro. Con estos métodos se puede proporcionar una serie de comandos `Graphics.lineTo()`, `Graphics.curveTo()` o `Graphics.moveTo()` para que el motor de ejecución de Flash los ejecute en una sola sentencia.

La clase de enumeración GraphicsPathCommand define una serie de constantes que corresponden a comandos de dibujo. El usuario pasa una serie de estas constantes (agrupadas en una instancia de Vector) como parámetro para el método `Graphics.drawPath()`. Posteriormente con un solo comando puede representar una forma completa o varias formas. También puede modificar los valores transmitidos a estos métodos para modificar una forma existente.

Además del Vector de comandos de dibujo, el método `drawPath()` requiere una serie de coordenadas correspondientes a las de cada comando de dibujo. Cree una instancia de Vector que contenga coordenadas (instancias Number) y pásela al método `drawPath()` como segundo argumento (`data`).

Nota: *los valores en el vector no son objetos Point; el vector es una serie de números donde cada grupo de dos números representa un par de coordenadas x/y.*

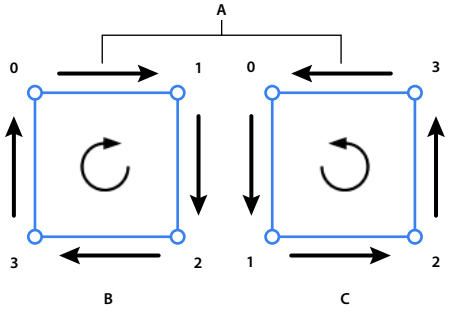
El método `Graphics.drawPath()` hace coincidir cada comando con sus respectivos valores de coma (un conjunto de dos o cuatro números) para generar un trazado en el objeto Graphics:

```
package
{
    import flash.display.*;

    public class DrawPathExample extends Sprite
    {
        public function DrawPathExample(){

            var squareCommands:Vector.<int> = new Vector.<int>(5, true);
            squareCommands[0] = GraphicsPathCommand.MOVE_TO;
            squareCommands[1] = GraphicsPathCommand.LINE_TO;
            squareCommands[2] = GraphicsPathCommand.LINE_TO;
            squareCommands[3] = GraphicsPathCommand.LINE_TO;
            squareCommands[4] = GraphicsPathCommand.LINE_TO;

            var squareCoord:Vector.<Number> = new Vector.<Number>(10, true);
            squareCoord[0] = 20; //x
            squareCoord[1] = 10; //y
            squareCoord[2] = 50;
            squareCoord[3] = 10;
            squareCoord[4] = 50;
            squareCoord[5] = 40;
            squareCoord[6] = 20;
            squareCoord[7] = 40;
            squareCoord[8] = 20;
            squareCoord[9] = 10;

            graphics.beginFill(0x442266);//set the color
            graphics.drawPath(squareCommands, squareCoord);
        }
    }
}
```

Definición de reglas de trazado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La API de dibujo mejorada también introduce el concepto de "dirección de trazado". La dirección de trazado es positiva (en el sentido de las agujas del reloj) o negativa (en el sentido contrario a las agujas del reloj). El orden en el que el procesador interpreta las coordenadas que proporciona el vector para el parámetro de datos determina la dirección de trazado.



Dirección de trazado positivo y negativo
A. *Las flechas indican la dirección de trazado* ***B.*** *Dirección de trazado de curva positiva (en el sentido de las agujas del reloj)* ***C.*** *Dirección de trazado de curva negativa (en el sentido contrario a las agujas del reloj)*

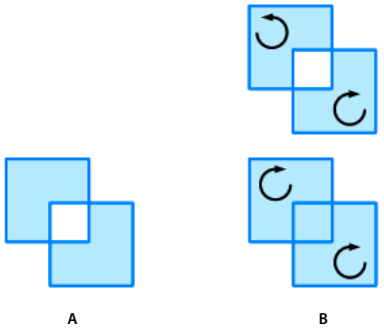
Asimismo, se debe tener en cuenta que el método `Graphics.drawPath()` tiene un tercer parámetro opcional denominado “winding”:

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

En este contexto, el tercer parámetro es una cadena o una constante que especifica la dirección de trazado o regla de relleno para los trazados de intersección. (Los valores de la constante se definen en la clase GraphicsPathWinding como`GraphicsPathWinding.EVEN_ODD` o `GraphicsPathWinding.NON_ZERO`.) La regla de dirección de trazado es importante cuando los trazados se intersecan.

La regla par-impar es la regla de dirección de trazado estándar y se utiliza en la API de dibujo heredada. La regla par-impar también es la predeterminada para el método `Graphics.drawPath()`. Con la regla de dirección de trazado par-impar, cualquier trazado de intersección alterna entre rellenos abiertos y cerrados. Si dos cuadrados dibujados con el mismo relleno intersecan, el área de intersección se rellena. Generalmente, las áreas adyacentes no están ambas rellenas ni vacías.

Por otra parte, la regla distinta a cero depende de la dirección de trazado(dirección de dibujo) para determinar si se rellenan las áreas definidas mediante trazados de intersección. Cuando los trazados de dirección opuesta se intersecan, el área definida no se rellena, al igual que con la regla par-impar. Para los trazados con la misma dirección, el área que quedaría vacía se rellena:

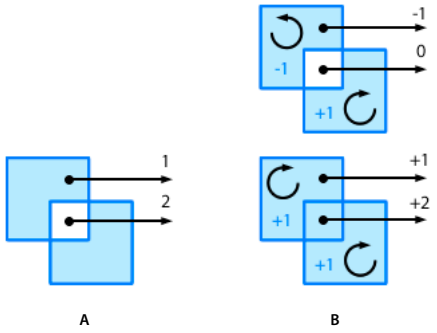


Reglas de dirección de trazado para áreas de intersección
A. *Regla de dirección de trazado par-impar* ***B.*** *Regla de dirección de trazado distinta a cero*

Nombres de reglas de dirección de trazado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Los nombres hacen referencia a una regla más específica que define el modo en que se administran estos rellenos. Los trazados de curva positiva se asignan como un valor de +1; los trazados de curva negativa se asignan con un valor de -1. Comenzando desde un punto en un área cerrada de una forma, dibuje una línea desde ese punto y prolónguela indefinidamente. El número de veces que la línea cruza un trazado y los valores combinados de esos trazados se utilizan para determinar el relleno. Para la dirección de trazado par-impar, se utiliza el recuento de veces que la línea cruza un trazado. Si el recuento es impar, el área se rellena. Para los recuentos pares, el área no se rellena. Para la dirección de trazado distinta a cero, se utilizan los valores asignados a los trazados. Cuando los valores combinados del trazado son distintos a 0, el área se rellena. Si el valor combinado es 0, el área no se rellena.



Rellenos y recuentos de la regla de dirección de trazado
A. *Regla de dirección de trazado par-impar* ***B.*** *Regla de dirección de trazado distinta a cero*

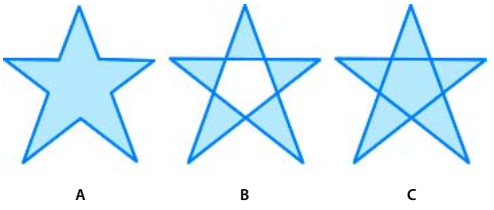
Uso de reglas de dirección de trazado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Estas reglas de relleno son complicadas, pero en algunas situaciones resultan necesarias. Pongamos como ejemplo el dibujo de una forma de estrella. Con la regla par-impar estándar, la forma necesitaría diez líneas diferentes. Con la regla de dirección de trazado distinta a cero, estas diez líneas se reducen a cinco. A continuación se muestra la secuencia de ActionScript para una estrella con cinco líneas y una regla de dirección de trazado distinta a cero:

```
graphics.beginFill(0x60A0FF);
graphics.drawPath( Vector.<int>([1,2,2,2,2]),  Vector.<Number>([66,10, 23,127, 122,50, 10,49,
109,127]),   GraphicsPathWinding.NON_ZERO);
```

Y a continuación se muestra la forma de estrella:



Forma de estrella con distintas reglas de dirección de trazado
A. *10 líneas (par-impar)* ***B.*** *5 líneas (par-impar)* ***C.*** *5 líneas distintas a cero*

A medida que las imágenes se animan o se utilizan como texturas en objetos tridimensionales y se superponen, las reglas de dirección de trazado cobran mayor importancia.

Uso de clases de datos gráficos

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La API de dibujo mejorada incluye varias clases en el paquete flash.display que implementan la interfaz IGraphicsData. Estas clases actúan como objetos de valor (contenedores de datos) que representan los métodos de dibujo de la API de dibujo.

Las siguientes clases implementan la interfaz IGraphicsData:

- GraphicsBitmapFill
- GraphicsEndFill

- GraphicsGradientFill
- GraphicsPath
- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Con estas clases, puede almacenar dibujos completos en un objeto Vector de tipo IGraphicsData (Vector.<IGraphicsData>). Así podrá reutilizar los datos de gráficos como origen de datos para otras instancias de forma, o para almacenar información de dibujo que utilizar posteriormente.

Se debe tener en cuenta que se dispone de varias clases de relleno para cada estilo de relleno, pero solo de una clase de trazo. ActionScript dispone únicamente de la clase de trazo IGraphicsData, ya que esta clase utiliza las clases de relleno para definir su estilo. Cada trazo se define mediante una combinación de la clase de trazo y una clase de relleno. De lo contrario, la API para estas clases de datos gráficos reflejan los métodos que representan en la clase flash.display.Graphics:

Método Graphics	**Clase correspondiente**
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

Asimismo, la clase GraphicsPath tiene sus propios métodos de utilidad `GraphicsPath.moveTo()`, `GraphicsPath.lineTo()`, `GraphicsPath.curveTo()`, `GraphicsPath.wideLineTo()` y `GraphicsPath.wideMoveTo()` para definir fácilmente los comandos para una instancia de GraphicsPath. Estos métodos de utilidad simplifican las tareas de definición o actualización de los comandos y valores de datos directamente.

Dibujo con datos de gráficos vectoriales

Cuando tenga una colección de instancias de IGraphicsData, utilice el método `drawGraphicsData()` para representar los gráficos. El método `drawGraphicsData()` contiene una serie de instrucciones de dibujo de un vector de instancias IGraphicsData en orden secuencial:

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70, 70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(GraphicsPathCommand.MOVE_TO, GraphicsPathCommand.LINE_TO,
GraphicsPathCommand.LINE_TO);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

Con la modificación de un valor en el trazado utilizado por el dibujo en el ejemplo, la forma se puede volver a dibujar varias veces para una imagen más compleja:

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

Aunque los objetos IGraphicsData pueden definir estilos de trazo y de relleno, estos no constituyen un requisito. Es decir, los métodos de la clase Graphics se pueden utilizar para definir estilos, mientras que los objetos IGraphicsData se pueden emplear para dibujar una colección de trazados guardados, o viceversa.

Nota: *utilice el método `Graphics.clear()` para borrar un dibujo anterior antes de comenzar uno nuevo, a no ser que se trate de una adición al dibujo original, tal y como se muestra en el ejemplo anterior. Conforme cambia una sola parte de un trazado o colección de objetos IGraphicsData, vuelva a dibujar todo el dibujo para ver los cambios.*

Cuando se utilizan clases de datos gráficos, el relleno se representa siempre que se dibujan tres o más puntos, ya que la figura se encuentra cerrada en este punto. Aunque se cierre el relleno, el trazo no lo hace y este comportamiento es diferente al uso de varios comandos `Graphics.lineTo()` o `Graphics.moveTo()`.

Lectura de datos de gráficos vectoriales

Flash Player 11.6 y posterior, Adobe AIR 3.6 y posterior

Además de dibujar contenido vectorial en un objeto de visualización, en Flash Player 11.6 y Adobe AIR 3.6 y versiones posteriores se puede utilizar el método de clase Graphics `readGraphicsData()` para obtener una representación de datos del contenido de gráficos vectoriales de un objeto de visualización. Esto puede servir para crear una instantánea de un gráfico para guardar, copiar o crear una hoja de Sprite en tiempo de ejecución, entre otras cosas.

Al llamar al método `readGraphicsData()` se obtiene una instancia Vector que contiene objetos IGraphicsData. Estos mismos objetos se usan para dibujar gráficos vectoriales con el método `drawGraphicsData()`.

La lectura de gráficos vectoriales mediante el método `readGraphicsData()` tiene varias limitaciones. Para obtener más información, consulte la información sobre readGraphicsData() en Referencia del lenguaje ActionScript.

Uso de drawTriangles()

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Otro método avanzado incluido en Flash Player 10 y Adobe AIR 1.5, `Graphics.drawTriangles()`, es similar al método `Graphics.drawPath()`. El método `Graphics.drawTriangles()` también utiliza un objeto Vector.<Number> para especificar ubicaciones de punto para dibujar un trazado.

Sin embargo, el principal objetivo del método `Graphics.drawTriangles()` es facilitar efectos tridimensionales en ActionScript. Para obtener más información sobre el uso de `Graphics.drawTriangles()` para generar efectos tridimensionales, consulte "Uso de triángulos para efectos 3D" en la página 381.

Capítulo 13: Trabajo con mapas de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además de sus funciones de dibujo vectorial, ActionScript 3.0 cuenta con la capacidad de crear imágenes de mapas de bits y de manipular los datos de píxeles de imágenes de mapas de bits externas cargadas en un archivo SWF. Gracias a la capacidad de acceder a los valores individuales de los píxeles y modificarlos, es posible crear efectos de imagen similares a los de los filtros y utilizar las funciones de ruido incorporadas para crear texturas y ruido aleatorio.

- Renaun Erickson: Rendering game assets in ActionScript using blitting techniques
- Bitmap programming: Capítulo 26 de ActionScript 3 de Colin Moock (O'Reilly Media, 2007)
- Mike Jones: Working with Sprites in Pushbutton Engine
- Flash & Math: Pixel Particles Made Simple
- Flixel

Fundamentos de la utilización de mapas de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se trabaja con imágenes digitales, lo habitual es encontrar dos tipos principales de gráficos: de mapas de bits y vectoriales. Los gráficos de mapas de bits, también conocidos como gráficos raster, se componen de cuadrados diminutos (píxeles) distribuidos en una cuadrícula rectangular. Los gráficos vectoriales se componen de formas geométricas generadas matemáticamente, como líneas, curvas y polígonos.

Las imágenes de mapa de bits se definen mediante la altura y la anchura de la imagen, medidas en píxeles, y el número de bits que contiene cada píxel, que representa el número de colores que un píxel puede mostrar. En el caso de las imágenes de mapa de bits que utilizan el modelo de color RGB, los píxeles se componen de tres bytes: rojo, verde y azul. Cada uno de ellos contiene un valor entre 0 y 255. Cuando los bytes se combinan en el píxel, producen un color de un modo similar a un artista cuando mezcla pinturas de distintos colores. Por ejemplo, un píxel cuyos bytes tuvieran los siguientes valores, rojo - 255, verde - 102 y azul - 0, mostraría un color anaranjado vivo.

La calidad de una imagen de mapa de bits viene dada por la combinación de su resolución y su número de bits de profundidad del color. La *resolución* hace referencia al número de píxeles que contiene una imagen. A mayor número de píxeles, mayor resolución y mayor nitidez de la imagen. La *profundidad del color* indica la cantidad de información que puede contener un píxel. Por ejemplo, una imagen con una profundidad de color de 16 bits por píxel no puede representar el mismo número de colores que una imagen con una profundidad de color de 48 bits. De este modo, la imagen de 48 bits tendrá unas variaciones de tonalidad más suaves que su homóloga de 16 bits.

Dado que los gráficos de mapa de bits dependen de la resolución, pueden presentar problemas al ajustar su escala. Esto resulta evidente cuando se intentan ajustar la escala para aumentar su tamaño. Al aumentar de tamaño un mapa de bits se suelen perder detalles y calidad.

Formatos de mapa de bits

Las imágenes de mapa de bits se agrupan en una serie de formatos de archivo comunes. Estos formatos utilizan distintos tipos de algoritmos de compresión para reducir el tamaño del archivo y optimizar su calidad basándose en el uso que se vaya a dar a la imagen. Los formatos de imágenes de mapa de bits admitidos en los motores de ejecución de Adobe son BMP, GIF, JPG, PNG y TIFF.

BMP

El formato BMP (mapa de bits) es un formato de imagen predeterminado utilizado por el sistema operativo Microsoft Windows. No utiliza ningún algoritmo de compresión, por lo que el tamaño de los archivos suele ser grande.

GIF

El formato GIF (Graphics Interchange Format) fue desarrollado originalmente por CompuServe en 1987 como medio para transmitir imágenes con 256 colores (color de 8 bits). Este formato da lugar a archivos de pequeño tamaño y es ideal para imágenes que se van a usar en la web. A causa de lo limitado de la paleta de colores de este formato, las imágenes GIF normalmente no son adecuadas para fotografías, que suelen requerir mayores variaciones de tonalidad y degradados de color. En las imágenes GIF es posible utilizar transparencia de un bit, lo que permite designar un color como vacío (o transparente). Esto hace que el color de fondo de una página web se pueda ver a través de la imagen en la que se ha asignado la transparencia.

JPEG

Desarrollado por el Joint Photographic Experts Group (JPEG), el formato de imagen JPEG (a veces escrito JPG) utiliza un algoritmo de compresión con pérdida que permite combinar una profundidad de color de 24 bits con un tamaño de archivo reducido. En la compresión con pérdida, la imagen pierde calidad y datos cada vez que se guarda, pero se logra un tamaño de archivo menor. El formato JPEG resulta perfecto para fotografías, ya que puede mostrar millones de colores. La capacidad de controlar el grado de compresión aplicado a una imagen permite jugar con la calidad de la imagen y el tamaño del archivo.

PNG

El formato PNG (Portable Network Graphics) se originó como alternativa de código abierto al formato de archivo GIF, que está patentado. Los archivos PNG permiten una profundidad de color de hasta 64 bits, lo que equivale a más de 16 millones de colores. Dado que el formato PNG es relativamente nuevo, algunos navegadores antiguos no admiten este tipo de archivos. Al contrario que los JPG, los archivos PNG utilizan una compresión sin pérdida, lo que quiere decir que cuando la imagen se guarda no se pierden datos. Los archivos PNG también permiten el uso de transparencias alfa, con un máximo de 256 niveles de transparencia.

TIFF

El formato TIFF (formato de archivo de imagen etiquetada) era el formato multiplataforma más utilizado antes de la aparición del formato PNG. El inconveniente del formato TIFF proviene de las muchas variedades de TIFF existentes y de la ausencia de un único lector capaz de gestionarlas todas. Además, ningún navegador web admite este formato actualmente. El formato TIFF puede utilizar compresión con o sin pérdidas y es capaz de gestionar espacios de color específicos del dispositivo (por ejemplo, CMYK).

Mapas de bits transparentes y opacos

Las imágenes de mapa de bits que utilizan los formatos GIF o PNG pueden tener un byte extra (canal alfa) para cada píxel, que representa su valor de transparencia.

Las imágenes GIF permiten una transparencia de un solo bit, lo que quiere decir que se puede especificar que un único color, de la paleta de 256, puede ser transparente. Por su parte, las imágenes PNG pueden tener hasta 256 niveles de transparencia. Esta función resulta especialmente útil cuando es necesario que las imágenes o el texto se fundan con los fondos.

ActionScript 3.0 replica este byte de transparencia extra de los píxeles dentro de la clase BitmapData. De manera similar al modelo de transparencia de PNG, ActionScript ofrece hasta 256 niveles de transparencia.

Conceptos y términos importantes

La siguiente lista contiene términos importantes que aparecen al tratar el tema de los gráficos de mapa de bits:

Alfa Nivel de transparencia (o, para ser más precisos, opacidad) de un color o una imagen. La cantidad de alfa suele describirse como el valor del *canal alfa*.

Color ARGB Esquema de color en el que el color de cada píxel es una mezcla de los valores de los colores rojo, verde y azul, y su transparencia se especifica como un valor alfa.

Canal de color Normalmente, los colores se representan como una mezcla de varios colores básicos (en el campo de los gráficos digitales suelen ser rojo, verde y azul). Cada color básico se considera un canal de color. La mezcla de la cantidad de color de cada canal determina el color final.

Profundidad de color También denominada *profundidad de bits*, este concepto hace referencia a la cantidad de memoria del equipo dedicada a cada píxel que, a su vez, determina el número de posibles colores que se pueden representar en la imagen.

Píxel Unidad de información mínima de una imagen de mapa de bits (esencialmente, un punto de color).

Resolución Dimensiones en píxeles de una imagen, que determinan el nivel de detalle que contiene la imagen. La resolución se suele expresar en términos del número de píxeles de anchura y de altura.

Color RGB Esquema de color en el que el color de cada píxel se representa como una mezcla de los valores de los colores rojo, verde y azul.

Las clases Bitmap y BitmapData

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clases principales de ActionScript 3.0 para trabajar con imágenes de mapa de bits son Bitmap, que se usa para visualizar imágenes de mapa de bits en pantalla, y la clase BitmapData, utilizada para acceder y manipular los datos de imagen sin procesar de un mapa de bits.

Más temas de ayuda

flash.display.Bitmap

flash.display.BitmapData

Aspectos básicos de la clase Bitmap

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Como subclase de la clase DisplayObject, la clase Bitmap es la clase principal de ActionScript 3.0 para mostrar imágenes de mapa de bits. Estas imágenes pueden haberse cargado a través de la clase flash.display.Loader o crearse dinámicamente utilizando el constructor `Bitmap()`. Al cargar una imagen de una fuente externa, los objetos Bitmap solo pueden usar los formatos de imagen GIF, JPEG o PNG. Una vez creada, la instancia de Bitmap se puede considerar como un envolvente de un objeto BitmapData que se debe representar en el escenario. Dado que las instancias de Bitmap son objetos de visualización, también se pueden utilizar todas las características y funcionalidades de los objetos de visualización para manipularlas. Para obtener más información sobre la utilización de los objetos de visualización, consulte "Programación de la visualización" en la página 154.

Ajuste a píxeles y suavizado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además de las funcionalidades comunes a todos los objetos de visualización, la clase Bitmap proporciona algunas características adicionales específicas de las imágenes de mapa de bits.

La propiedad `pixelSnapping` de la clase Bitmap determina si un objeto Bitmap se ajusta o no a su píxel más cercano. Esta propiedad acepta una de las tres constantes definidas en la clase PixelSnapping: `ALWAYS`, `AUTO` y `NEVER`.

La sintaxis para aplicar el ajuste a píxeles es la siguiente:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Suele ocurrir que, cuando se escalan las imágenes de mapa de bits, estas se vuelven difusas y distorsionadas. Para reducir esta distorsión se puede utilizar la propiedad `smoothing` de la clase BitmapData. Cuando esta propiedad booleana está definida como `true`, al escalar la imagen, los píxeles de esta se suavizan. Esto otorga a la imagen una apariencia más clara y natural.

Aspectos básicos de la clase BitmapData

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase BitmapData, que pertenece al paquete flash.display, se asemeja a una instantánea fotográfica de los píxeles que contiene una imagen de mapa de bits cargada o creada dinámicamente. Esta instantánea se representa mediante un conjunto que contiene los datos de los píxeles del objeto. La clase BitmapData también contiene una serie de métodos incorporados que resultan muy útiles a la hora de crear y manipular los datos de los píxeles.

Se puede usar el siguiente código para crear una instancia de un objeto BitmapData:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Los parámetros `width` y `height` especifican el tamaño del mapa de bits. Desde AIR 3 y Flash Player 11, ya no hay límite de tamaño para un objeto BitmapData. El tamaño máximo de un mapa de bits depende del sistema operativo.

En AIR 1.5 y en Flash Player 10, el tamaño máximo de un objeto BitmapData es 8.191 píxeles de anchura o de altura; el número total de píxeles no puede superar los 16.777.215. (De modo que, si un objeto BitmapData tiene 8.191 píxeles de anchura, tendrá como mucho 2.048 píxeles de altura.) En Flash Player 9 y versiones anteriores, y en AIR 1.1 y versiones anteriores, la limitación es de 2.880 píxeles de altura y 2.880 de anchura.

El parámetro `transparent` especifica si entre los datos del mapa de bits se incluye un canal alfa (`true`) o no (`false`). El parámetro `fillColor` es un valor de color de 32 bits que especifica el color de fondo, así como el valor de la transparencia (si se le ha definido como `true`). En el siguiente ejemplo se crea un objeto BitmapData con un fondo anaranjado y una transparencia del 50 por ciento:

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Para representar en la pantalla un objeto BitmapData recién creado, se debe asignar a una instancia de Bitmap o envolverse en ella. Para ello, se puede pasar el objeto BitmapData como parámetro del constructor del objeto Bitmap o asignarlo a la propiedad `bitmapData` de una instancia de Bitmap existente. También es necesario añadir la instancia de Bitmap a la lista de visualización llamando para ello a los métodos `addChild()` o `addChildAt()` del contenedor del objeto de visualización que contendrá a la instancia de Bitmap. Para obtener más información sobre el trabajo con la lista de visualización, consulte "Añadir objetos de visualización a la lista de visualización" en la página 163.

En el siguiente ejemplo se crea un objeto BitmapData con un relleno rojo y se muestra en una instancia de Bitmap:

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Manipulación de píxeles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase BitmapData contiene un conjunto de métodos que permiten manipular los valores de los datos de los píxeles.

Manipulación de píxeles individuales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se desea cambiar la apariencia de una imagen de mapa de bits a nivel de sus píxeles, en primer lugar es necesario obtener los valores de color de los píxeles que contiene el área que se pretende manipular. Para leer esos valores de los píxeles se utiliza el método `getPixel()`.

El método `getPixel()` obtiene el valor RGB del par de coordenadas x, y (píxel) que se le pasan como parámetros. Si alguno de los píxeles que se desea manipular incluye información de transparencia (canal alfa), será necesario emplear el método `getPixel32()`. Este método también lee un valor RGB pero, al contrario que `getPixel()`, el valor devuelto por `getPixel32()` contiene datos adicionales que representan el valor del canal alfa (transparencia) del píxel seleccionado.

Por otra parte, si simplemente se desea cambiar el color o la transparencia de un píxel que pertenece a un mapa de bits, se pueden usar los métodos `setPixel()` o `setPixel32()`. Para definir el color de un píxel basta con pasar las coordenadas x, y, además del valor del color, a uno de estos métodos.

En el siguiente ejemplo se utiliza `setPixel()` para dibujar una cruz en un fondo BitmapData verde. A continuación, se emplea `getPixel()` para leer el valor del color del píxel que se encuentra en las coordenadas 50, 50 y se realiza un seguimiento del valor devuelto.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

Si se desea leer el valor de un grupo de píxeles, en lugar del de uno solo, se debe usar el método `getPixels()`. Este método genera un conjunto de bytes a partir de una región rectangular de datos de píxeles que se transmite como parámetro. Cada uno de los elementos del conjunto de bytes (dicho de otro modo, los valores de los píxeles) es un entero sin signo, es decir, un valor de píxel no multiplicado de 32 bits.

A la inversa, para cambiar (o definir) el valor de un grupo de píxeles, se usa el método `setPixels()`. Este método recibe dos parámetros (`rect` y `inputByteArray`), que se combinan para dar lugar a una región rectangular (`rect`) de datos de píxeles (`inputByteArray`).

A medida que se leen (y escriben) los datos de `inputByteArray`, se llama al método `ByteArray.readUnsignedInt()` para cada uno de los píxeles del conjunto. Si, por algún motivo, `inputByteArray` no contiene todo un rectángulo de datos de píxeles, el método deja de procesar los datos de imagen en ese punto.

Es importante recordar que, tanto para leer como para definir los datos de los píxeles, el conjunto de bytes espera valores de píxeles de 32 bits compuestos por los canales alfa, rojo, verde y azul (ARGB).

En el siguiente ejemplo se utilizan los métodos `getPixels()` y `setPixels()` para copiar un grupo de píxeles de un objeto BitmapData a otro:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

Detección de colisiones a nivel de píxeles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `BitmapData.hitTest()` lleva a cabo una detección de colisiones a nivel de píxeles entre los datos de un mapa de bits y otro objeto o punto.

El método `BitmapData.hitTest()` acepta cinco parámetros:

- `firstPoint` (Point): este parámetro hace referencia a la posición del píxel de la esquina superior izquierda del primer objeto BitmapData sobre el que se realizará la comprobación de colisiones.
- `firstAlphaThreshold` (uint): este parámetro especifica el valor de canal alfa más alto que se considera opaco para esta prueba.
- `secondObject` (objeto): este parámetro representa el área de impacto. El objeto `secondObject` puede ser un objeto Rectangle, Point, Bitmap o BitmapData. Este objeto representa el área de impacto sobre la que se realizará la detección de colisiones.
- `secondBitmapDataPoint` (Point): este parámetro opcional se utiliza para definir la posición de un píxel en el segundo objeto BitmapData. y solo se utiliza cuando el valor de `secondObject` es un objeto BitmapData. El valor predeterminado es `null`.
- `secondAlphaThreshold` (uint): este parámetro opcional representa el valor de canal alfa más alto que se considera opaco en el segundo objeto BitmapData. El valor predeterminado es 1. Este parámetro solo se utiliza cuando el valor de `secondObject` es un objeto BitmapData y los dos objetos BitmapData son transparentes.

Al llevar a cabo la detección de colisiones sobre imágenes opacas, es conveniente recordar que ActionScript trata la imagen como si fuera un rectángulo (o recuadro delimitador) totalmente opaco. Por otra parte, al realizar la prueba de impactos a nivel de píxeles en imágenes transparentes, es necesario que las dos imágenes sean transparentes. Además, ActionScript utiliza los parámetros de umbral alfa para determinar en qué punto los píxeles pasan de ser transparentes a opacos.

En el siguiente ejemplo se crean tres imágenes de mapa de bits y se realiza una detección de colisiones de píxeles utilizando dos puntos de colisión distintos (uno devuelve false y el otro true):

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

Copia de datos de mapas de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para copiar datos de mapas de bits de una imagen a otra, dispone de varios métodos: `clone()`, `copyPixels()`, `copyChannel()`, `draw()` y `drawWithQuality()` (`drawWithQuality` disponibles en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores).

Como indica su nombre, el método `clone()` permite clonar datos de mapas de bits (o tomar muestras de ellos) de un objeto BitmapData a otro. Cuando se le llama, este método devuelve un nuevo objeto BitmapData que es una copia exacta de la instancia original que se ha clonado.

En el siguiente ejemplo se clona una copia de un cuadrado de color naranja (el elemento principal) y se coloca el clon junto al cuadro naranja principal:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

El método `copyPixels()` constituye una forma rápida y sencilla de copiar píxeles de un objeto BitmapData a otro. Este método toma una instantánea rectangular (definida por el parámetro `sourceRect`) de la imagen de origen y la copia en un área rectangular distinta (del mismo tamaño). La ubicación del rectángulo recién "pegado" se define mediante el parámetro `destPoint`.

El método `copyChannel()` toma una muestra de un valor de canal de color predefinido (alfa, rojo, verde o azul) de un objeto BitmapData de origen y la copia en un canal de un objeto BitmapData de destino. Llamar a este método no afecta a los demás canales del objeto BitmapData de destino.

Los métodos `draw()` y `drawWithQuality()` dibujan, o procesan, el contenido gráfico de un objeto Sprite, de un clip de película o de otro objeto de visualización de origen en un nuevo mapa de bits. Mediante los parámetros `matrix`, `colorTransform`, `blendMode` y `clipRect` de destino, es posible modificar la forma en la que se representa el nuevo mapa de bits. Este método utiliza el vector procesado en Flash Player y AIR para generar los datos.

Al llamar a `draw()` o a `drawWithQuality()`, se le transfiere el objeto de origen (Sprite, clip de película u otro objeto de visualización) como primer parámetro, tal y como puede verse a continuación:

```
myBitmap.draw(movieClip);
```

Si al objeto de origen se le ha aplicado alguna transformación (color, matriz, etc.) después de haber sido cargado originalmente, estas no se copiarán en el nuevo objeto. Para copiar las transformaciones en el nuevo mapa de bits, es necesario copiar el valor de la propiedad `transform` del objeto original en la propiedad `transform` del objeto Bitmap utilizado por el nuevo objeto BitmapData.

Compresión de datos de mapa de bits

Flash Player 11.3 y posterior, AIR 3.3 y posterior

El método `flash.display.BitmapData.encode()` permite comprimir de forma nativa datos de mapa de bits con uno de los siguientes formatos de compresión de imagen:

- **PNG**: utiliza compresión PNG, opcionalmente con compresión rápida. Esto da más prioridad a la velocidad de compresión que al tamaño del archivo. Para utilizar compresión PNG, transfiera un objeto `flash.display.PNGEncoderOptions` nuevo como segundo parámetro del método `BitmapData.encode()`.
- **JPEG**: utiliza compresión JPEG, opcionalmente con ajuste de calidad de imagen. Para utilizar compresión JPEG, transfiera un objeto `flash.display.JPEGEncoderOptions` nuevo como segundo parámetro del método `BitmapData.encode()`.

- **JPEGXR**: utiliza compresión JPEG Extended Range (XR), opcionalmente con ajuste de canal de color, sin pérdidas y con configuración de entropía. Para utilizar compresión JPEGXR, transfiera un objeto `flash.display.JPEGXREncoderOptions` nuevo como segundo parámetro del método `BitmapData.encode()`.

Puede utilizar esta función de procesamiento de imagen como parte del flujo de trabajo de carga y descarga en el servidor.

El siguiente ejemplo de fragmento de código comprime un objeto BitmapData con `JPEGEncoderOptions`:

```
// Compress a BitmapData object as a JPEG file.
var bitmapData:BitmapData = new BitmapData(640,480,false,0x00FF00);
var byteArray:ByteArray = new ByteArray();
bitmapData.encode(new Rectangle(0,0,640,480), new flash.display.JPEGEncoderOptions(),
byteArray);
```

Creación de texturas con funciones de ruido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para modificar la apariencia de un mapa de bits, se le puede aplicar un efecto de ruido utilizando para ello los métodos `noise()` o `perlinNoise()`. Un efecto de ruido puede asemejarse a la estática que aparece en una pantalla de televisión no sintonizada.

Para aplicar un efecto de ruido a un mapa de bits se utiliza el método `noise()`. Este método aplica un valor de color aleatorio a los píxeles que se hallan dentro de un área especificada de una imagen de mapa de bits.

Este método acepta cinco parámetros:

- `randomSeed` (int): número de inicialización aleatorio que determinará el patrón. A pesar de lo que indica su nombre, este número crea los mismos resultados siempre que se pasa el mismo número. Para obtener un resultado realmente aleatorio, es necesario utilizar el método `Math.random()` para transmitir un número aleatorio a este parámetro.
- `low` (uint): este parámetro hace referencia al valor más bajo que se generará para cada píxel (de 0 a 255). El valor predeterminado es 0. Si se establece un valor más bajo, se originará un patrón de ruido más oscuro, mientras que con un valor más alto, el patrón será más brillante.
- `high` (uint): este parámetro hace referencia al valor más alto que se generará para cada píxel (de 0 a 255). El valor predeterminado es 255. Si se establece un valor más bajo, se originará un patrón de ruido más oscuro, mientras que con un valor más alto, el patrón será más brillante.
- `channelOptions` (uint): este parámetro especifica el canal de color del objeto de mapa de bits al que se aplicará el patrón de ruido. El número puede ser una combinación de cualquiera de los cuatro valores ARGB de los canales de color. El valor predeterminado es 7.
- `grayScale` (Boolean): cuando se define como `true`, este parámetro aplica el valor de `randomSeed` a los píxeles del mapa de bits, de modo que se elimina todo el color de la imagen. El canal alfa no se ve afectado por este parámetro. El valor predeterminado es `false`.

En el siguiente ejemplo se crea una imagen de mapa de bits y se le aplica un patrón de ruido azul:

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise1 extends Sprite
    {
        public function BitmapNoise1()
        {
            var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
            myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
            var image:Bitmap = new Bitmap(myBitmap);
            addChild(image);
        }
    }
}
```

Si se desea crear una textura de apariencia más orgánica, es aconsejable utilizar el método `perlinNoise()`. El método `perlinNoise()` produce texturas orgánicas realistas que resultan idóneas para humo, nubes, agua, fuego e incluso explosiones.

Dado que se genera mediante un algoritmo, el método `perlinNoise()` utiliza menos memoria que las texturas basadas en mapas de bits. Sin embargo, puede repercutir en el uso del procesador y ralentizar el contenido, haciendo que la pantalla se vuelva a dibujar más lentamente que la velocidad de fotogramas, sobre todo en equipos antiguos. Principalmente esto se debe a los cálculos en punto flotante que se deben ejecutar para procesar los algoritmos de ruido Perlin.

Este método acepta nueve parámetros (los seis primeros son obligatorios):

- `baseX` (Number): determina el valor x (tamaño) de los patrones creados.
- `baseY` (Number): determina el valor y (tamaño) de los patrones creados.
- `numOctaves` (uint): número de octavas o funciones de ruido individuales que se van a combinar para crear este ruido. A mayor número de octavas, mayor detalle pero también más tiempo de procesamiento.
- `randomSeed` (int): el número de inicialización aleatorio funciona exactamente igual que en la función `noise()`. Para obtener un resultado realmente aleatorio, es necesario utilizar el método `Math.random()` para transmitir un número aleatorio a este parámetro.
- `stitch` (Boolean): si se establece en `true`, este método intenta unir (o suavizar) los bordes de transición de la imagen para crear texturas continuas que se pueden utilizar como mosaicos para rellenos de mapas de bits.
- `fractalNoise` (Boolean): este parámetro hace referencia a los bordes de los degradados generados mediante este método. Si se define como `true`, el método genera ruido fractal que suaviza los bordes del efecto. Si se define como `false`, genera turbulencia. Una imagen con turbulencia presenta discontinuidades visibles en el degradado que pueden producir efectos visuales más nítidos, como llamas u olas del mar.
- `channelOptions` (uint): el parámetro `channelOptions` funciona del mismo modo que en el método `noise()`. Con él es posible especificar a qué canal de color (del mapa de bits) se aplicará el patrón de ruido. El número puede ser una combinación de cualquiera de los cuatro valores ARGB de los canales de color. El valor predeterminado es 7.
- `grayScale` (Boolean): el parámetro `grayScale` funciona del mismo modo que en el método `noise()`. Si se define como `true`, aplica el valor de `randomSeed` a los píxeles del mapa de bits, de modo que se elimina todo el color de la imagen. El valor predeterminado es `false`.

- `offsets` (Conjunto): conjunto de puntos que corresponde a desplazamientos x e y para cada octava. Mediante la manipulación de los valores de desplazamiento se pueden mover suavemente las capas de la imagen. Cada punto de la conjunto de desplazamiento afecta a una función de ruido de octava específica. El valor predeterminado es `null`.

En el siguiente ejemplo se crea un objeto BitmapData de 150 x 150 píxeles que llama al método `perlinNoise()` para generar un efecto de nubes azul y verde:

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise2 extends Sprite
    {
        public function BitmapNoise2()
        {
            var myBitmapDataObject:BitmapData =
                new BitmapData(150, 150, false, 0x00FF0000);

            var seed:Number = Math.floor(Math.random() * 100);
            var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
            myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

            var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
            addChild(myBitmap);
        }
    }
}
```

Desplazamiento por mapas de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Supongamos que se ha creado una aplicación de mapas de calles en la que cada vez que el usuario mueve el mapa, hay que actualizar la vista (incluso si el mapa solo se ha movido unos pocos píxeles).

Una forma de crear esta funcionalidad sería volver a representar una nueva imagen que contuviese la vista actualizada del mapa cada vez que el usuario lo moviese. También se podría crear una única imagen de gran tamaño y utilizar el método `scroll()`.

El método `scroll()` copia un mapa de bits que aparece en pantalla y lo pega en una nueva posición desplazada, que viene especificada por los parámetros (`x`, `y`). Si una parte del mapa de bits se encuentra fuera de la pantalla, se produce la impresión que la imagen se ha desplazado. Cuando se combina con una función de temporizador (o un evento `enterFrame`), se puede hacer que la imagen parezca moverse o desplazarse.

En el siguiente ejemplo se parte del ejemplo anterior del ruido de Perlin y se genera una imagen de mapa de bits mayor (tres cuartas partes de la cual se representan fuera de la pantalla). A continuación, se aplica el método `scroll()` junto con un detector de eventos `enterFrame` que desplaza la imagen un píxel en dirección diagonal descendente. Este método se llama cada vez que se entra en el fotograma y, de este modo, las partes de la imagen que quedan fuera de la pantalla se representan en el escenario a medida que la imagen se desplaza hacia abajo.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

Aprovechamiento de la técnica de mipmapping

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los *mapas MIP* (también denominados *mipmaps*), son mapas de bits agrupados de forma conjunta y asociados a una textura para aumentar el rendimiento y la calidad de representación en tiempo de ejecución. Cada imagen de mapa de bits del mapa MIP es una versión de la imagen de mapa de bits principal, pero a un nivel detalle menor que la imagen principal.

Por ejemplo, puede tener un mapa MIP que una imagen principal de 64 × 64 píxeles con la máxima calidad. Las imágenes de menor calidad del mapa MIP serían de 32 × 32, 16 × 16, 8 × 8, 4 × 4, 2 × 2 y 1 × 1 píxeles.

El *flujo de texturas* es la capacidad de cargar primero el mapa de bits de menor calidad y, a continuación, visualizar progresivamente mapas de bits de mayor calidad a medida que se carguen. Como los mapas de bits de menor calidad son pequeños, se cargan más rápido que la imagen principal. Por ello, los usuarios de la aplicación podrán ver la imagen de una aplicación antes de que se cargue el mapa de bits principal con mayor calidad.

Flash Player 9.115.0. y las versiones posteriores y AIR implementan esta tecnología (el proceso se denomina *mipmapping*), creando versiones optimizadas de escala variable de cada mapa de bits (comenzando en un 50%).

Flash Player 11.3 y AIR 3.3 admiten flujo de texturas mediante el parámetro `streamingLevels` de los métodos `Context3D.createCubeTexture()` y `Context3D.createTexture()`.

La compresión de texturas permite guardar imágenes de textura en formato comprimido directamente en la GPU, ahorrando así memoria de GPY y ancho de banda de memoria. Normalmente, las texturas comprimidas se comprimen sin conexión y se cargan en la GPU en formato comprimido. No obstante, Flash Player 11.4 y AIR 3.4 admiten compresión de texturas en tiempo de ejecución, y esto resulta útil en muchas situaciones, por ejemplo, cuando se procesan texturas dinámicas de ilustraciones vectoriales. Para utilizar la compresión de texturas en tiempo de ejecución, lleve a cabo estos pasos:

- Cree el objeto de textura llamando al método `Context3D.createTexture()`, transfiriendo `flash.display3D.Context3DTextureFormat.COMPRESSED` o `flash.display3D.Context3DTextureFormat.COMPRESSED_ALPHA` en el tercer parámetro.

- Mediante la instancia de `flash.display3D.textures.Texture` devuelta por `createTexture()`, llame a `flash.display3D.textures.Texture.uploadFromBitmapData()` o a `flash.display3D.textures.Texture.uploadFromByteArray()`. Estos métodos cargan y comprimen la textura en un solo paso.

Los mapas MIP se crean para los siguientes tipos de mapas de bits:

- Un mapa de bits (archivos JPEG, GIF o PNG) que se muestra utilizando la clase Loader de ActionScript 3.0.
- Un mapa de bits de la biblioteca de un documento de Flash Professional.
- Un objeto BitmapData.
- Un mapa de bits que se muestra utilizando la función `loadMovie()` de ActionScript 2.0.

Los mapas MIP no se aplican a objetos filtrados o clips de película almacenados en caché de mapa de bits. Sin embargo, los mapas MIP se aplican si dispone de transformaciones de mapa de bits en un objeto de visualización filtrado, aunque el mapa de bits esté en un contenido con máscara.

La técnica de "mipmapping" se realiza automáticamente, pero puede seguir algunas directrices para garantizar que sus imágenes aprovechan esta optimización:

- Para la reproducción de vídeo, establezca la propiedad `smoothing` en `true` para el objeto Video (consulte la clase Video).
- Para los mapas de bits, la propiedad `smoothing` no tiene que establecerse en `true`, pero las mejoras de calidad serán más visibles si los mapas de bits utilizan el suavizado.
- Utilice tamaños de mapa de bits que sean divisibles por 4 o por 8 para imágenes bidimensionales (por ejemplo, 640 x 128, que se puede reducir del siguiente modo: 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1).

  En el caso de texturas tridimensionales, utilice mapas MIP en los que cada imagen esté a una resolución que sea potencia de 2 (es decir, 2^n). Por ejemplo, la imagen principal está a una resolución de 1024 x 1024 píxeles. Las imágenes de menor calidad del mapa MIP estarían entre 512 x 512, 256 x 256, 128 x 128 y 1 x 1 píxeles para un total de 11 imágenes en el mapa MIP.

  Tenga en cuenta que la técnica de mipmapping no se produce para el contenido de mapa de bits con una altura o anchura distinta.

Ejemplo de mapa de bits: Luna giratoria animada

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Con el ejemplo de la luna giratoria animada se muestran estas técnicas para trabajar con objetos Bitmap y datos de imagen de mapa de bits (objetos BitmapData). El ejemplo crea una animación de una luna esférica giratoria utilizando una imagen plana de la superficie de la luna como datos de imagen sin procesar. Se muestran las siguientes técnicas:

- Carga de una imagen externa y acceso a sus datos de imagen sin procesar.
- Creación de una animación copiando píxeles de forma repetida de diferentes partes de una imagen de origen.
- Creación de una imagen de mapa de bits estableciendo valores de píxel

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación de la luna giratoria animada se encuentran en la carpeta Samples/SpinningMoon. La aplicación consta de los siguientes archivos:

Archivo	Descripción
SpinningMoon.mxml o SpinningMoon.fla	Archivo principal de la aplicación en Flex (MXML) o en Flash (FLA).
com/example/programmingas3/moon/MoonSphere.as	Clase que realiza la funcionalidad de cargar, visualizar y animar la luna.
moonMap.png	Archivo de imagen que contiene una fotografía de la superficie de la luna, que se carga y se utiliza para crear la luna giratoria animada.

Carga de una imagen externa como datos de mapa de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La primera tarea importante que se realiza en este ejemplo es cargar un archivo de imagen externo, que es una fotografía de la superficie de la luna. La operación de carga se administra mediante dos métodos en la clase MoonSphere: el constructor `MoonSphere()`, donde se inicia el proceso de carga y el método `imageLoadComplete()`, que se llama cuando la imagen externa está completamente cargada.

La carga de una imagen externa es similar a la carga de un archivo SWF externo; ambos utilizan una instancia de la clase flash.display.Loader para realizar la operación de carga. El código real del método `MoonSphere()` que inicia la carga de la imagen se presenta del siguiente modo:

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

La primera línea declara la instancia de Loader denominada `imageLoader`. La tercera línea inicia realmente el proceso de carga llamando al método `load()` del objeto Loader, transmitiendo una instancia de URLRequest que representa la URL de la imagen que se va a cargar. La segunda línea configura el detector de eventos que se activará cuando la imagen se haya cargado por completo. Se debe tener en cuenta que el método `addEventListener()` no se llama en la propia instancia de Loader; en cambio, se llama en la propiedad `contentLoaderInfo` del objeto Loader. La propia instancia de Loader no distribuye eventos relacionados con el contenido que se está cargando. No obstante, su propiedad `contentLoaderInfo` contiene una referencia al objeto LoaderInfo que se asocia al contenido que se está cargando en el objeto Loader (la imagen externa en este caso). El objeto LoaderInfo proporciona varios eventos relacionados con el curso y la finalización de la carga de contenido externo, incluyendo el evento `complete` (`Event.COMPLETE`) que activará una llamada al método `imageLoadComplete()` cuando la imagen se haya cargado por completo.

Aunque el inicio de la carga de la imagen externa es una parte importante del proceso, es igualmente relevante saber qué hacer cuando finalice la carga. Tal y como se muestra en el código anterior, la función `imageLoadComplete()` se llama cuando se carga la imagen. La función realiza varias operaciones con los datos de imagen cargados, lo que se describe posteriormente. No obstante, para utilizar los datos de imagen, es necesario acceder a los mismos. Si se usa un objeto Loader para cargar una imagen externa, la imagen cargada se convierte en una instancia de Bitmap, que se asocia como objeto de visualización secundario del objeto Loader. En este caso, la instancia de Loader está disponible para el método del detector de eventos como parte del objeto de evento que se transmite al método como parámetro. Las primeras líneas del método `imageLoadComplete()` se presentan del siguiente modo:

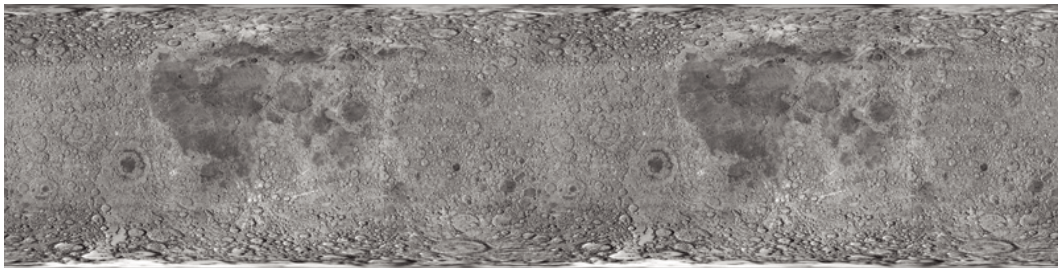
```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Observe que el parámetro del objeto de evento se denomina `event` y es una instancia de la clase Event. Todas las instancias de la clase Event disponen de una propiedad `target`, que hace referencia al objeto que activa el evento (en este caso, la instancia de LoaderInfo en la que se llamó el método `addEventListener()`, tal y como se describió anteriormente). Por su parte, el objeto LoaderInfo tiene una propiedad `content` que (una vez completado el proceso de carga) contiene la instancia de Bitmap con la imagen de mapa de bits cargada. Si desea visualizar la imagen directamente en pantalla, puede asociar esta instancia de Bitmap (`event.target.content`) a un contenedor de objetos de visualización. (También puede asociar el objeto Loader a un contenedor de objetos de visualización). No obstante, en este ejemplo el contenido cargado se utiliza como origen de los datos de la imagen sin procesar en lugar de mostrarse en pantalla. Por ello, la primera línea del método `imageLoadComplete()` lee la propiedad `bitmapData` de la instancia de Bitmap cargada (`event.target.content.bitmapData`) y la almacena en la variable de la instancia denominada `textureMap`, que se utiliza como origen de los datos de imagen para crear la animación de la luna en rotación. Este punto se describe a continuación.

Creación de animación con copia de píxeles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una definición básica de animación es la ilusión de movimiento o cambio, creado mediante el cambio de una imagen en el tiempo. En este ejemplo, el objetivo es crear la ilusión de una rotación de la luna esférica alrededor de su eje vertical. Sin embargo, para los objetivos de la animación, se puede omitir el aspecto de distorsión esférica del ejemplo. Observe la imagen real que se carga y se utiliza como el origen de los datos de imagen de la luna:



Como se puede ver, la imagen no es una o varias esferas; se trata de una fotografía rectangular de la superficie de la luna. Debido a que la foto fue tomada exactamente en el ecuador de la luna, las partes de la imagen más cercanas a la parte superior e inferior de la imagen están expandidas y distorsionadas. Para eliminar la distorsión de la imagen y hacerla parecer esférica, utilizaremos un filtro de mapa de desplazamiento, tal y como se describe más adelante. Sin embargo. debido a que esta imagen de origen es un rectángulo, para crear la ilusión de que la esfera está girando, el código simplemente necesita deslizar la fotografía de la superficie de la luna horizontalmente.

Se debe tener en cuenta que la imagen incluye realmente dos copias de la fotografía de la superficie de la luna próximas entre sí. Esta imagen es la de origen a partir de la que los datos de imagen se copian repetidamente para crear la apariencia de movimiento. Al tener dos copias de la imagen próximas entre sí, un efecto de desplazamiento ininterrumpido y continuo puede ser más fácil de crear. Analicemos el proceso de animación paso a paso para ver su funcionamiento.

El proceso incluye dos objetos de ActionScript independientes. En primer lugar se dispone de la imagen de origen cargada, que en el código se representa mediante una instancia de BitmapData denominada `textureMap`. Tal y como se ha descrito anteriormente, `textureMap` se llena con datos de imagen una vez cargada la imagen externa, utilizando este código:

```
textureMap = event.target.content.bitmapData;
```

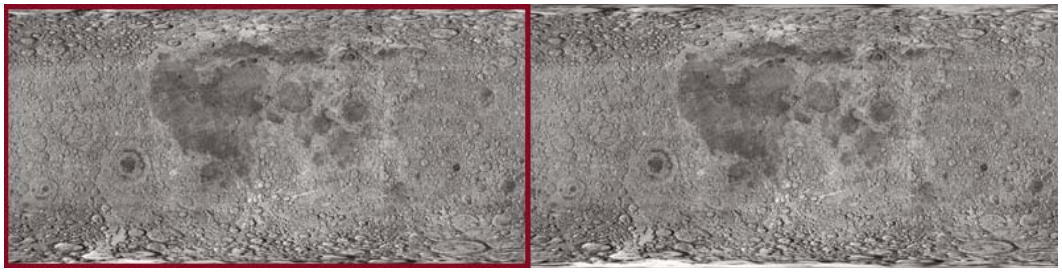
El contenido de `textureMap` es la imagen de la luna con forma rectangular. Asimismo, para crear la rotación animada, en el código se utiliza una instancia de Bitmap denominada `sphere`, que es el objeto de visualización real que muestra la imagen de la luna en pantalla. Al igual que sucede con `textureMap`, el objeto `sphere` se crea y se llena con sus datos de imagen inicial en el método `imageLoadComplete()`, empleando el siguiente código:

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                        new Rectangle(0, 0, sphere.width, sphere.height),
                        new Point(0, 0));
```

Tal y como muestra el código, se crea una instancia de `sphere`. Su propiedad `bitmapData` (los datos de la imagen sin procesar que se muestran mediante `sphere`) se crea con la misma altura y la mitad de anchura de `textureMap`. Es decir, el contenido de `sphere` será el tamaño de una fotografía de la luna (ya que la imagen `textureMap` contiene dos fotografías de la luna, una junto a otra). A continuación, la propiedad `bitmapData` se llena con los datos de imagen utilizando su método `copyPixels()`. Los parámetros en la llamada al método `copyPixels()` indican varios puntos:

- El primer parámetro indica que los datos de imagen se copian desde `textureMap`.
- El segundo parámetro, una nueva instancia de Rectangle, especifica desde qué parte de `textureMap` se debe tomar la instantánea de la imagen; en este caso, la instantánea es un rectángulo que comienza en la esquina superior izquierda de `textureMap` (indicado mediante los dos primeros parámetros `Rectangle(): 0, 0`) y la altura y anchura de la instantánea del rectángulo coinciden con las propiedades `width` y `height` de `sphere`.
- El tercer parámetro, una nueva instancia de Point con los valores x e y de `0`, define el destino de los datos de píxel; en este caso, la esquina superior izquierda de (0, 0) de `sphere.bitmapData`.

Representado visualmente, el código copia los píxeles desde `textureMap` destacado en la siguiente imagen y los pega en `sphere`. Es decir, el contenido de BitmapData de `sphere` es la parte de `textureMap` resaltada aquí:



No obstante, recuerde que esto solo es el estado inicial de `sphere`; el contenido de la primera imagen que se copia en `sphere`.

Con la imagen de origen cargada y `sphere` creada, la tarea final realizada por el método `imageLoadComplete()` es configurar la animación. La animación se activa mediante una instancia de Timer denominada`rotationTimer`, que se crea y se inicia mediante el siguiente código:

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

En primer lugar, el código crea la instancia de Timer denominada `rotationTimer`; el parámetro transmitido al constructor `Timer()` indica que `rotationTimer` debe activar su evento `timer` cada 15 millisegundos. A continuación, se llama al método `addEventListener()`, especificando que cuando sucede el evento `timer` (`TimerEvent.TIMER`), se llama al método `rotateMoon()`. Finalmente, timer se inicia realmente llamando a su método `start()`.

Debido al modo en que se define `rotationTimer`, aproximadamente cada 15 millisegundos Flash Player llama al método `rotateMoon()` en la clase MoonSphere, que es donde sucede la animación de la luna. El código fuente del método `rotateMoon()` se presenta del siguiente modo:

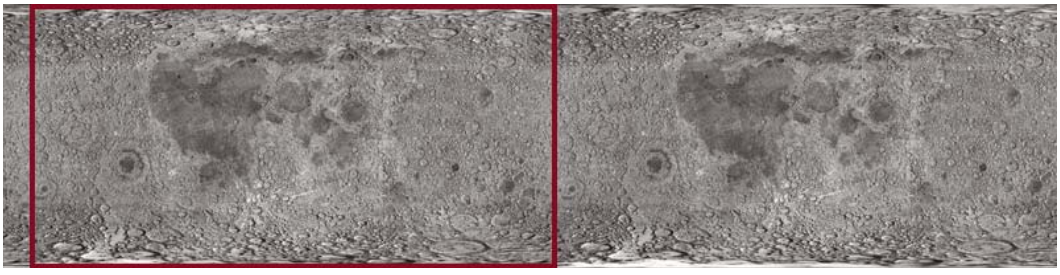
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.Data.copyPixels(textureMap,
                                   new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                   new Point(0, 0));

    event.updateAfterEvent();
}
```

El código realiza tres operaciones:

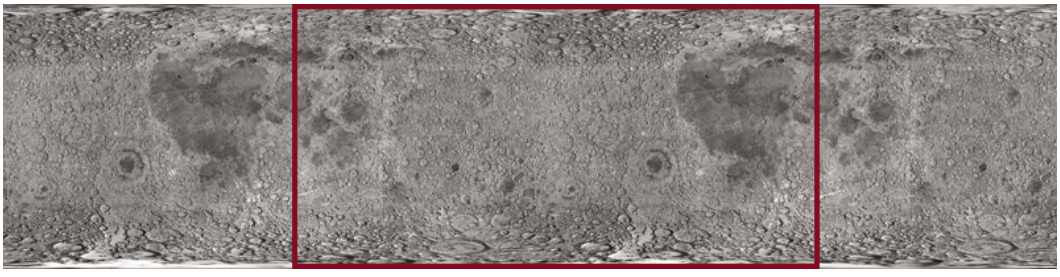
1 El valor de la variable `sourceX` (establecido inicialmente en 0) se incrementa en 1.

```
sourceX += 1;
```

Tal y como se puede observar, `sourceX` se usa para determinar la ubicación en `textureMap` desde donde los píxeles se copiarán en `sphere`, por lo que este código tiene el efecto de mover el rectángulo un píxel a la derecha en `textureMap`. Volviendo a la representación visual, tras varios ciclos de animación, el rectángulo de origen se habrá movido varios píxeles hacia la derecha, tal y como se muestra a continuación:

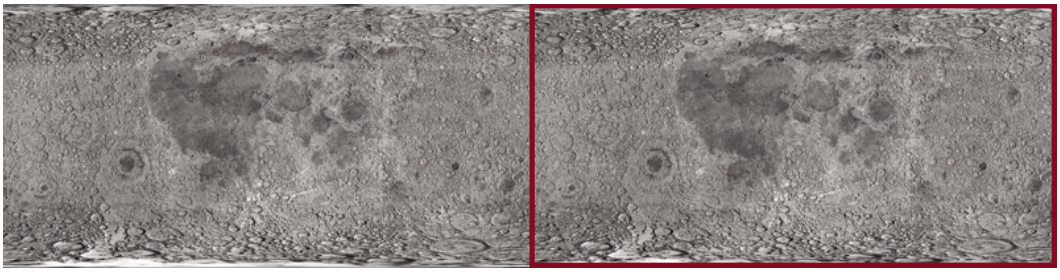


Tras varios ciclos más, el rectángulo se habrá movido incluso más lejos:



Este cambio constante y gradual en la ubicación desde la que se copian los píxeles es la clave en la animación. Con el movimiento lento y continuo de la ubicación de origen hacia la derecha, la imagen que se muestra en la pantalla en `sphere` parecer deslizarse continuamente hacia la izquierda. Por este motivo, es necesario que la imagen de origen (`textureMap`) tenga dos copias de la fotografía de la superficie de la luna. Debido a que el rectángulo se mueve continuamente hacia derecha, la mayor parte del tiempo no lo hace sobre una sola fotografía de la luna, sino que se superponen las dos fotografías.

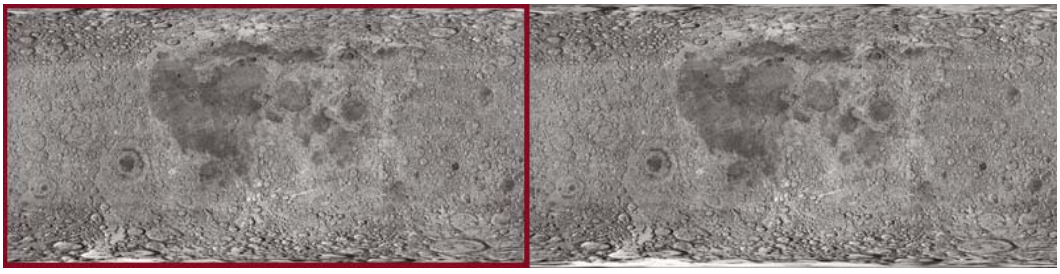
2 Con el movimiento lento del rectángulo de origen hacia la derecha, existe un problema. Finalmente el rectángulo llegará al borde derecho de `textureMap` y no dispondrá de píxeles de la fotografía de la luna para copiar en `sphere`:



En las siguientes líneas de código se aborda este problema:

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

El código comprueba si `sourceX` (borde izquierdo del rectángulo) ha alcanzado la mitad de `textureMap`. Si es así, vuelve a restablecer `sourceX` a 0, moviéndolo de nuevo hacia el borde izquierdo de `textureMap` y volviendo a iniciar el ciclo:



3 Con el valor adecuado y calculado de `sourceX`, el paso final en la creación de la animación consiste en copiar los nuevos píxeles del rectángulo de origen en `sphere`. El código que realiza esta operación es muy similar al que llenó en un principio `sphere` (descrito anteriormente); en este caso la única diferencia es que en la llamada al constructor `new Rectangle()`, el borde izquierdo del rectángulo se sitúa en `sourceX`:

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Recuerde que este código se llama repetidamente, cada 15 milisegundos. Debido a que la ubicación del rectángulo de origen se desplaza continuamente y los píxeles se copian en `sphere`, la apariencia en pantalla es que la imagen de la fotografía de la luna representada mediante `sphere` se desliza constantemente. Es decir, la luna parece girar de forma continua.

Creación de la apariencia esférica

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Obviamente, la luna es una esfera y no un rectángulo. Por lo tanto, en el ejemplo es necesario tomar la fotografía de la superficie de la luna rectangular, conforme se anima continuamente, y convertirla en una esfera. Esto implica dos pasos independientes: una máscara se utiliza para ocultar todo el contenido excepto una región circular de la fotografía de la superficie lunar y un filtro de mapa de desplazamiento se emplea para distorsionar la apariencia de la fotografía para hacerla parecer tridimensional.

En primer lugar, se usa una máscara con forma de círculo para ocultar todo el contenido del objeto MoonSphere excepto la esfera creada por el filtro. El siguiente código crea la máscara como una instancia de Shape y la aplica como la máscara de la instancia de MoonSphere:

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Se debe tener en cuenta que dado que MoonSphere es un objeto de visualización (se basa en la clase Sprite), la máscara se puede aplicar directamente a la instancia de MoonSphere utilizando su propiedad heredada `mask`.

Ocultar simplemente las partes de la fotografía utilizando una máscara con forma de círculo no es suficiente para crear un efecto de esfera giratoria con un aspecto realista. Debido al modo en que se tomó la fotografía de la superficie lunar, sus dimensiones no son proporcionales; las partes de la imagen que se encuentran más hacia la parte superior o inferior de la imagen están más distorsionadas y expandidas en comparación con las partes del ecuador. Para distorsionar la apariencia de la fotografía para hacerla parecer tridimensional, utilizaremos un filtro de mapa de desplazamiento.

Un filtro de mapa de desplazamiento es un tipo de filtro que se usa para distorsionar una imagen. En este caso, la fotografía de la luna se "distorsionará" para hacerla más realista, contrayendo la parte superior e inferior de la imagen horizontalmente, mientras el centro se deja sin cambio. Suponiendo que el filtro funcione en una parte con forma de cuadrado de la fotografía, la contracción de la parte superior e inferior pero no del centro convertirá el cuadrado en un círculo. Un efecto indirecto de la animación de esta imagen distorsionada es que el centro de la imagen parece moverse más lejos en la distancia de píxeles real que las áreas cercanas a la parte superior e inferior, lo que crea la ilusión de que el círculo es realmente un objeto tridimensional (una esfera).

El siguiente código se utiliza para crear el filtro de mapa de desplazamiento denominado `displaceFilter`:

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

El primer parámetro, `fisheyeLens`, se conoce como la imagen del mapa; en este caso se trata de un objeto BitmapData que se crea mediante programación. La creación de esta imagen se describe en "Creación de una imagen de mapa de bits estableciendo valores de píxel" en la página 272. Los demás parámetros describen la posición en la imagen filtrada en la que se debe aplicar el filtro, qué canales de color se utilizarán para controlar el efecto de desplazamiento y hasta qué punto afectarán al desplazamiento. Una vez creado el filtro de mapa de desplazamiento, se aplica a `sphere`, aún en el método `imageLoadComplete()`:

```
sphere.filters = [displaceFilter];
```

La imagen final, con máscara y filtro de mapa de desplazamiento aplicados, presenta el siguiente aspecto:

Con cada ciclo de la animación de rotación de la luna, el contenido de BitmapData de la esfera se sobrescribe con una nueva instantánea de los datos de la imagen de origen. No obstante, el filtro no necesita volver a aplicarse cada vez. Esto se debe a que el filtro se aplica en la instancia de Bitmap (el objeto de visualización) en lugar de en los datos del mapa de bits (información del píxel sin procesar). Recuerde que la instancia de Bitmap no son los datos de mapa de bits reales; se trata de un objeto de visualización que muestra los datos de mapa de bits en pantalla. Para utilizar una analogía, una instancia de Bitmap es como el proyector de diapositivas que se utiliza para visualizar diapositivas fotográficas en una pantalla y un objeto BitmapData es como la diapositiva real que se puede presentar mediante un proyector. Es posible aplicar un filtro directamente a un objeto BitmapData, lo que sería comparable a dibujar directamente en una diapositiva fotográfica para alterar la imagen. También se puede aplicar un filtro a cualquier objeto de visualización, incluyendo una instancia de Bitmap; esto sería como colocar un filtro frente a la lente del proyector de diapositivas para distorsionar el resultado que se muestra en pantalla (sin que la diapositiva original se vea alterada). Debido a que se puede acceder a los datos de mapa de bits mediante la propiedad bitmapData de una instancia de Bitmap, el filtro podría haberse aplicado directamente en los datos de mapa bits sin procesar. Sin embargo, en este caso resulta lógico aplicar el filtro al objeto de visualización Bitmap en lugar de a los datos de mapa de bits.

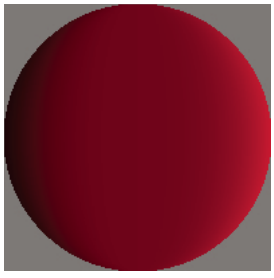
Para obtener información detallada sobre el uso del filtro de mapa de desplazamiento en ActionScript, consulte "Aplicación de filtros a objetos de visualización" en la página 279.

Creación de una imagen de mapa de bits estableciendo valores de píxel

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un aspecto importante del filtro de mapa de desplazamiento es que implica realmente dos imágenes. Una imagen, la imagen de origen, es la que se vé modificada por el filtro. En este ejemplo, la imagen de origen en la instancia de Bitmap denominada `sphere`. La otra imagen utilizada por el filtro se denomina imagen del mapa. La imagen del mapa no se visualiza realmente en pantalla. En cambio, el color de cada uno de sus píxeles se utiliza como una entrada en la función de desplazamiento; el color del píxel en una determinada coordenada x, y en la imagen del mapa determina el grado de desplazamiento (cambio físico de posición) que se aplica al píxel en dicha coordenada x, y en la imagen de origen.

Por lo tanto, para utilizar el filtro de mapa de desplazamiento para crear un efecto de esfera, el ejemplo necesita la imagen del mapa adecuada; una imagen con fondo gris y un círculo relleno con un degradado de un solo color (rojo) que pase horizontalmente de un tono oscuro a claro, tal y como se muestra a continuación:



Debido a que únicamente se utiliza un filtro y una imagen del mapa en este ejemplo, la imagen del mapa solo se crea una sola vez, en el método `imageLoadComplete()` (es decir, cuando la imagen externa termina de cargarse). La imagen del mapa, denominada `fisheyeLens`, se crea llamando al método `createFisheyeMap()` de la clase MoonSphere:

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Dentro del método `createFisheyeMap()`, la imagen del mapa se dibuja un píxel cada vez utilizando el método `setPixel()` de la clase BitmapData. El código completo para el método `createFisheyeMap()` se incluye a continuación, seguido de un análisis paso a paso sobre su funcionamiento:

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
```

```
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

En primer lugar, cuando se llama al método este recibe un parámetro, `radius`, que indica el radio de la imagen de forma circular que se va a crear. A continuación, el código crea el objeto BitmapData en el que se dibujará el círculo. Dicho objeto, llamado `result`, se transmite finalmente como el valor devuelto del método. Tal y como se muestra en el siguiente fragmento de código, la instancia de BitmapData `result` se crea con una anchura y una altura similares al diámetro del círculo, sin transparencia (`false` para el tercer parámetro) y rellena previamente con el color `0x808080` (gris medio):

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

A continuación, el código utiliza dos bucles para repetir cada píxel de la imagen. El bucle exterior recorre todas las columnas de la imagen de izquierda a derecha (utilizando la variable `i` para representar la posición horizontal del píxel que se está manipulando en ese momento), mientras que el bucle interior recorre todos los píxeles de la columna actual desde la parte superior a la inferior (con la variable `j` que representa la posición vertical del píxel actual). El código de los bucles (donde se omite el contenido del bucle interior) se muestra a continuación:

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

A medida que los bucles recorren los píxeles uno por uno, en cada píxel se calcula un valor (el valor del color del píxel en la imagen del mapa). Este proceso consta de cuatro pasos:

1. El código calcula la distancia del píxel actual desde el centro del círculo a través del eje x (`i - radius`). Dicho valor se divide por el radio para hacerlo un porcentaje del radio en lugar de una distancia absoluta (`(i - radius) / radius`). Ese valor del porcentaje se almacena en una variable denominada`pctX` y el valor equivalente para el eje y se calcula y almacena en la variable `pctY`, tal y como se indica en este código:

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2 Utilizado una fórmula trigonométrica estándar, el teorema de Pitágoras, la distancia lineal entre el centro del círculo y el punto actual se calcula a partir de `pctX` y `pctY`. Dicho valor se almacena en una variable denominada `pctDistance`, tal y como se muestra a continuación:

```
var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
```

3 A continuación, el código comprueba si el porcentaje de distancia es inferior a 1 (lo que significa el 100% del radio; es decir, si el píxel que se considera está en el radio del círculo). Si el píxel se encuentra dentro del círculo, se la asigna un valor de color calculado (se omite aquí, pero se describe en el paso 4); de lo contrario, no sucede nada más con ese píxel, de forma que su color se deja como gris medio predeterminado:

```
if (pctDistance < 1)
{
    ...
}
```

4 Para los píxeles que se encuentren dentro del círculo, se calcula un valor de color. El color final será una tonalidad de rojo que comprende desde negro (0% rojo) en el borde izquierdo del círculo hasta rojo claro (100%) en el borde derecho del círculo. El valor de color se calcula en un principio en tres partes (rojo, verde y azul), tal y como se indica a continuación:

```
red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
green = 0;
blue = 0;
```

Se debe tener en cuenta que únicamente la parte roja del color (variable `red`) dispone realmente de un valor. Los valores de azul y verde (variables `green` y `blue`) se muestran aquí para mayor claridad, pero se pueden omitir. El objetivo de este método es crear un círculo que incluya un degradado de rojo, por lo que los valores de verde o azul no son necesarios.

Una vez que se determinen los tres valores de color independientes, se combinan en un solo valor de color entero utilizando un algoritmo estándar de desplazamiento de bits, tal y como se muestra en este código:

```
rgb = (red << 16 | green << 8 | blue);
```

Finalmente, con el valor de color calculado, dicho valor se asigna al píxel actual utilizando el método `setPixel()` del objeto BitmapData de `result`, tal y como se indica a continuación:

```
result.setPixel(i, j, rgb);
```

Descodificación asíncrona de imágenes de mapa de bits

Flash Player 11 y posterior, Adobe AIR 2.6 y posterior

Cuando se trabaja con imágenes de mapa de bits, se puede descodificar de forma asíncrona y cargar imágenes de mapa de bits con el fin de mejorar el rendimiento percibido de la aplicación. La descodificación de una imagen de mapa de bits puede tardar el mismo tiempo que de la descodificación sincrónica de imágenes en muchos casos. Sin embargo, la imagen de mapa de bits se descodifica en un subproceso independiente antes de que el objeto `Loader` asociado envíe el evento `COMPLETE`. Por lo tanto, es posible descodificar de forma asíncrona imágenes de mayor tamaño tras cargarlas.

La clase `ImageDecodingPolicy` del paquete `flash.system`, permite especificar el esquema de carga de mapa de bits. El esquema de carga predeterminado es sincrónico.

Política de descodificación de mapas de bits	Esquema de carga de mapas de bits	Descripción
`ImageDecodingPolicy.ON_DEMAND`	Sincrónico	Las imágenes cargadas se descodifican cuando se accede a los datos de la imagen. Utilice esta política para descodificar imágenes más pequeñas. Esta política también se puede utilizar cuando la aplicación no confía en transiciones y efectos complejos.
`ImageDecodingPolicy.ON_LOAD`	Asincrónico	Las imágenes cargadas se descodifican durante la carta, antes de que se distribuya el evento `COMPLETE`. Ideal para imágenes de mayor tamaño (superiores a 10 MP). Cuando desarrolle aplicaciones móviles basadas en AIR con transiciones de página, utilice esta política de carga de mapas de bits para que el rendimiento percibido de la aplicación sea mejor.

Nota: *si el archivo que se está cargando es una imagen de mapa de bits y la política de descodificación utilizada es `ON_LOAD`, la imagen se descodifica de forma asíncrona antes de que se distribuya el evento `COMPLETE`.*

El siguiente código muestra el uso de la clase `ImageDecodingPolicy`:

```
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD
var loader:Loader = new Loader();
loader.load(new URLRequest("http://www.adobe.com/myimage.png"), loaderContext);
```

Aún se puede utilizar la descodificación `ON_DEMAND` con los métodos `Loader.load()` y `Loader.loadBytes()`. Sin embargo, todos los demás métodos que obtengan un objeto `LoaderContext` como argumento, omitirán todos los valores `ImageDecodingPolicy` transmitidos.

En el siguiente ejemplo se muestra la diferencia en la descodificación de una imagen de mapa de bits de forma sincrónica y asíncrona:

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.ImageDecodingPolicy;
    import flash.system.LoaderContext;

    public class AsyncTest extends Sprite
    {
        private var loaderContext:LoaderContext;
        private var loader:Loader;
        private var urlRequest:URLRequest;
        public function AsyncTest()
        {
            //Load the image synchronously
            loaderContext = new LoaderContext();
            //Default behavior.
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_DEMAND;
            loader = new Loader();
            loadImageSync();

            //Load the image asynchronously
            loaderContext = new LoaderContext();
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;
            loader = new Loader();
            loadImageASync();
        }

        private function loadImageASync():void{
            trace("Loading image asynchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onAsyncLoadComplete);
```

```
        }

        private function onAsyncLoadComplete(event:Event):void{
            trace("Async. Image Load Complete");
        }

        private function loadImageSync():void{
            trace("Loading image synchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onSyncLoadComplete);
        }

        private function onSyncLoadComplete(event:Event):void{
            trace("Sync. Image Load Complete");
        }
    }
}
```

Para ver una demostración del efecto de las distintas políticas de descodificación, consulte Thibaud Imbert: Asynchronous bitmap decoding in the Adobe Flash runtimes

Capítulo 14: Aplicación de filtros a objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Históricamente, la aplicación de efectos de filtro a imágenes de mapa de bits ha sido un campo reservado al software especializado de edición de imágenes, como Adobe Photoshop® y Adobe Fireworks®. ActionScript 3.0 incluye el paquete flash.filters, que contiene una serie de clases de filtros de efectos para mapas de bits. Estos efectos permiten a los desarrolladores aplicar mediante programación filtros a mapas de bits y objetos de visualización para lograr muchos de los efectos disponibles en las aplicaciones de manipulación de gráficos.

Fundamentos de la aplicación de filtros a los objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las formas de conformar la apariencia de una aplicación consiste en añadir efectos gráficos sencillos. Se puede proyectar una sombra tras una foto para lograr una ilusión tridimensional o un brillo en torno a un botón con objeto de mostrar que se encuentra activo. ActionScript 3.0 incluye diez filtros que se pueden aplicar a cualquier objeto de visualización o instancia de BitmapData. Entre los filtros incorporados se incluyen desde filtros básicos, como los de sombra e iluminado, hasta filtros complejos para crear efectos diversos, como los de mapa de desplazamiento y convolución.

Nota: *además de los filtros incorporados, también se pueden programar efectos y filtros personalizados con Pixel Bender. Consulte* *"Trabajo con sombreados de Pixel Bender" en la página 314.*

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que pueden aparecer al crear filtros:

Bisel Borde creado iluminando los píxeles de dos de los lados y oscureciendo al mismo tiempo los píxeles de los dos lados opuestos. Este efecto crea la apariencia de borde tridimensional. El efecto se suele utilizar para botones alzados o cincelados y gráficos similares.

Convolución Distorsión de los píxeles de una imagen obtenida combinando el valor de cada píxel con los valores de algunos o todos sus vecinos en diferentes proporciones.

Desplazamiento Movimiento, o deslizamiento, de los píxeles de una imagen a una nueva posición.

Matrix Cuadrícula de números utilizada para realizar determinados cálculos matemáticos aplicando los números de esta cuadrícula a distintos valores y combinando luego los resultados.

Más temas de ayuda

Paquete flash.filters

flash.display.DisplayObject.filters

flash.display.BitmapData.applyFilter()

Creación y aplicación de filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los filtros permiten aplicar toda una serie de efectos a los objetos de visualización y mapas de bits, desde las sombras hasta la creación de biseles y desenfoques. Cada filtro se define como una clase, de manera que al aplicar filtros se crean instancias de objetos de filtro, lo que es exactamente igual que crear cualquier otro objeto. Una vez creada una instancia de un objeto de filtro, se puede aplicar fácilmente a un objeto de visualización utilizando la propiedad `filters` del objeto o, en el caso de un objeto BitmapData, usando el método `applyFilter()`.

Crear un filtro

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear un objeto de filtro, basta con llamar al método constructor de la clase de filtro seleccionada. Por ejemplo, para crear un objeto DropShadowFilter, se usaría el siguiente código:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Aunque no se muestra aquí, el constructor `DropShadowFilter()` (como todos los constructores de clases de filtros) acepta varios parámetros opcionales que se pueden utilizar para personalizar la apariencia del efecto de filtro.

Aplicación de un filtro

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una vez construido un objeto de filtro, es posible aplicarlo a un objeto de visualización o a un objeto BitmapData; la forma de aplicarlo depende del objeto al que se vaya a aplicar.

Aplicación de un filtro a un objeto de visualización

Los efectos de filtro se aplican a los objetos de visualización mediante la propiedad `filters`. Dicha propiedad `filters` de los objetos de visualización es una instancia de Array cuyos elementos son los objetos de filtro aplicados al objeto de visualización. Para aplicar un único filtro a un objeto de visualización, se debe crear la instancia de filtro, añadirla a una instancia de Array y asignar ese objeto Array a la propiedad `filters` del objeto de visualización:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Si se desea asignar varios filtros al objeto, basta con añadir todos los filtros a la instancia de Array antes de asignarla a la propiedad `filters`. Se pueden añadir varios objetos a un elemento Array pasándolos como parámetros a su constructor. Por ejemplo, el siguiente código aplica un filtro de bisel y otro de iluminado al objeto de visualización creado con anterioridad:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Es posible crear un conjunto que contiene los filtros utilizando el constructor `new Array()` (según se muestra en los ejemplos anteriores) o se puede emplear la sintaxis literal de Array, encerrando los filtros entre corchetes (`[]`). Por ejemplo, esta línea de código:

```
var filters:Array = new Array(dropShadow, blur);
```

hace lo mismo que esta otra:

```
var filters:Array = [dropShadow, blur];
```

Al aplicar varios filtros a objetos de visualización, la aplicación se lleva a cabo de forma acumulativa y secuencial. Por ejemplo, si un conjunto de filtros tiene dos elementos, un filtro de bisel añadido en primer lugar y un filtro de sombra en segundo, el filtro de sombra se aplicará tanto al filtro de bisel como al objeto de visualización. Esto se debe a que el filtro de sombra está en segundo lugar en el conjunto de filtros. Para aplicar filtros de forma no acumulativa, es necesario aplicar cada filtro a una nueva copia del objeto de visualización.

Si solo se está aplicando un filtro o unos pocos a un objeto de visualización, es posible crear la instancia del filtro y asignarlo al objeto en una única sentencia. Por ejemplo, la siguiente línea de código aplica un filtro de desenfoque a un objeto de visualización llamado `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

El código anterior crea una instancia de Array utilizando su sintaxis literal (con corchetes), crea una instancia de BlurFilter como elemento de Array y asigna dicho Array a la propiedad `filters` del objeto de visualización llamado `myDisplayObject`.

Eliminación de filtros de un objeto de visualización

Eliminar todos los filtros de un objeto de visualización es tan sencillo como asignar un valor nulo a la propiedad `filters`:

```
myDisplayObject.filters = null;
```

Si se han aplicado varios filtros a un objeto y se desea eliminar solamente uno de ellos, es necesario seguir una serie de pasos para cambiar el conjunto de la propiedad `filters`. Para obtener más información, consulte "Posibles problemas al trabajar con filtros" en la página 282.

Aplicación de un filtro a un objeto BitmapData

Para aplicar un filtro a un objeto BitmapData es necesario utilizar el método `applyFilter()` del objeto BitmapData:

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

El método `applyFilter()` aplica un filtro a un objeto BitmapData de origen, dando lugar a una nueva imagen filtrada. Este método no modifica la imagen original. En vez de eso, el resultado de la aplicación del filtro a la imagen original se almacena en la instancia de BitmapData en la que se llama al método `applyFilter()`.

Funcionamiento de los filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al aplicar un filtro a un objeto de visualización, se crea en caché una copia del objeto original como un mapa de bits transparente.

Una vez que un filtro se ha aplicado a un objeto de visualización, el motor de ejecución almacena en caché el objeto como un mapa de bits mientras este mantiene una lista de filtros válida. Este mapa de bits de origen se usa luego como imagen original para todas las aplicaciones posteriores de efectos de filtros.

Normalmente, un objeto de visualización contiene dos mapas de bits: uno con el objeto de visualización de origen sin filtrar (original) y otro con la imagen final una vez filtrada. La imagen final es la que se utiliza en representaciones. Mientras el objeto de visualización no cambie, la imagen final no necesita actualización.

Posibles problemas al trabajar con filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen varias fuentes de confusión o problemas potenciales que hay que tener en cuenta al trabajar con filtros.

Filtros y caché de mapas de bits

Para aplicar un filtro a un objeto de visualización, debe estar activada la caché de mapa de bits para ese objeto. Cuando se aplica un filtro a un objeto de visualización cuya propiedad `cacheAsBitmap` se establece en `false`, la propiedad `cacheAsBitmap` del objeto se define automáticamente como `true`. Si posteriormente se eliminan todos los filtros del objeto de visualización, la propiedad `cacheAsBitmap` recupera el valor en que se estableció por última vez.

Cambio de filtros en tiempo de ejecución

Si un objeto de visualización ya tiene aplicados uno o varios filtros, no podrá cambiar este grupo de filtros agregando o eliminando filtros en el conjunto de la propiedad `filters`. En su lugar, para modificar el grupo de filtros aplicados o añadir otros nuevos, debe aplicar los cambios en un conjunto diferente y, a continuación, asignar dicho conjunto a la propiedad filters del objeto de visualización correspondiente a los filtros que se aplicarán al objeto. El modo más sencillo de realizar esta tarea consiste en leer el conjunto de la propiedad `filters` en una variable Array y aplicar las modificaciones en este conjunto temporal. A continuación, se puede volver a asignar este conjunto a la propiedad `filters` del objeto de visualización. En los casos más complejos, es posible que sea necesario mantener un conjunto maestro de filtros diferente. Los cambios se aplicarán a este conjunto maestro para después asignarlo a la propiedad `filters` del objeto de visualización tras cada modificación.

Añadir un filtro adicional

El siguiente código demuestra el proceso de añadir un filtro adicional a un objeto de visualización que ya tiene aplicados uno o varios filtros. Inicialmente se aplica un filtro de iluminado al objeto de visualización llamado `myDisplayObject`; luego, cuando se hace clic en el objeto, se llama a la función `addFilters()`. Mediante esta función se aplican dos filtros más a:`myDisplayObject`:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

Eliminación de un filtro de un grupo de filtros

Si un objeto de visualización tiene aplicados varios filtros y desea eliminar uno de ellos y seguir aplicando los demás, cópielos en un conjunto temporal, elimine de este el filtro que desee y vuelva a asignar el conjunto temporal a la propiedad `filters` del objeto de visualización. En la sección "Recuperación de valores y eliminación de elementos de conjunto" en la página 32 se describen varias formas de eliminar uno o varios elementos de un conjunto.

La forma más sencilla consiste en eliminar el filtro situado al principio del objeto (el último filtro que se le ha aplicado). Se puede utilizar el método `pop()` de la clase Array para eliminar el filtro del conjunto:

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

De igual modo, para eliminar el filtro situado al final (el primero aplicado al objeto) se puede utilizar el mismo código, substituyendo el método `shift()` de la clase Array en lugar del método `pop()`.

Para eliminar un filtro del centro de un conjunto de filtros (suponiendo que el conjunto tenga más de dos filtros), utilice el método `splice()`. Es preciso conocer el índice (la posición en el conjunto) del filtro que se desee eliminar. Por ejemplo, con el siguiente código se elimina el segundo filtro (el del índice 1) de un objeto de visualización:

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Determinación del índice de un filtro

Es preciso saber qué filtro que se va a eliminar del conjunto, de forma que se pueda identificar el índice correspondiente. Debe conocer (por el modo en que está diseñada la aplicación) o calcular el índice del filtro que se va a eliminar.

El mejor enfoque consiste en diseñar la aplicación de forma que el filtro que se desee eliminar se encuentre siempre en la misma posición en el grupo de filtros. Por ejemplo, si existe un único objeto de visualización que tiene aplicados un filtro de convolución y otro de sombra (en este orden) y desea eliminar este último y mantener el anterior, el filtro se debe situar en una posición conocida (la superior) de forma que se pueda determinar con antelación el método de Array que hay que utilizar (en este caso, `Array.pop()` para eliminar el filtro de sombra).

Si el filtro que desea eliminar siempre es de un tipo específico, pero no se sitúa siempre en la misma posición del grupo de filtros, puede comprobar el tipo de datos de cada filtro del conjunto para determinar cuál hay que eliminar. Por ejemplo, el siguiente código identifica el filtro de iluminado dentro de un grupo de filtros y lo elimina de este.

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

En un caso más complejo, por ejemplo, cuando el filtro que se va a eliminar se selecciona en tiempo de ejecución, la mejor solución consiste en crear un copia persistente e independiente del conjunto de filtros que sirve como la lista maestra de filtros. Cada vez que modifique el grupo de filtros, debe cambiar la lista maestra y, a continuación, aplicar el conjunto de filtros como la propiedad `filters` del objeto de visualización.

Por ejemplo, en el siguiente listado de código se aplican varios filtros de convolución a un objeto de visualización para crear diferentes efectos visuales y, en un punto posterior de la aplicación, se elimina uno de estos filtros y se mantienen los demás. En este caso, el código conserva una copia maestra del conjunto de filtros, así como una referencia al filtro que se va a eliminar. La identificación y eliminación del filtro específico es un proceso similar al anterior, excepto que en lugar de realizar una copia temporal del conjunto de filtros, se manipula la copia maestra y se aplica después al objeto de visualización.

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

En este enfoque (cuando se compara una referencia a un filtro almacenado con los elementos del conjunto de filtros para determinar el filtro que se va a eliminar), se *debe* crear una copia independiente de conjunto de filtros — el código no funciona si se compara la referencia al filtro almacenado con los elementos de un conjunto temporal copiado a partir de la propiedad `filters` del objeto de visualización. El motivo es que al asignar un conjunto a la propiedad `filters`, el motor de ejecución realiza internamente una copia de cada objeto de filtro del conjunto. Y son estas copias (en lugar de los objetos originales) las que se aplican al objeto de visualización, de forma que al leer la propiedad `filters` en un conjunto temporal, este contiene referencias a los objetos de filtro copiados en lugar de las referencias a los objetos de filtro originales. En consecuencia, si en el ejemplo anterior se intenta determinar el índice de `filterToRemove` por comparación con los filtros de un conjunto de filtros temporal, no se obtendrá ninguna coincidencia.

Filtros y transformaciones de objetos

Ninguna región filtrada (una sombra, por ejemplo) que se encuentre fuera del recuadro de delimitación rectangular de un objeto de visualización se considera parte de la superficie a efectos de la detección de colisiones (determinar si una instancia se solapa o corta con otra instancia). Dado que los métodos de detección de colisiones de la clase DisplayObject se basan en vectores, no es posible llevar a cabo esta operación sobre un resultado que es un mapa de bits. Por ejemplo, si se aplica un filtro de bisel a una instancia de botón, la detección de colisiones no estará disponible en la parte biselada de la instancia.

Los filtros no son compatibles con el escalado, la rotación ni el sesgo. Si el objeto de visualización al que se le ha aplicado un filtro es escalado (si `scaleX` y `scaleY` no tienen un valor de 100%), el efecto de filtro no se escalará con la instancia. Esto quiere decir que la forma original de la instancia rotará, se escalará o sesgará, pero el filtro no lo hará junto con la instancia.

Se puede animar una instancia con un filtro para crear efectos realistas o anidar instancias y usar la clase BitmapData para animar filtros y así lograr este efecto.

Filtros y objetos de mapas de bits

Al aplicar un filtro a un objeto BitmapData, la propiedad `cacheAsBitmap` toma automáticamente el valor `true`. De este modo, el filtro se aplica en realidad a la copia del objeto en lugar de al original.

Esta copia se coloca a continuación en la visualización principal (sobre el objeto original) tan cerca como sea posible del píxel más cercano. Si los límites del mapa de bits original cambian, la copia del mapa de bits con el filtro se vuelve a crear de nuevo en lugar de estirarla o distorsionarla.

Si se quitan todos los filtros de un objeto de visualización, se restablece el valor que la propiedad `cacheAsBitmap` tenía antes de aplicarse el filtro.

Filtros de visualización disponibles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye diez clases de filtros que se pueden aplicar a objetos de visualización y a objetos BitmapData:

- Filtro de biselado (clase BevelFilter)
- Filtro de desenfoque (clase BlurFilter)
- Filtro de sombra (clase DropShadowFilter)
- Filtro de iluminado (clase GlowFilter)
- Filtro de bisel degradado (clase GradientBevelFilter)
- Filtro de iluminado degradado (clase GradientGlowFilter)
- Filtro de matriz de colores (clase ColorMatrixFilter)
- Filtro de convolución (clase ConvolutionFilter)
- Filtro de mapa de desplazamiento (clase DisplacementMapFilter)
- Filtro de sombreado (clase ShaderFilter)

Los seis primeros son filtros sencillos que se pueden utilizar para crear un efecto específico con posibilidades de personalización limitadas del efecto disponible. Todos ellos se pueden aplicar mediante ActionScript, además de poderse aplicar a objetos en Flash Professional desde el panel Filtros. Por lo tanto, incluso si se usa ActionScript para aplicar filtros, se puede utilizar la interfaz visual, si se dispone de Flash Professional, para probar de forma rápida los distintos filtros y configuraciones y, de ese modo, probar la mejor forma de lograr el efecto deseado.

Los últimos cuatro filtros solo están disponibles en ActionScript. Estos filtros (matriz de colores, convolución, mapa de desplazamiento y sombreado) son mucho más flexibles en los tipos de efectos para cuya creación se pueden utilizar. En lugar de ser optimizados para un único efecto, ofrecen potencia y flexibilidad. Por ejemplo, seleccionando distintos valores para su matriz, el filtro de convolución se puede utilizar para crear efectos de desenfoque, relieve, nitidez, búsqueda de bordes de color, transformaciones y mucho más.

Cada uno de ellos, ya sea sencillo o complejo, se puede personalizar usando sus propiedades. En general, existen dos posibilidades a la hora de definir las propiedades de un filtro. Todos los filtros permiten establecer sus propiedades pasando los valores de los parámetros al constructor del objeto de filtro. Como alternativa, tanto si se definen las propiedades del filtro pasando parámetros como si no, es posible ajustar los filtros más adelante estableciendo valores para las propiedades del objeto de filtro. En la mayor parte de los listados de código, las propiedades se definen directamente para que el ejemplo sea más fácilmente comprensible. No obstante, se suele poder obtener el mismo resultado con menos líneas de código pasando los valores como parámetros en el constructor del objeto de filtro. Para obtener más información sobre cada filtro, sus propiedades y sus parámetros de constructor, consulte el paquete flash.filters en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Filtro de bisel

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase BevelFilter permite añadir un borde tridimensional en bisel al objeto filtrado. Este filtro hace que las esquinas o bordes duros del objeto adquieran una apariencia cincelada o biselada.

Las propiedades de la clase BevelFilter permiten personalizar la apariencia del bisel. Se pueden definir los colores de iluminación y sombra, el desenfoque de los bordes del bisel, los ángulos de biselado y la colocación de los bordes del bisel; incluso se puede crear un efecto extractor.

En el siguiente ejemplo se carga una imagen externa y se le aplica un filtro de bisel.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

Filtro de desenfoque

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase BlurFilter difumina o desenfoca un objeto de visualización y su contenido. Los efectos de desenfoque resultan útiles para dar la impresión de que un objeto no está bien enfocado o para simular un movimiento rápido, como en el desenfoque por movimiento. Si se define la propiedad `quality` del filtro de desenfoque en un nivel bajo, se puede simular un efecto de lente ligeramente desenfocada. Si la propiedad `quality` se establece en un nivel alto, da lugar a un efecto de desenfoque suave similar a un desenfoque gaussiano.

En el siguiente ejemplo se crea un objeto circular usando el método `drawCircle()` de la clase Graphics y se le aplica un efecto de desenfoque:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

Filtro de sombra

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las sombras dan la impresión de que existe una fuente de luz independiente situada sobre el objeto de destino. La posición e intensidad de esta fuente de luz se puede modificar para originar toda una gama de distintos efectos de sombra.

La clase DropShadowFilter usa un algoritmo similar al del filtro de desenfoque. La principal diferencia es que el filtro de sombra tiene unas pocas propiedades más que se pueden modificar para simular los distintos atributos de la fuente de luz (como alfa, color, desplazamiento y brillo).

El filtro de sombra también permite aplicar opciones de transformación personalizadas sobre el estilo de la sombra, incluidas la sombra interior o exterior y el modo extractor (también conocido como silueta).

El siguiente código crea una un objeto Sprite cuadrado y le aplica un filtro de sombra:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

Filtro de iluminado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase GlowFilter aplica un efecto de iluminación a los objetos de visualización, de forma que parece que bajo el objeto brilla una luz que lo ilumina suavemente.

El filtro de iluminado es similar al del sombra, pero incluye propiedades para modificar la distancia, el ángulo y el color de la fuente de luz con las que es posible lograr diversos efectos. GlowFilter también dispone de varias opciones para modificar el estilo de la iluminación, entre ellos el brillo interior y exterior y el modo extractor.

El siguiente código crea una cruz usando la clase Sprite y le aplica un filtro de iluminado:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

Filtro de bisel degradado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase GradientBevelFilter permite aplicar un efecto de bisel mejorado a los objetos de visualización o a los objetos BitmapData. Al utilizar un degradado de color en el bisel se mejora enormemente su profundidad espacial, dando a los bordes una apariencia en 3D más realista.

El siguiente código crea un objeto rectangular empleando el método `drawRect()` de la clase Shape y le aplica un filtro de bisel degradado.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

Filtro de iluminado degradado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase GradientGlowFilter permite aplicar un efecto de iluminación mejorada a los objetos de visualización o a los objetos BitmapData. Este efecto proporciona un mayor control sobre el color de la iluminación y, de este modo, permite lograr un efecto de brillo más realista. Además, mediante el filtro de iluminado degradado, es posible aplicar un brillo degradado al borde interior, exterior o superior de un objeto.

En el siguiente ejemplo se dibuja un círculo en el escenario y se le aplica un filtro de iluminado degradado. A medida que se mueve el ratón hacia la derecha y abajo, la cantidad de desenfoque aumenta en dirección horizontal y vertical respectivamente. Además, cuando se hace clic en el escenario, la fuerza del desenfoque aumenta.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

Ejemplo: Combinación de filtros básicos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el siguiente ejemplo de código se utilizan varios filtros básicos combinados con un temporizador para crear una simulación animada de la acción repetitiva de un semáforo.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
```

```
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees,
colors, alphas, ratios, blurX, blurY, strength, quality, type, knockout);
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true,
false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

Filtro de matriz de colores

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ColorMatrixFilter se utiliza para manipular los valores de color y alfa del objeto filtrado. Esto permite crear cambios de saturación, rotaciones de matiz (cambio de una paleta de una gama de colores a otra), cambios de luminancia a alfa y otros efectos de manipulación del color usando los valores de un canal de color y aplicándolos potencialmente a otros canales.

Conceptualmente, el filtro recorre de uno en uno los píxeles de la imagen de origen y separa cada uno de ellos en sus componentes rojo, verde, azul y alfa. A continuación, multiplica los valores indicados en la matriz de colores por cada uno de estos valores, sumando los resultados para determinar el valor del color resultante que se mostrará en la pantalla para cada píxel. La propiedad `matrix` del filtro es una conjunto de 20 números que se utiliza para calcular el color final. Para obtener información sobre el algoritmo específico utilizado para calcular los valores de color, consulte la entrada que describe la propiedad `matrix` de la clase ColorMatrixFilter en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Filtro de convolución

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ConvolutionFilter se puede utilizar para aplicar una amplia gama de transformaciones gráficas, como efectos de desenfoque, detección de bordes, nitidez, relieve y biselado, a los objetos de visualización o a los objetos BitmapData.

Conceptualmente, el filtro de convolución recorre los píxeles de la imagen de origen de uno en uno y determina su color final basándose en el valor de cada píxel y de los que lo rodean. Una matriz, especificada como un conjunto de valores numéricos, indica la intensidad con la que cada uno de los píxeles circundantes afecta al valor final.

A modo de ejemplo, se puede tomar el tipo de matriz usado con mayor frecuencia, la matriz de tres por tres, que consta de nueve valores:

```
N   N   N
N   P   N
N   N   N
```

Si el filtro de convolución se aplica a un píxel determinado, este consulta el valor de color del píxel en sí ("P" en el ejemplo), además de los valores de los píxeles que lo rodean (llamados "N" en el ejemplo). No obstante, al definir los valores de la matriz, se especifica la prioridad que algunos píxeles tendrán a la hora de afectar a la imagen resultante.

Por ejemplo, la siguiente matriz, aplicada mediante un filtro de convolución, dejará la imagen exactamente igual que estaba:

```
0   0   0
0   1   0
0   0   0
```

La razón por la que la imagen no se ve modificada es porque el valor del píxel original tiene una fuerza relativa de 1 para determinar el color final del píxel, mientras que todos los que lo rodean tienen una fuerza relativa de 0, lo que significa que sus colores no afectan a la imagen final.

Del mismo modo, la siguiente matriz hará que los píxeles de una imagen se desplacen un píxel hacia la izquierda:

```
0   0   0
0   0   1
0   0   0
```

Hay que tener en cuenta que, en este caso, el píxel en sí no tiene ningún efecto sobre el valor final del píxel mostrado en esa posición en la imagen final, ya que solo se usa el valor del píxel de la derecha para determinar el valor resultante de ese píxel.

En ActionScript, la matriz se crea como una combinación de una instancia de Array que contiene los valores y dos propiedades que especifican el número de filas y columnas de la matriz. En el siguiente ejemplo se carga una imagen y, una vez cargada, se le aplica un filtro de convolución utilizando la matriz del listado anterior:

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                    0, 0, 1,
                                    0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

Algo que no resulta obvio en este código es el efecto que se produce al utilizar valores diferentes a 0 y 1 en la matriz. Por ejemplo, la misma matriz con el número 8 en lugar de 1 en la posición de la derecha, realiza la misma acción (desplazar los píxeles a la izquierda) y, además, afecta a los colores de la imagen, haciéndolos 8 veces más brillantes. Esto se debe a que el color final de los píxeles se calcula multiplicando los valores de matriz por los colores de los píxeles originales, sumando los valores y dividiendo por el valor de la propiedad `divisor` del filtro. Hay que tener en cuenta que en el código de ejemplo, la propiedad `divisor` tiene el valor 1. En general, si se desea que el brillo de los colores permanezca aproximadamente igual que en la imagen original, el divisor debe ser igual a la suma de los valores de la matriz. De este modo, en el caso de una matriz cuyos valores suman 8 y el divisor vale 1, la imagen resultante será aproximadamente 8 veces más brillante que la original.

Si bien el efecto de esta matriz no es demasiado apreciable, es posible utilizar otros valores en la matriz para crear diversos efectos. A continuación se ofrecen varios conjuntos estándar de valores de matriz con los que obtener distintos efectos mediante una matriz de tres por tres:

- Desenfoque básico (divisor 5):

```
0 1 0
1 1 1
0 1 0
```

- Nitidez (divisor 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Detección de bordes (divisor 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Efecto de relieve (divisor 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

  Cabe destacar que en la mayor parte de estos efectos el divisor es 1. Esto se debe a que los valores negativos de la matriz, sumandos a los positivos, dan un valor de 1 (0 en el caso de la detección de bordes, pero el valor de la propiedad `divisor` no puede ser 0).

Filtro de mapa de desplazamiento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase DisplacementMapFilter usa los valores de los píxeles de un objeto BitmapData (conocido como imagen del mapa de desplazamiento) para llevar a cabo un efecto de desplazamiento sobre un nuevo objeto. La imagen del mapa de desplazamiento suele ser distinta al objeto de visualización o a la instancia de BitmapData a los que se va a aplicar el filtro. En un efecto de desplazamiento, los píxeles de la imagen filtrada se desplazan, es decir, cambian su posición en mayor o menor medida con respecto a la imagen original. Este filtro se puede utilizar para crear un efecto de desplazamiento, combado o moteado.

La ubicación y la cantidad de desplazamiento aplicadas a un píxel determinado vienen dadas por el valor del color de la imagen del mapa de desplazamiento. Cuando se trabaja con el filtro, además de especificar la imagen del mapa, es necesario indicar los siguientes valores para controlar la forma en la que se calcula el desplazamiento a partir de la imagen del mapa:

- Punto del mapa: la posición de la imagen filtrada en la que se aplicará la esquina superior izquierda del filtro de desplazamiento. Se puede usar este valor para aplicar el filtro solo a una parte de la imagen.
- Componente X: el canal de color de la imagen del mapa que afectará a la posición x de los píxeles.
- Componente Y: el canal de color de la imagen del mapa que afectará a la posición y de los píxeles.
- Escala X: un valor multiplicador que especifica la intensidad del desplazamiento en el eje x.
- Escala Y: un valor multiplicador que especifica la intensidad del desplazamiento en el eje y.
- Modo de filtro: determina la operación que se debe realizar en los espacios vacíos dejados por los píxeles desplazados. Las opciones, definidas como constantes en la clase DisplacementMapFilterMode, son mostrar los píxeles originales (modo de filtro `IGNORE`), cruzar los píxeles del otro lado de la imagen (modo de filtro `WRAP`, que es el predeterminado), usar el píxel desplazado más cercano (modo de filtro `CLAMP`) o rellenar los espacios con un color (modo de filtro `COLOR`).

Para entender el funcionamiento del filtro de mapa de desplazamiento, se puede considerar el siguiente ejemplo. En el código que aparece a continuación se carga una imagen y, cuando termina la carga, se centra en el escenario y se le aplica un filtro de mapa de desplazamiento que hace que los píxeles de toda la imagen se desplacen horizontalmente hacia la izquierda.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

Las propiedades utilizadas para definir el desplazamiento son las siguientes:

- Mapa de bits del mapa: el mapa de bits del desplazamiento es una nueva instancia de BitmapData creada por el código. Sus dimensiones coinciden con las de la imagen cargada (de modo que el desplazamiento se aplica a toda la imagen). Tiene un relleno sólido de píxeles rojos.
- Punto del mapa: este valor está ajustado al punto 0, 0, lo que hace que el desplazamiento se aplique a toda la imagen.
- Componente X: este valor está ajustado a la constante `BitmapDataChannel.RED`, lo que significa que el valor rojo del mapa de bits del mapa determinará la cantidad de desplazamiento de los píxeles (cuánto se moverán) en el eje x.
- Escala X: este valor está ajustado a 250. La cantidad total de desplazamiento (teniendo en cuenta que la imagen del mapa es completamente roja) solo movería la imagen una distancia muy pequeña (aproximadamente medio píxel), de modo que si este valor se ajustase a 1, la imagen solo se desplazaría 0,5 píxeles horizontalmente. Al ajustarlo a 250, la imagen se mueve aproximadamente 125 píxeles.

Esta configuración hace que los píxeles de la imagen filtrada se desplacen 250 píxeles hacia la izquierda. La dirección (izquierda o derecha) y la cantidad de desplazamiento se basan en el valor de color de los píxeles de la imagen del mapa. Conceptualmente, el filtro recorre de uno en uno los píxeles de la imagen filtrada (al menos los píxeles de la región en la que se aplica el filtro, que, en este caso, son todos) y lleva a cabo el siguiente proceso con cada píxel:

1. Buscan el píxel correspondiente en la imagen del mapa. Por ejemplo, cuando el filtro calcula la cantidad de desplazamiento para el píxel de la esquina superior izquierda de la imagen filtrada, consulta el píxel de la esquina superior izquierda de la imagen del mapa.
2. Determinan el valor del canal de color especificado en el píxel del mapa. En este caso, el canal de color del componente x es el rojo, de modo que el filtro comprueba qué valor tiene el canal rojo de la imagen del mapa en el píxel en cuestión. Dado que la imagen del mapa es de color rojo sólido, el canal rojo del píxel es 0xFF, o 255. Este será el valor que se use como desplazamiento.
3. Comparan el valor de desplazamiento con el valor "medio" (127, que es el valor intermedio entre 0 y 255). Si el valor de desplazamiento es menor que el valor medio, el píxel se desplaza en dirección positiva (hacia la derecha para el desplazamiento en el eje x y hacia abajo para el desplazamiento en el eje y). Por otro lado, si el valor de desplazamiento es superior al valor medio (como en el ejemplo), el píxel se desplaza en dirección negativa (hacia la izquierda para el desplazamiento en el eje x y hacia arriba para el desplazamiento en el eje y). Para ser más precisos, el filtro resta a 127 el valor de desplazamiento y el resultado (positivo o negativo) es la cantidad relativa de desplazamiento que se aplica.
4. Finalmente, determinan la cantidad de desplazamiento real calculando qué porcentaje del desplazamiento completo representa el valor del desplazamiento relativo. En este caso, el rojo sólido significa un desplazamiento del 100%. Ese porcentaje se multiplica por el valor de escala x o escala y a fin de establecer el número de píxeles de desplazamiento que se aplicará. En este ejemplo, la cantidad de desplazamiento equivale a 100% veces un multiplicador de 250 (aproximadamente 125 píxeles a la izquierda).

Dado que no se han especificado valores para el componente y la escala y, se utilizan los valores predeterminados (que no causan ningún desplazamiento), por eso la imagen no se mueve en dirección vertical.

En el ejemplo se utiliza el modo de filtro predeterminado, `WRAP`, de modo que cuando los píxeles se desplazan hacia la izquierda, el espacio vacío de la derecha se llena con los píxeles que se han desplazado fuera del borde izquierdo de la imagen. Se puede experimentar con este valor para ver los distintos efectos. Por ejemplo, si se añade la siguiente línea a la parte del código en la que se definen las propiedades del desplazamiento (antes de la línea `loader.filters = [displacementMap]`), la imagen tendrá la apariencia de haber sido arrastrada por el escenario:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

A continuación se muestra un ejemplo más complejo, con un listado que utiliza un mapa de desplazamiento para crear un efecto de lupa sobre una imagen:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
BitmapDataChannel.BLUE);
```

```
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

El código genera primero dos círculos degradados, que se combinan para formar la imagen del mapa de desplazamiento. El círculo rojo crea el desplazamiento en el eje x (`xyFilter.componentX = BitmapDataChannel.RED`), y el azul crea el desplazamiento en el eje y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Para entender mejor la apariencia de la imagen del mapa de desplazamiento, el código añade los círculos originales y el círculo combinado que sirve de imagen del mapa en la parte inferior de la pantalla.

A continuación, el código carga una imagen y, a medida que se mueve el ratón, aplica el filtro de desplazamiento a la parte de la imagen que se encuentra bajo este. Los círculos degradados que se utilizan como imagen del mapa de desplazamiento hacen que la región desplazada se aleje del puntero. Hay que tener en cuenta que las regiones grises del mapa de desplazamiento no causan ningún movimiento. El color gris es `0x7F7F7F`. Los canales rojo y azul de esa tonalidad de gris coinciden exactamente con la tonalidad media de esos canales de color, de manera que no hay desplazamiento en las zonas grises de la imagen del mapa. Del mismo modo, en el centro del círculo tampoco hay desplazamiento. Si bien el color de esa región no es gris, los canales rojo y azul de ese color son idénticos a los canales rojo y azul del gris medio y, puesto que el rojo y el azul son los colores que causan el desplazamiento, no se produce ninguno.

Filtro de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La clase ShaderFilter permite utilizar un efecto de filtro personalizado definido como sombreado de Pixel Bender. Dado que el efecto de filtro se escribe como un sombreado de Pixel Bender, el efecto se puede personalizar completamente. El contenido filtrado se pasa al sombreado en forma de entrada de imagen y el resultado de la operación del sombreado se convierte en el resultado del filtro.

Nota: *el filtro Shader está disponible en ActionScript a partir de Flash Player 10 y Adobe AIR 1.5.*

Para aplicar un filtro de sombreado a un objeto, primero debe crear una instancia de Shader que represente al sombreado de Pixel Bender que se va a utilizar. Para obtener información detallada sobre el procedimiento de creación de una instancia de Shader y el modo de especificar los valores de los parámetros y las imágenes de entrada, consulte "Trabajo con sombreados de Pixel Bender" en la página 314.

Al utilizar un sombreado a modo de filtro, es preciso recordar tres cosas:

- El sombreado se debe definir de forma que acepte al menos una imagen de entrada.

- El objeto filtrado (el objeto de visualización o BitmapData al que se aplica el filtro) se pasa al sombreado como el primer valor de imagen de entrada. Por este motivo, no se debe especificar manualmente un valor para la primera imagen de entrada.
- Si el sombreado define varias imágenes de entrada, las entradas adicionales se deben especificar manualmente (es decir, es preciso establecer la propiedad `input` de cualquier instancia de ShaderInput que pertenezca a la instancia de Shader).

Una vez disponga de un objeto Shader para el sombreado, cree una instancia de ShaderFilter. Este es el objeto de filtro real que se utiliza como cualquier otro filtro. Para crear una clase ShaderFilter que utilice un objeto Shader, llame al constructor de `ShaderFilter()` y pase el objeto Shader en forma de argumento, tal y como se muestra en este listado:

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Para ver un ejemplo completo del uso de un filtro de sombreado, consulte "Uso de un sombrado como filtro" en la página 332.

Ejemplo de filtrado de objetos de visualización: Filter Workbench

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Filter Workbench proporciona una interfaz de usuario con la que es posible aplicar distintos filtros a imágenes y otro contenido visual y ver el código resultante que se puede utilizar para generar el mismo efecto en ActionScript. Además de ofrecer una herramienta para la experimentación con filtros, esta aplicación ilustra las siguientes técnicas:

- Creación de instancias de diferentes filtros
- Aplicación de varios filtros a un objeto de visualización

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Filter Workbench se encuentran en la carpeta Samples/FilterWorkbench. La aplicación consta de los siguientes archivos:

Archivo	Descripción
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Clase que proporciona la principal funcionalidad de la aplicación, incluido el cambio de contenido al que se aplican los filtros y la aplicación de filtros al contenido.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Interfaz que define los métodos comunes implementados por cada una de las clases Factory de los filtros. Esta interfaz define la funcionalidad común que utiliza la clase FilterWorkbenchController para interactuar con cada clase Factory de los filtros.
En la carpeta com/example/programmingas3/filterWorkbench/: BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	Conjunto de clases, cada una de las cuales implementa la interfaz IFilterFactory. Cada una de estas clases proporciona la funcionalidad de crear y definir valores para un único tipo de filtro. Los paneles de propiedad de filtro de la aplicación utilizan estas clases Factory para crear instancias de sus filtros específicos, los cuales son recuperados y aplicados al contenido de la imagen por la clase FilterWorkbenchController.
com/example/programmingas3/filterWorkbench/IFilterPanel.as	Interfaz que define los métodos comunes implementados por las clases que, a su vez, definen los paneles de interfaz de usuario utilizados para manipular los valores de filtro en la aplicación.
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Clase de utilidad que incluye un método para convertir un valor de color numérico a un formato de cadena hexadecimal.
com/example/programmingas3/filterWorkbench/GradientColor.as	Clase que funciona como un objeto de valor, combinando en un único objeto los tres valores (color, alfa y proporción) asociados a cada color en las clases GradientBevelFilter y GradientGlowFilter.
Interfaz de usuario (Flex)	
FilterWorkbench.mxml	El archivo principal que define la interfaz de usuario de la aplicación.
flexapp/FilterWorkbench.as	Clase que proporciona la funcionalidad para la interfaz de usuario principal de la aplicación; esta clase se utiliza como la clase de código subyacente para el archivo MXML de la aplicación.

Archivo	Descripción
En la carpeta flexapp/filterPanels: BevelPanel.mxml BlurPanel.mxml ColorMatrixPanel.mxml ConvolutionPanel.mxml DropShadowPanel.mxml GlowPanel.mxml GradientBevelPanel.mxml GradientGlowPanel.mxml	Conjunto de componentes MXML que proporciona la funcionalidad en cada panel utilizado para ajustar las opciones para un filtro único.
flexapp/ImageContainer.as	Un objeto de visualización que funciona como contenedor para la imagen cargada de la pantalla.
flexapp/controls/BGColorCellRenderer.as	Procesador de celdas personalizado que se utiliza para cambiar el color de fondo de una celda en el componente DataGrid.
flexapp/controls/QualityComboBox.as	Control personalizado que define un cuadro combinado que permite ajustar la calidad en varios paneles de filtros.
flexapp/controls/TypeComboBox.as	Control personalizado que define un cuadro combinado que permite ajustar el tipo en varios paneles de filtros.
Interfaz de usuario (Flash)	
FilterWorkbench.fla	El archivo principal que define la interfaz de usuario de la aplicación.
flashapp/FilterWorkbench.as	Clase que proporciona la funcionalidad para la interfaz de usuario principal de la aplicación; esta clase se utiliza como la clase de documento para el archivo FLA de la aplicación.
En la carpeta flashapp/filterPanels: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	Conjunto de clases que proporciona la funcionalidad para cada panel utilizado para ajustar las opciones para un filtro único. Para cada clase, existe además un símbolo MovieClip asociado en la biblioteca del archivo FLA principal de la aplicación, cuyo nombre coincide con el de la clase (por ejemplo, el símbolo “BlurPanel” está relacionado con la clase definida en BlurPanel.as). Los componentes que conforman la interfaz de usuario se ubican y designan de acuerdo con estos símbolos.
flashapp/ImageContainer.as	Un objeto de visualización que funciona como contenedor para la imagen cargada de la pantalla.
flashapp/BGColorCellRenderer.as	Procesador de celdas personalizado que se utiliza para cambiar el color de fondo de una celda en el componente DataGrid.

Archivo	Descripción
flashapp/ButtonCellRenderer.as	Procesador de celdas personalizado que se utiliza para incluir un componente de botón en una celda del componente DataGrid.
Contenido de imagen filtrado	
com/example/programmingas3/filterWorkbench/ImageType.as	Esta clase sirve como objeto de valor que contiene el tipo y el URL del archivo de imagen única en el que la aplicación puede cargar y aplicar los filtros. La clase también incluye un conjunto de constantes que representan los archivos de imagen disponibles.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Imágenes y otro contenido visual al que se aplican los filtros en la aplicación.

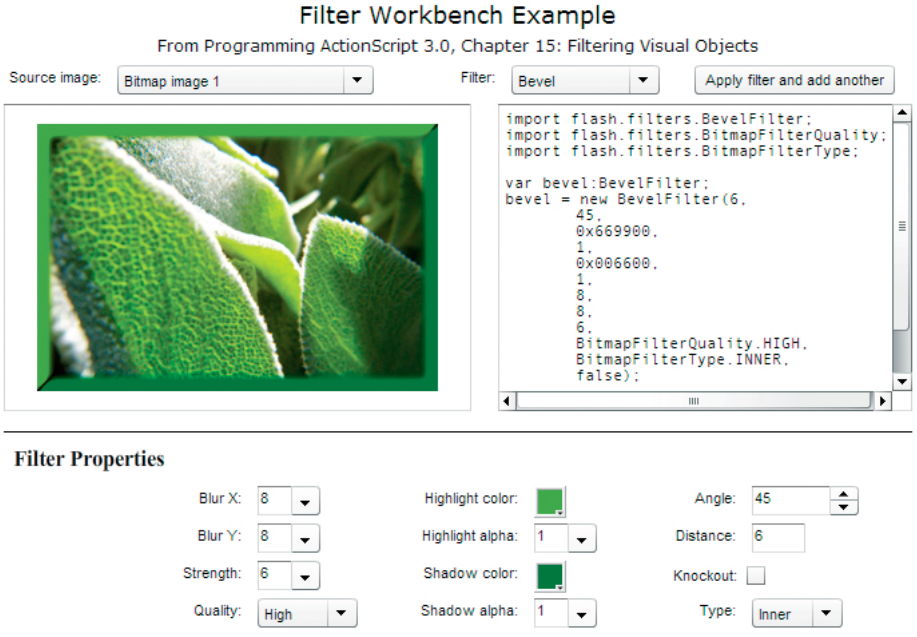
Pruebas con filtros de ActionScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación Filter Workbench se ha diseñado para ayudar a probar los diferentes efectos de filtro y a generar el código ActionScript necesario para lograr cada uno. La aplicación permite elegir entre tres archivos diferentes que incluyen contenido visual, tales como imágenes de mapa de bits y animaciones creadas por Flash, así como aplicar ocho filtros de ActionScript diferentes a la imagen seleccionada, ya sea de forma individual o en combinación con otros filtros. La aplicación incluye los filtros siguientes:

- Bisel (flash.filters.BevelFilter)
- Desenfoque (flash.filters.BlurFilter)
- Matriz de colores (flash.filters.ColorMatrixFilter)
- Convolución (flash.filters.ConvolutionFilter)
- Sombra (flash.filters.DropShadowFilter)
- Iluminado (flash.filters.GlowFilter)
- Bisel degradado (flash.filters.GradientBevelFilter)
- Iluminado degradado (flash.filters.GradientGlowFilter)

Tras seleccionar una imagen y el filtro que se le va a aplicar, la aplicación muestra un panel con una serie de controles para el ajuste de las propiedades específicas del filtro elegido. Por ejemplo, en la siguiente imagen aparece la aplicación con el filtro de bisel seleccionado:



La vista previa se actualiza en tiempo real conforme se ajustan las propiedades del filtro. También es posible aplicar varios filtros. Para ello, personalice un filtro, haga clic en el botón Aplicar, personalice otro filtro y vuelva a hacer clic en dicho botón, y así sucesivamente.

A continuación se describen algunas funciones y limitaciones de los paneles de filtro de la aplicación:

- El filtro de matriz de colores incluye un grupo de controles que permiten manipular de forma directa las propiedades de imagen comunes, tales como el brillo, el contraste, la saturación y el matiz. Asimismo, es posible especificar valores personalizados de la matriz de colores.
- El filtro de convolución, disponible solo con ActionScript, incluye un conjunto de valores de matriz de convolución utilizados comúnmente, al tiempo que es posible especificar valores personalizados. No obstante, aunque la clase ConvolutionFilter acepta matrices de cualquier tamaño, la aplicación Filter Workbench utiliza una matriz fija de 3 x 3, el tamaño de filtro que se utiliza con más frecuencia.
- Los filtros de mapa de desplazamiento y de sombreado no se encuentran disponibles en la aplicación Filter Workbench, solo en ActionScript.

Creación de instancias de filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación Filter Workbench incluye un conjunto de clases (una para cada filtro disponible) que son utilizadas por los distintos paneles para crear los filtros. Al seleccionar un filtro, el código de ActionScript asociado al panel correspondiente crea una instancia de la clase Factory de filtro apropiada. (Estas clases también se denominan *clases de fábrica* porque su finalidad es crear instancias de otros objetos, del mismo modo que en una fábrica del mundo real se crean productos individuales.)

Cada vez que se cambia el valor de una propiedad en el panel, el código de este llama al método apropiado de la clase de fábrica. Cada clase de fábrica incluye métodos específicos utilizados por el panel para crear la instancia de filtro apropiada. Por ejemplo, al seleccionar el filtro de desenfoque, la aplicación crea una instancia de BlurFactory. La clase BlurFactory incluye un método `modifyFilter()` que acepta tres parámetros: `blurX`, `blurY`, y `quality`, los cuales se utilizan conjuntamente para crear la instancia de BlurFilter deseada:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Por otro lado, al seleccionar el filtro de convolución se obtiene una mayor flexibilidad y, en consecuencia, es posible controlar un mayor número de propiedades. En la clase ConvolutionFactory se llama al siguiente código cuando se selecciona un valor diferente en el panel del filtro:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                            matrixY:Number = 0,
                                            matrix:Array = null,
                                            divisor:Number = 1.0,
                                            bias:Number = 0.0,
                                            preserveAlpha:Boolean = true,
                                            clamp:Boolean = true,
                                            color:uint = 0,
                                            alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha,
clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Observe en cada caso que, cuando se modifican los valores del filtro, el objeto Factory distribuye un evento `Event.CHANGE` para comunicar a los detectores que los valores del filtro han cambiado. La clase FilterWorkbenchController, responsable de aplicar los filtros al contenido filtrado, detecta dicho evento a fin de determinar cuándo es necesario recuperar una nueva copia del filtro y volver a aplicarla al contenido filtrado.

La clase FilterWorkbenchController no necesita conocer los detalles específicos de la clase de fábrica de cada filtro, solo saber que el filtro ha cambiado y poder acceder a una copia del mismo. Para que esto sea posible, la aplicación incluye una interfaz, IFilterFactory, que define el comportamiento que debe tener una clase de fábrica de filtro para que la instancia de FilterWorkbenchController de la aplicación pueda cumplir su cometido. IFilterFactory define el método `getFilter`() utilizado en la clase FilterWorkbenchController:

```
function getFilter():BitmapFilter;
```

Observe que la definición del método de interfaz `getFilter()` especifica que devuelve una instancia de BitmapFilter en lugar de un tipo de filtro específico. La clase BitmapFilter no define ningún tipo específico de filtro, sino que se trata de la clase de base a partir de la cual se crean todas las clases de filtro. Cada clase de fábrica de filtro define una implementación específica del método `getFilter()` en la cual se devuelve una referencia al objeto de filtro creado. Por ejemplo, aquí se muestra una versión abreviada del código fuente de la clase ConvolutionFactory:

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

En la implementación de la clase ConvolutionFactory del método `getFilter()` se devuelve una instancia de ConvolutionFilter, aunque esto no lo sabrán necesariamente los objetos que llamen a `getFilter()` — según la definición del método `getFilter()` que sigue ConvolutionFactory, debe devolver una instancia de BitmapFilter, que podría ser una instancia de cualquiera de las clases de filtro de ActionScript.

Aplicación de filtros a objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Tal y como se ha explicado anteriormente, la aplicación Filter Workbench utiliza una instancia de la clase FilterWorkbenchController (denominada en lo sucesivo "instancia de controlador"), la cual realiza la tarea de aplicar los filtros al objeto visual seleccionado. Antes de que la instancia de controlador pueda aplicar un filtro, debe identificar la imagen o el contenido visual al que se aplicará. Cuando se selecciona una imagen, la aplicación llama al método `setFilterTarget()` de la clase FilterWorkbenchController, pasando una de las constantes definidas en la clase ImageType:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

Con esta información, la instancia de controlador carga el archivo designado y, a continuación, lo almacena en una variable de instancia denominada `_currentTarget`:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Cuando se selecciona un filtro, la aplicación llama al método `setFilter()` de la instancia de controlador, ofreciendo al controlador una referencia al objeto de fábrica de filtro pertinente, el cual se almacena una variable de instancia denominada `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Observe que, tal y como se ha descrito anteriormente, la instancia de controlador no conoce el tipo específico de datos de la instancia de fábrica de filtro que se le facilita; solo sabe que el objeto implementa la instancia de IFilterFactory, lo que significa que incluye un método `getFilter()` y distribuye un evento `change` (`Event.CHANGE`) cuando se modifica el filtro.

Cuando se cambian las propiedades de un filtro en su panel correspondiente, la instancia de controlador detecta que el filtro se ha modificado a través del evento `change` de la fábrica del filtro, evento que llama al método `filterChange()` de la instancia de controlador. Este método, a su vez, llama al método `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

La tarea de aplicar el filtro al objeto de visualización se produce en el método `applyTemporaryFilter()`. En primer lugar, el controlador recupera una referencia al objeto de filtro llamando al método `getFilter()` de la fábrica de filtro.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

La instancia de controlador incluye una variable de instancia de Array, denominada `_currentFilters`, que es un conjunto en el cual almacena todos los filtros aplicados al objeto de visualización. El siguiente paso consiste en añadir el filtro recién actualizado a dicho conjunto:

```
_currentFilters.push(currentFilter);
```

A continuación, el código asigna el conjunto de filtros a la propiedad `filters` del objeto de visualización, la cual aplica en realidad los filtros a la imagen:

```
_currentTarget.filters = _currentFilters;
```

Por último, dado que este filtro recién añadido sigue siendo el filtro "de trabajo", conviene no aplicarlo de forma permanente al objeto de visualización, por lo que se elimina del conjunto `_currentFilters`:

```
_currentFilters.pop();
```

La eliminación del filtro no afecta al objeto de visualización filtrado, ya que este realiza una copia del conjunto de filtros cuando se asigna a la propiedad `filters` y utiliza este conjunto interno en lugar del original. Por este motivo, los cambios realizados en el conjunto de filtros no afectan al objeto de visualización hasta que dicho conjunto se vuelve a asignar a su propiedad `filters`.

Capítulo 15: Trabajo con sombreados de Pixel Bender

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El kit de herramientas de Adobe Pixel Bender permite a los desarrolladores escribir sombreados que generan efectos gráficos y llevar a cabo procesamiento adicional de datos e imágenes. El código de bytes de Pixel Bender se puede ejecutar en ActionScript para aplicar el efecto a datos de imagen o contenido visual. El uso de Pixel Bender en ActionScript permite crear efectos visuales personalizados y realizar procesamiento de datos más allá de las capacidades que incorpora ActionScript.

Nota: *la compatibilidad con Pixel Bender está disponible en Flash Player 10 y Adobe AIR 1.5. Los rellenos, filtros y mezclas de Pixel Bender no se admiten en el procesamiento con GPU.*

Más temas de ayuda

Centro de tecnología de Adobe Pixel Bender (en inglés)

Guía del desarrollador de Pixel Bender (en inglés)

Referencia de Pixel Bender (en inglés)

flash.display.Shader

flash.filters.ShaderFilter

Conceptos básicos de Pixel Bender para Flash (en inglés)

Conceptos básicos de Pixel Bender para Flex (en inglés)

Fundamentos de los sombreados de Pixel Bender

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Adobe Pixel Bender es un lenguaje de programación que se utiliza para crear o manipular contenido de imagen. Con Pixel Bender puede crear un núcleo, también denominado sombreado en este documento. El sombreado define una sola función que se ejecuta en cada uno de los píxeles de una imagen de forma individual. El resultado de cada llamada a la función es el color de salida en las coordenadas del píxel de la imagen. Las imágenes de entrada y los valores de parámetro se pueden especificar para personalizar la operación. En una sola ejecución de un sombreado, los valores de parámetro y entrada son constantes. Lo único que varía son las coordenadas del píxel cuyo color es el resultado de la llamada de la función.

Si es posible, la función de sombreado se llama para varias coordenadas de píxel de salida en paralelo. Esto mejora el rendimiento del sombreado y puede proporcionar un procesamiento de alto rendimiento.

En ActionScript, se pueden crear fácilmente tres tipos de efectos utilizando un sombreado:

- Relleno de dibujo
- Modo de mezcla

- Filtro

Un sombreado también se puede ejecutar en modo autónomo. Con el uso del modo autónomo se accede directamente al resultado de un sombreado en lugar de especificar previamente su uso previsto. Se puede acceder al resultado como datos de imagen o datos binarios o numéricos. No es necesario que los datos sean de imagen. De este modo puede proporcionar a un sombreado un conjunto de datos como entrada. El sombreado procesa los datos y puede acceder a los datos de resultado que devuelve.

la compatibilidad con Pixel Bender existe a partir de Flash Player 10 y Adobe AIR 1.5. Los rellenos, filtros y mezclas de Pixel Bender no se admiten en el procesamiento con GPU. En los dispositivos móviles, los sombreados de Pixel Bender no se ejecutan con el procesamiento con CPU. Sin embargo, el rendimiento no está al mismo nivel que en un equipo de escritorio. Es posible que muchos programas de sombreado solo se ejecuten en unos pocos fotogramas por segundo.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al crear y utilizar sombreados de Pixel Bender:

Núcleo Para Pixel Bender, un núcleo es lo mismo que un sombreado. Con el uso de Pixel Bender, el código define un núcleo, que define una sola función que se ejecuta en cada uno de los píxeles de una imagen individualmente.

Código de bytes de Pixel Bender Cuando se compila un núcleo de Pixel Bender, se transforma en un código de bytes de Pixel Bender. Al código de bytes se accede y se ejecuta en tiempo de ejecución.

Lenguaje Pixel Bender Lenguaje de programación utilizado para crear un núcleo de Pixel Bender.

Kit de herramientas de Pixel Bender Aplicación que se emplea para crear un archivo de código de bytes de Pixel Bender desde el código fuente del lenguaje. El kit de herramientas permite escribir, probar y compilar código fuente de Pixel Bender.

Sombreado En este documento, un sombreado es un conjunto de funcionalidad escrito en el lenguaje Pixel Bender. Un código del sombreado crea un efecto visual o realiza un cálculo. En cualquier caso, el sombreado devuelve un conjunto de datos (generalmente los píxeles de una imagen). El sombreado realiza la misma operación en cada punto de datos, con la única diferencia de las coordenadas del pí-xel de salida. El sombreado no se escribe en ActionScript. Se escribe en el lenguaje Pixel Bender y se compila en el código de bytes de Pixel Bender. Se puede incorporar a un archivo SWF en tiempo de compilación o cargarse como archivo externo en tiempo de ejecución. En cualquier caso se puede acceder al mismo en ActionScript creando un objeto Shader y vinculando este objeto al código de bytes del sombreado.

Entrada de sombreado Entrada compleja, generalmente datos de imagen de mapa de bits, que se proporciona a un sombreado para su uso en sus cálculos. Para cada variable de entrada definida en un sombreado, se utiliza un solo valor (es decir, una sola imagen o conjunto de datos binarios) para la ejecución competa del sombreado.

Parámetro de sombreado Un solo valor (o conjunto limitado de valores) que se proporciona a un sombreado para su uso en sus cálculos. Cada valor de parámetro se define para una sola ejecución del sombreado y el mismo valor se emplea en toda la ejecución.

Ejecución de los ejemplos de código

Puede que desee probar las listas de código de ejemplo que se proporcionan. Para probar el código hay que ejecutarlo y ver los resultados en el archivo SWF creado. Todos los ejemplos crean contenido utilizando la API de dibujo, que utiliza o se modifica mediante el efecto de sombreado.

La mayor parte de los listados del código de ejemplo incluyen dos partes. Una parte es el código fuente de Pixel Bender para el sombreado utilizado en el ejemplo. En primer lugar debe utilizar el kit de herramientas de Pixel Bender para compilar el código fuente en un archivo de código de bytes de Pixel Bender. Siga estos pasos para crear el archivo de código de bytes de Pixel Bender:

1. Abra el kit de herramientas de Adobe Pixel Bender. Si es necesario, en el menú Build (Compilación), seleccione "Turn on Flash Player warnings and errors" (Activar errores y avisos de Flash Player).
2. Copie el listado de código de Pixel Bender y péguelo en el panel de editor de código de su kit de herramientas.
3. En el menú File (Archivo), seleccione "Export kernel filter for Flash Player" (Exportar filtro de núcleo para Flash Player).
4. Guarde el archivo de código de bytes de Pixel Bender en el mismo directorio que el documento Flash. El nombre del archivo debe coincidir con el nombre especificado en la descripción del ejemplo.

La parte de ActionScript de todos los ejemplos se escribe como un archivo de clase. Para probar el ejemplo en Flash Professional:

1. Cree un documento de Flash vacío y guárdelo en el equipo.
2. Cree un nuevo archivo de ActionScript y guárdelo en el mismo directorio que el documento de Flash. El nombre del archivo debe coincidir con el nombre de la clase del listado de código. Por ejemplo, si el listado de código define una clase denominada MyApplication, use el nombre MyApplication.as para guardar el archivo de ActionScript.
3. Copie el listado de código en el archivo de ActionScript y guarde el archivo.
4. En el documento de Flash, haga clic en una parte vacía del escenario o espacio de trabajo para activar el inspector de propiedades del documento.
5. En el inspector de propiedades, en el campo Clase de documento, escriba el nombre de la clase de ActionScript que copió del texto.
6. Ejecute el programa utilizando Control > Probar película.

   Verá el resultado del ejemplo en el panel de previsualización.

En el capítulo "Cómo utilizar ejemplos de ActionScript" en la página 1145 se explican de forma más detallada estas técnicas para la comprobación de listados de código.

Carga e incorporación de un sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El primer paso para utilizar un sombreado de Pixel Bender en ActionScript es obtener acceso al sombreado en el código ActionScript. Debido a que los sombreados se crean a través del kit de herramientas de Adobe Pixel Bender, y se escriben en el lenguaje de Pixel Bender, no se puede acceder directamente a ellos en ActionScript. En su lugar, se crea una instancia de la clase Shader que representa el sombreado de Pixel Bender en ActionScript. El objeto Shader permite encontrar información sobre el sombreado. Así, permite determinar si el sombreado espera parámetros o valores de imagen de entrada. El objeto Shader se pasa a otros objetos para utilizar el sombreado. Por ejemplo, para utilizar el sombreado como un filtro, el objeto Shader se asigna a la propiedad `shader` de un objeto ShaderFilter. Como alternativa, para utilizar el sombreado como un relleno de dibujo, el objeto Shader se pasa como un argumento al método `Graphics.beginShaderFill()`.

El código ActionScript puede acceder a un sombreado creado por el kit de herramientas de Adobe Pixel Bender (un archivo .pbj) de dos formas diferentes:

- Cargado en tiempo de ejecución: el archivo de sombreado se puede cargar como un activo externo a través de un objeto URLLoader. Esta técnica es similar a cargar un activo externo, como un archivo de texto. En el siguiente ejemplo se muestra la carga de un archivo de código de bytes de sombreado en tiempo de ejecución y su vinculación a una instancia de Shader:

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- Incorporado en el archivo SWF: el archivo de sombreado se puede incorporar en el archivo SWF en tiempo de compilación a través de la etiqueta de metadatos `[Embed]`. La etiqueta de metadatos `[Embed]` solo está disponible si utiliza el SDK de Flex para compilar el archivo SWF. El parámetro `[Embed]` de la etiqueta `source` señala al archivo de sombreado y su parámetro `mimeType` es `"application/octet-stream"`, tal y como se muestra en este ejemplo:

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shader:Shader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

En cualquier caso, se vincula el código de bytes de sombreado sin procesar (la propiedad `URLLoader.data` o una instancia de la clase de datos `[Embed]`) a la instancia de Shader. Como se muestra en el ejemplo anterior, el código de bytes se puede asignar a la instancia de Shader de dos modos diferentes. Puede transferir el código de bytes de sombreado como un argumento al constructor `Shader()`. También puede establecerlo como la propiedad `byteCode` de la instancia de Shader.

Una vez que se ha creado un sombreado de Pixel Bender y se ha vinculado a un objeto Shader, el sombreado se puede utilizar para crear efectos de formas diferentes. Se puede utilizar como filtro, modo de mezcla, relleno de mapa de bits o para la reproducción autónoma de mapas de bits u otros datos. La propiedad `data` del objeto Shader se puede utilizar para acceder a los metadatos del sombreado, especificar imágenes de entrada y establecer valores de parámetro.

Acceso a los metadatos de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Al crear un núcleo de sombreado de Pixel Bender, el autor puede especificar los metadatos del sombreado en el código fuente de Pixel Bender. Al utilizar un sombreado en ActionScript, puede examinar el sombreado y extraer sus metadatos.

Al crear una instancia de Shader y vincularla a un sombreado de Pixel Bender, se crea un objeto ShaderData que contiene datos sobre el sombreado y se almacena en la propiedad `data` del objeto Shader. La clase ShaderData no define propiedades propias. Sin embargo, en tiempo de ejecución una propiedad se añade dinámicamente al objeto ShaderData para cada valor de metadatos definido en el código fuente del sombreado. El nombre dado a cada propiedad es el mismo que el especificado en los metadatos. Por ejemplo, supongamos que el código fuente de un sombreado de Pixel Bender incluye la siguiente definición de metadatos:

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

El objeto ShaderData creado para dicho sombreado se genera con las propiedades y valores siguientes:

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"`
- `version` (String): `"1"`
- `description` (String): `"Crea una versión de la imagen especificada con el brillo especificado"`

Debido a que las propiedades de metadatos se añaden dinámicamente al objeto ShaderData, el bucle `for..in` se puede utilizar para examinar el objeto ShaderData. Esta técnica permite determinar si el sombreado contiene metadatos y cuáles son los valores de estos. Además de las propiedades de metadatos, los objetos ShaderData pueden contener propiedades que representan entradas y parámetros definidos en el sombreado. Al utilizar un bucle `for..in` para examinar un objeto ShaderData, es necesario comprobar los tipos de datos de cada propiedad para determinar si la propiedad es una entrada (una instancia de ShaderInput), un parámetro (una instancia de ShaderParameter) o un valor de metadatos (una instancia de String). El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para examinar las propiedades dinámicas de la propiedad `data` de un sombreado. Cada valor de metadatos se añade a una instancia de Vector denominada `metadata`. Es necesario tener en cuenta que en este ejemplo se asume que ya se ha creado una instancia de Shader denominada `myShader`:

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

Para obtener una versión de este ejemplo que también extrae entradas y parámetros de sombreado, consulte "Identificación de entradas y parámetros de sombreado" en la página 319. Para obtener más información sobre las propiedades de entrada y parámetro, consulte "Especificación de valores de entrada y parámetro de sombreado" en la página 319.

Especificación de valores de entrada y parámetro de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Un gran número de sombreados de Pixel Bender se definen para utilizar una o varias imágenes de entrada que se utilizan en el procesamiento de sombreado. Por ejemplo, es habitual que un sombreado acepte una imagen fuente y dar lugar a dicha imagen con un efecto determinado aplicado en la misma. En función de cómo se utilice el sombreado, el valor de entrada se podrá especificar automáticamente o quizá será necesario proporcionar un valor de forma explícita. Asimismo, numerosos sombreados especifican parámetros que se utilizan para personalizar la salida del sombreado. Para poder utilizar el sombreado, también debe establecer explícitamente un valor para cada parámetro.

La propiedad `data` del objeto Shader se utiliza para establecer entradas y parámetros de sombreado y para determinar si un sombreado específico espera entradas o parámetros. La propiedad `data` es una instancia de ShaderData.

Identificación de entradas y parámetros de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El primer paso para especificar los valores de entrada y parámetro de sombreado es determinar si el sombreado que se utiliza espera imágenes o parámetros de entrada. Cada instancia de Shader tiene una propiedad `data` que contiene un objeto ShaderData. Si el sombreado define entradas o parámetros, a estos se accede como propiedades de dicho objeto ShaderData. Los nombres de las propiedades coinciden con los nombres especificados para las entradas y los parámetros del código fuente del sombreado. Por ejemplo, si un sombreado define una entrada `src`, el objeto ShaderData tiene una propiedad `src` que representa dicha entrada. Cada propiedad que representa una entrada es una instancia de ShaderInput, y cada propiedad que representa un parámetro es una instancia de ShaderParameter.

Lo ideal es que el autor del sombreado proporcione la documentación para dicho sombreado, indicando los valores de imagen de entrada y los parámetros que espera el sombrado, lo que representan, los valores adecuados, etc.

Sin embargo, si el sombreado no está documentado (y no se dispone de su código fuente), se pueden examinar los datos del sombreado para identificar las entradas y los parámetros. Las propiedades que representan las entradas y los parámetros se añaden de forma dinámica al objeto ShaderData. En consecuencia, se puede utilizar un bucle `for..in` para examinar el objeto ShaderData y determinar si su sombreado asociado define entradas o parámetros. Como se describe en "Acceso a los metadatos de sombreado" en la página 318, a los valores de metadatos definidos para un sombrado también se accede como una propiedad dinámica añadida a la propiedad `Shader.data`. Al utilizar esta técnica para identificar las entradas y los parámetros de sombreado, compruebe los tipos de datos de las propiedades dinámicas. Si una propiedad es una instancia de ShaderInput dicha propiedad representa una entrada. Si es una instancia de ShaderParameter, representa un parámetro. De lo contrario, es un valor de metadatos. El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para examinar las propiedades dinámicas de la propiedad `data` de un sombreado. Cada entrada (objeto ShaderInput) se añade a la instancia de Vector `inputs`. Cada parámetro (objeto ShaderParameter) se añade a la instancia de Vector `parameters`. Por último, las propiedades de metadatos se añaden a la instancia de Vector `metadata`. Es necesario tener en cuenta que en este ejemplo se asume que ya se ha creado una instancia de Shader denominada `myShader`:

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

Especificación de valores de entrada de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Un gran número de sombreados esperan una o varias imágenes de entrada que se utilizan en el procesamiento de sombreados. Sin embargo, en un gran número de casos se especifica una entrada automáticamente al utilizar el objeto Shader. Por ejemplo, si un sombreado requiere una entrada, y dicho sombreado se utiliza como filtro. Cuando el filtro se aplica a un objeto de visualización o un objeto BitmapData, dicho objeto se establece automáticamente con la entrada. En este caso no se establece explícitamente un valor de entrada.

No obstante, en algunos casos, en especial si un sombreado define varias entradas, el valor para la entrada se establece explícitamente. Cada entrada que se define en un sombreado viene representada en ActionScript por un objeto ShaderInput. El objeto ShaderInput es una propiedad de la instancia de ShaderData de la propiedad `data` del objeto Shader, como se describe en "Identificación de entradas y parámetros de sombreado" en la página 319. Por ejemplo, si un sombreado define una entrada denominada `src` y dicho sombreado está vinculado a un objeto Shader denominado `myShader`. En este caso, se accede al objeto ShaderInput correspondiente a la entrada `src` a través del identificador siguiente:

```
myShader.data.src
```

Cada objeto ShaderInput tiene una propiedad `input` que se utiliza para establecer el valor de la entrada. La propiedad `input` se establece como una instancia de BitmapData para especificar datos de imagen. La propiedad `input` también se puede establecer como BitmapData o Vector.<Number> instancia para especificar datos binarios o numéricos. Para obtener información detallada y conocer las restricciones relativas a la utilización de una instancia de BitmapData o Vector.<Number> como entrada, consulte el listado `ShaderInput.input` en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Además de la propiedad `input`, el objeto ShaderInput presenta propiedades que se pueden utilizar para determinar el tipo de imagen que espera la entrada. Entre estas propiedades se incluyen `width`, `height` y `channels`. Cada objeto ShaderInput también tiene una propiedad `index` que resulta útil para determinar si se debe proporcionar un valor explícito para la entrada. Si un sombreado espera más entradas de las establecidas automáticamente, se deberán establecer valores para dichas entradas. Para obtener información detallada sobre las diferentes formas de utilizar un sombreado, y si los valores de entrada se establecen automáticamente, consulte "Uso de un sombreado" en la página 324.

Especificación de valores de parámetro de sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Algunos sombreados definen valores de parámetro que el sombreado utiliza al crear su resultado. Por ejemplo, un sombreado que modifica el brillo de una imagen podría especificar un parámetro de brillo que determine el grado en que la operación afecta el brillo. Un solo parámetro definido en un sombreado puede esperar uno o varios valores, en función de la definición de parámetro en el sombreado. Cada parámetro que se define en un sombreado está representado en ActionScript por un objeto ShaderParameter. El objeto ShaderParameter es una propiedad de la instancia de ShaderData de la propiedad data del objeto Shader, como se describe en "Identificación de entradas y parámetros de sombreado" en la página 319. Por ejemplo, si un sombreado define un parámetro denominado `brightness`, y dicho sombreado está representado por un objeto Shader denominado `myShader`. En este caso se accede al objeto ShaderParameter correspondiente al parámetro `brightness` a través del identificador siguiente:

```
myShader.data.brightness
```

Para establecer un valor (o varios valores) para el parámetro, cree un conjunto ActionScript que contenga el valor o los valores y asígnela a la propiedad `value` del objeto ShaderParameter. La propiedad `value` se define como una instancia de Array porque es posible que un solo parámetro de sombreado requiera varios valores. Incluso si el parámetro de sombreado espera únicamente un solo valor, este valor se debe incluir en un objeto Array para asignarlo a la propiedad `ShaderParameter.value`. El listado siguiente muestra cómo establecer un solo valor con la propiedad `value`:

```
myShader.data.brightness.value = [75];
```

Si el código fuente de Pixel Bender del sombreado define un valor predeterminado para el parámetro, se creará un conjunto que contendrá el valor o los valores predeterminados y se asignará a la propiedad `value` del ShaderParameter al crear el objeto Shader. Una vez que se ha asignado un conjunto a la propiedad `value` (incluso si se trata del conjunto predeterminado), se podrá modificar el valor de parámetro cambiando el valor del elemento de conjunto. No es necesario crear un conjunto nuevo y asignarlo a la propiedad `value`.

En el ejemplo siguiente se muestra cómo establecer un valor de parámetro de sombreado en ActionScript. En este ejemplo el sombrado define el parámetro `color`. El parámetro `color` se declara como una variable `float4` en el código fuente de Pixel Bender, por lo que se trata de un conjunto de cuatro números de coma flotante. En el ejemplo, el valor del parámetro `color` cambia continuamente, y cada vez que cambia el sombreado se utiliza para dibujar un rectángulo de color en la pantalla. El resultado es un cambio de color animado.

Nota: *el código de este ejemplo fue escrito por Ryan Taylor. Gracias Ryan por compartir este ejemplo. Para obtener información sobre el trabajo de Ryan, consulte www.boostworthy.com/.*

El código ActionScript se centra en tres métodos:

- `init()`: en el método `init()` el código carga el archivo de código de bytes de Pixel Bender que contiene el sombreado. Cuando se carga el archivo, se llama al método `onLoadComplete()`.

- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. También se crea una instancia de Sprite denominada `texture`. En el método `renderShader()`, el código dibuja el resultado del sombreado en `texture` una sola vez por fotograma.
- `onEnterFrame()`: el método `onEnterFrame()` se llama una vez en cada fotograma, lo que crea el efecto de animación. En este método, el código establece el valor del parámetro del sombreado en el color nuevo y, a continuación, llama al método `renderShader()` para dibujar el resultado del sombreado como un rectángulo.
- `renderShader()`: en el método `renderShader()`, el código llama al método `Graphics.beginShaderFill()` para especificar un relleno de sombreado. A continuación, dibuja un rectángulo cuyo relleno viene definido por la salida del sombreado (el color generado). Para obtener más información sobre la utilización de un sombreado de este modo, consulte "Uso de un sombreado como relleno de dibujo" en la página 325.

Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto solo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en Flash Professional:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
```

```
        {
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

A continuación se incluye el código fuente del núcleo del sombreado ColorFilter, que se utiliza para crear el archivo de código de bytes “ColorFilter.pbj” de Pixel Bender:

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

Si utiliza un sombreado cuyos parámetros no están documentados, podrá determinar cuántos elementos y de qué tipo se deben incluir en el conjunto si comprueba la propiedad `type` del objeto ShaderParameter. La propiedad `type` indica el tipo de datos del parámetro como se define en el propio sombreado. Para obtener una lista del número y tipo de elementos que espera cada tipo de parámetro, consulte el listado de la propiedad `ShaderParameter.value` en la referencia del lenguaje.

Cada objeto ShaderParameter también tiene una propiedad `index` que indica la ubicación adecuada del parámetro en el orden de los parámetros del sombreado. Además de estas propiedades, los objetos ShaderParameter pueden presentar propiedades adicionales que contienen valores de metadatos proporcionados por el autor del sombreado. Por ejemplo, el autor puede especificar los valores de metadatos mínimo, máximo y predeterminado para un parámetro determinado. Los valores de metadatos que el autor especifica se añaden al objeto ShaderParameter como propiedades dinámicas. Para examinar estas propiedades, utilice el bucle `for..in` para recorrer las propiedades dinámicas del objeto ShaderParameter e identificar sus metadatos. El ejemplo siguiente muestra cómo utilizar un bucle `for..in` para identificar los metadatos de un objeto ShaderParameter. Cada valor de metadatos se añade a una instancia de Vector denominada `metadata`. Tenga en cuenta que este ejemplo asume que las instancias de Shader `myShader` ya están creadas, y que tienen un parámetro `brightness`:

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

Uso de un sombreado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Una vez que un sombreado de Pixel Bender está disponible en ActionScript como un objeto Shader, se puede utilizar de varias formas:

- Relleno de dibujo de sombreado: el sombreado define la parte de relleno de una forma dibujada con la API de dibujo
- Modo de mezcla: el sombreado define la mezcla entre dos objetos de visualización solapadas
- Filtro: el sombreado define un filtro que modifica el aspecto del contenido visual
- Procesamiento autónomo de sombreados: el procesamiento de sombreados se ejecuta sin especificar el uso pretendido de la salida. El sombreado se puede ejecutar opcionalmente en el fondo, y estará disponible cuando el procesamiento se complete. Esta técnica se puede utilizar para generar datos de mapa de bits y procesar datos no visuales.

Uso de un sombreado como relleno de dibujo

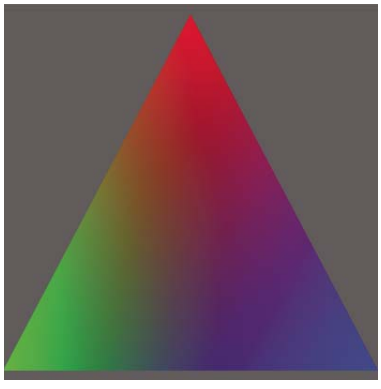
Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Cuando se utiliza un sombreado para crear un relleno de dibujo, se emplean los métodos de la API de dibujo para crear una forma vectorial. La salida del sombreado se usa para rellenar la forma, del mismo modo que cualquier imagen de mapa de bits se puede utilizar como relleno de mapa de bits con la API de dibujo. Para crear un relleno de sombreado, en el punto del código en el que desea comenzar a dibujar la forma, llame al método `beginShaderFill()` del objeto Graphics. Pase el objeto Shader como primer argumento al método `beginShaderFill()`, tal y como se muestra en este listado:

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

Cuando se utiliza un sombreado como relleno de dibujo, se establecen todos los valores de la imagen de entrada y los valores de parámetro que requiere el sombreado.

En el siguiente ejemplo se muestra el uso de un sombreado como relleno de dibujo. En este ejemplo, el sombreado crea un degradado de tres puntos. Este degradado tiene tres colores, cada uno de ellos en cada punto de un triángulo, con una mezcla de degradado entre los mismos. Asimismo, los colores giran para crear un efecto de color giratorio animado.



Nota: *Petri Leskinen ha escrito el código de este ejemplo. Gracias Petri por compartir este ejemplo. Para ver más ejemplos y tutoriales de Petri, consulte http://pixelero.wordpress.com/.*

El código de ActionScript se encuentra en tres métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método, el código establece los valores iniciales para los objetos Point que representan los puntos del triángulo. El código también crea una instancia de Sprite denominada `canvas`. Posteriormente, en `updateShaderFill()`, el código dibuja el resultado del sombreado en `canvas` una vez por fotograma. Finalmente, el código carga el archivo de código de bytes del sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. También establece los valores de parámetro iniciales. Finalmente, el código añade el método `updateShaderFill()` como detector para el evento `enterFrame`, lo que significa que se llama una vez por fotograma para crear un efecto de animación.

- `updateShaderFill()`: el método `updateShaderFill()` se llama una vez en cada fotograma, lo que crea el efecto de animación. En este método, el código calcula y define los valores de los parámetros de sombreado. A continuación el código llama al método `beginShaderFill()` para crear un relleno de sombreado y llama a otros métodos de la API de dibujo para dibujar el resultado de sombreado en un triángulo.

Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto solo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en Flash Professional:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);
```

```
            shader.data.point1.value = [topMiddle.x, topMiddle.y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener(Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120);

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

Este es el código fuente del núcleo del sombreado ThreePointGradient, que se utiliza para crear el archivo de código de bytes “ThreePointGradient.pbj” de Pixel Bender:

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
```

```
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;

    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

Nota: *si se utiliza un relleno de sombreado durante la representación con la unidad de procesamiento de gráficos (GPU), el área rellena se coloreará en cian.*

Para obtener más información sobre el dibujo de formas utilizando la API de dibujo, consulte "Uso de la API de dibujo" en la página 229.

Uso de un sombreado como modo de mezcla

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La utilización de un sombreado como modo de mezcla es similar al uso de otros modos de mezcla. El sombreado define la apariencia que se obtiene de dos objetos de visualización que se mezclan de forma conjunta visualmente. Para utilizar un sombreado como modo de mezcla, asigne el objeto Shader a la propiedad `blendShader` del objeto de visualización del primer plano. Si se asigna una valor distinto a `null` a la propiedad `blendShader`, la propiedad `blendMode` del objeto de visualización se establece automáticamente en `BlendMode.SHADER`. En el siguiente listado se muestra el uso de un sombreado como modo de mezcla. Observe que en este ejemplo se da por hecho que existe un objeto de visualización llamado `foreground`, que se incluye en el mismo elemento principal de la lista de visualización que el resto de contenido de visualización, con el valor `foreground` que se superpone al resto de contenido:

```
foreground.blendShader = myShader;
```

Cuando se utiliza un sombreado como modo de mezcla, el sombreado se debe definir con dos entradas como mínimo. Tal y como muestra el ejemplo, en el código no se definen los valores de entrada. Sin embargo, las dos imágenes mezcladas se utilizan automáticamente como entradas del sombreado. La imagen en primer plano se establece como segunda imagen. (Este es el objeto de visualización al que se aplica el modo de mezcla.) Una imagen de primer plano se crea adoptando un compuesto de todos los píxeles detrás del cuadro delimitador de la imagen de primer plano. Esta imagen se establece como la primera imagen de entrada. Si se utiliza un sombreado que espera más de dos entradas, se proporciona un valor para todas las entradas posteriores a las dos primeras.

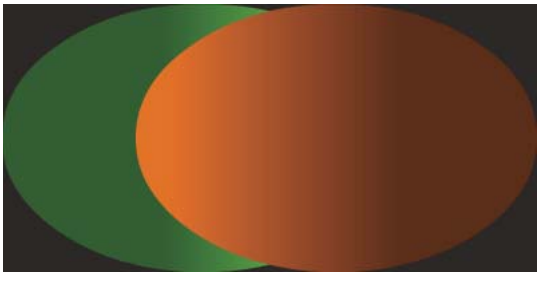
En el siguiente ejemplo se muestra el uso de un sombreado como modo de mezcla. El ejemplo utiliza un modo de mezcla de aclarado basado en la luminosidad. El resultado de la mezcla es que el valor de píxel más claro de cualquiera de los objetos mezclados pasa a ser el píxel que se muestra.

Nota: *Mario Klingemann ha escrito el código de este ejemplo. Gracias Mario por compartir este ejemplo. Para obtener más información sobre el trabajo de Mario y consultar su material, visite www.quasimondo.com/.*

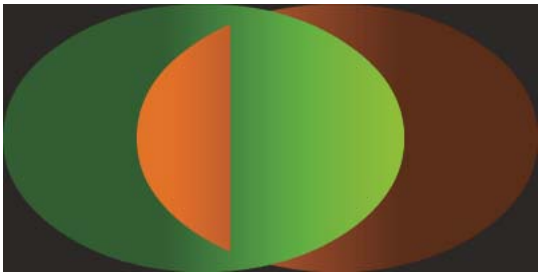
El código de ActionScript importante se encuentra en estos dos métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método el código carga el archivo de código de bytes de sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. A continuación dibuja tres objetos. El primero, `backdrop`, es un fondo gris oscuro detrás de los objetos mezclados. El segundo, `backgroundShape`, es una elipse con degradado verde. El tercer objeto, `foregroundShape`, es una elipse con degradado naranja.

  La elipse `foregroundShape` es el objeto de primer plano de la mezcla. La imagen de fondo de la mezcla está formada por la parte de `backdrop` y la parte de `backgroundShape`, que se superponen mediante al cuadro delimitador del objeto `foregroundShape`. El objeto `foregroundShape` es el objeto de primer orden en la lista de visualización. Se superpone parcialmente a `backgroundShape` y completamente a `backdrop`. Debido a este solapamiento, si no se aplica un modo de mezcla, la elipse naranja (`foregroundShape`) se muestra completamente y parte de la elipse verde (`backgroundShape`) queda oculta:



Sin embargo, cuando se aplica el modo de mezcla, la parte más clara de la elipse verde se "deja ver", ya que es más clara que la parte de `foregroundShape` que se solapa con ella:



Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto solo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en Flash Professional:

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
```

```
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

Este es el código fuente del núcleo del sombreado LumaLighten, que se utiliza para crear el archivo de código de bytes “LumaLighten.pbj” de Pixel Bender:

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

Para obtener más información sobre el uso de los modos de mezcla, consulte "Aplicación de modos de mezcla" en la página 191.

Nota: *cuando se ejecuta un programa de sombreado de Pixel Bender como mezcla en Flash Player o AIR, las funciones de muestra y* `outCoord()` *se comportan de forma distinta que en otros contextos. En una mezcla, una función de muestra siempre devuelve el píxel actual que está evaluando el sombreador. Por ejemplo, no se puede añadir un desplazamiento a* `outCoord()` *para obtener una muestra de un píxel vecino. Del mismo modo, si utiliza la función* `outCoord()` *fuera de una función de muestra, sus coordenadas siempre se calcularán como 0. Por ejemplo, no se puede utilizar la posición de un píxel para influir en la combinación de las imágenes mezcladas.*

Uso de un sombrado como filtro

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La utilización de un sombreado como filtro es similar al uso de cualquier otro filtro en ActionScript. Cuando se usa un sombreado como filtro, la imagen filtrada (un objeto de visualización u objeto BitmapData) se transmite al sombreado. El sombreado utiliza la imagen de entrada para crear la salida del filtro, que suele ser una versión modificada de la imagen original. Si el objeto filtrado es un objeto de visualización, la salida del sombreado se muestra en pantalla en lugar del objeto de visualización filtrado. Si el objeto filtrado es un objeto BitmapData, la salida del sombreado se convierte en el contenido del objeto BitmapData cuyo método `applyFilter()` se llama.

Para utilizar un sombreado como filtro, en primer lugar cree el objeto Shader tal y como se describe en "Carga e incorporación de un sombreado" en la página 316. A continuación, cree un objeto ShaderFilter vinculado al objeto Shader. El objeto ShaderFilter es el filtro que se aplica al objeto filtrado. Se aplica a un objeto del mismo modo que se aplica cualquier filtro. Se transmite a la propiedad `filters` de un objeto de visualización o se llama al método `applyFilter()` en un objeto BitmapData. Por ejemplo, el siguiente código crea un objeto ShaderFilter y aplica el filtro a un objeto de visualización denominado `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

Si se utiliza una sombreado como filtro, el sombreado se debe definir con una entrada como mínimo. Tal y como muestra el ejemplo, en el código no se establece el valor de entrada. Sin embargo, el objeto de visualización filtrado u objeto BitmapData se define como imagen de entrada. Si se emplea un sombreado que espera varias entradas, indique un valor para todas las entradas que no sean la primera.

En algunos casos, un filtro cambia las dimensiones de la imagen original. Por ejemplo, un efecto típico de sombra paralela añade píxeles adicionales que contienen la sombra que se agrega a la imagen. Si se utiliza un sombreado que cambia las dimensiones de la imagen, establezca las propiedades `leftExtension`, `rightExtension`, `topExtension` y `bottomExtension` para indicar el grado de cambio de tamaño deseado de la imagen.

El siguiente ejemplo muestra el uso de un sombreado como filtro. El filtro de este ejemplo invierte los valores del canal rojo, verde y azul de una imagen. El resultado en la versión "negativa" de la imagen.

Nota: *el sombreado que utiliza este ejemplo es el núcleo de Pixel Bender invertRGB.pbk que se incluye en el kit de herramientas del lenguaje. Puede cargar el código fuente para el núcleo desde el directorio de instalación del kit de herramientas de Pixel Bender. Compile el código fuente y guarde el archivo de código de bytes en el mismo directorio que el código fuente.*

El código de ActionScript importante se encuentra en estos dos métodos:

- `init()`: el método `init()` se llama cuando se carga la aplicación. En este método el código carga el archivo de código de bytes de sombreado.
- `onLoadComplete()`: en el método `onLoadComplete()` el código crea el objeto Shader denominado `shader`. Crea y dibuja el contenido de un objeto denominado `target`. El objeto `target` es un rectángulo relleno con un color de degradado lineal que es rojo en la parte izquierda, amarillo-verde en el centro y azul claro en la parte derecha. El objeto no filtrado presenta el siguiente aspecto:

Cuando se aplica el filtro los colores se invierten y el rectángulo presenta el siguiente aspecto:

El sombreado que utiliza este ejemplo es el núcleo de Pixel Bender de ejemplo “invertRGB.pbk”, incluido en el kit de herramientas del lenguaje. El código fuente está disponible en el archivo “invertRGB.pbk” en el directorio de instalación del kit de herramientas de Pixel Bender. Compile el código fuente y guarde el archivo de código de bytes con el nombre “invertRGB.pbj” en el mismo directorio que el código fuente de ActionScript.

Este es el código de ActionScript de este ejemplo. Utilice esta clase como clase principal de aplicación para un proyecto solo ActionScript en Flex, o bien, como la clase de documento para el archivo FLA en Flash Professional:

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
```

```
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

Para obtener más información sobre la aplicación de filtros, consulte “Creación y aplicación de filtros” en la página 280.

Uso de un sombreado en modo autónomo

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Cuando se utiliza un sombreado en modo autónomo, el procesamiento del sombreado se ejecuta independientemente de cómo se vaya a utilizar la salida. Se especifica un sombreado para ejecutar, se establecen los valores de parámetro y entrada y se designa un objeto en el que se sitúan los datos del resultado. Un sombreado se puede emplear en modo autónomo por dos motivos:

- Procesamiento de datos que no son de imagen: en el modo autónomo, puede optar por transmitir datos numéricos o binarios arbitrarios al sombreado en lugar de datos de imagen de mapa de bits. Puede optar por que el resultado del sombreado se devuelva como datos binarios o numéricos además de como datos de imagen de mapa de bits.
- Procesamiento en segundo plano: cuando un sombreado se ejecuta en modo autónomo, de forma predeterminada se hace asíncronicamente. Esto significa que el sombreado se ejecuta en segundo plano mientras la aplicación continúa en ejecución y el código se notifica cuando finaliza el procesamiento del sombreado. Puede utilizar un sombreado que tarde más tiempo en ejecutarse y no bloquee la interfaz de usuario de la aplicación u otro procesamiento mientras que el sombreado se esté ejecutando.

Puede utilizar un objeto ShaderJob para ejecuta un sombreado en modo autónomo. En primer lugar cree un objeto ShaderJob y vincúlelo al objeto Shader que representa el sombreado que se va a ejecutar:

```
var job:ShaderJob = new ShaderJob(myShader);
```

A continuación, se establecen todos los valores de parámetro o entrada que espera el sombreado. Si está ejecutando el sombreado en segundo plano, también se registra un detector para el evento `complete` del objeto ShaderJob. El detector se llama cuando el sombreado finaliza su tarea:

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

A continuación, se crea un objeto en el que se escribe el resultado de la operación de sombreado cuando finaliza la operación. Ese objeto se asigna a la propiedad `target` del objeto ShaderJob:

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Asigne una instancia de BitmapData a la propiedad `target` si está utilizando ShaderJob para realizar el procesamiento de imagen. Si va a procesar datos numéricos o binarios, asigne un objeto ByteArray o instancia de Vector.<Number> a la propiedad `target`. En este caso, debe establecer las propiedades `width` y `height` del objeto ShaderJob para que especifiquen la cantidad de datos de salida en el objeto `target`.

Nota: *se pueden establecer las propiedades `shader`, `target`,`width` y `height` del objeto ShaderJob en un paso transmitiendo argumentos al constructor `ShaderJob()`, del siguiente modo:`var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`*

Cuando esté listo para ejecutar el sombreado, puede llamar al método `start()` del objeto ShaderJob:

```
job.start();
```

De forma predeterminada, la llamada a `start()` hace que ShaderJob se ejecute asincrónicamente. En ese caso, la ejecución del programa continúa inmediatamente con la siguiente línea de código en lugar de esperar a que termine el sombreado. Una vez terminada la operación del sombreado, el objeto ShaderJob llama a sus detectores de eventos `complete`, notificándoles que se ha finalizado. En este punto (es decir, en el cuerpo del detector de eventos `complete`) el objeto `target` contiene el resultado de la operación de sombreado.

Nota: *en lugar de utilizar el objeto de propiedad `target`, puede recuperar el resultado del sombreado directamente del objeto de evento que se transmite al método detector. El objeto de evento es una instancia de ShaderEvent. El objeto ShaderEvent tiene tres propiedades que se pueden utilizar para acceder al resultado, dependiendo del tipo de datos del objeto que se establezca como propiedad `target`: `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` y `ShaderEvent.vector`.*

De forma alternativa, puede pasar un argumento `true` al método `start()`. En ese caso, la operación de sombreado se ejecuta sincrónicamente. Todo el código (incluyendo la interacción con la interfaz de usuario y cualquier otro evento) se detiene mientras se ejecuta el sombreado. Cuando el sombreado finaliza, el objeto `target` contiene el resultado del proceso y el programa continúa con la siguiente línea de código.

```
job.start(true);
```

Capítulo 16: Trabajo con clips de película

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase MovieClip es la principal para los símbolos de clip de película y animación que se crean en el entorno de desarrollo de Adobe® Flash®. Tiene todos los comportamientos y la funcionalidad de los objetos de visualización, pero con propiedades y métodos adicionales para controlar la línea de tiempo de un clip de película.

Fundamentos de la utilización de película

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los clips de película son un elemento clave para las personas que crean contenido animado con la herramienta de edición Flash y desean controlar el contenido con ActionScript. Siempre que se crea un símbolo de clip de película en Flash, Flash añade el símbolo a la biblioteca del documento Flash en el que se crea. De forma predeterminada, este símbolo se convierte en una instancia de la clase MovieClip y, como tal, dispone de las propiedades y métodos de esta clase.

Cuando se coloca una instancia de un símbolo de clip de película en el escenario, el clip de película progresa automáticamente por su línea de tiempo (si tiene más de un fotograma) a menos que se haya modificado la reproducción con ActionScript. Esta línea de tiempo es lo que distingue a la clase MovieClip, que permite crear animación mediante interpolaciones de movimiento y de forma, a través de la herramienta de edición de Flash. En cambio, con un objeto de visualización que es una instancia de la clase Sprite solo se puede crear una animación modificando mediante programación los valores del objeto.

En versiones anteriores de ActionScript, la clase MovieClip era la clase base de todas las instancias en el escenario. En ActionScript 3.0, un clip de película solo es uno de los numerosos objetos de visualización que aparecen en la pantalla. Si no se necesita una línea de tiempo para el funcionamiento de un objeto de visualización, utilizar la clase Shape o Sprite en lugar de la clase MovieClip puede mejorar el rendimiento de la representación. Para obtener más información sobre cómo elegir el objeto de visualización apropiado para una tarea, consulte "Selección de una subclase DisplayObject" en la página 177.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes relacionados con los clips de película:

AVM1 SWF archivo SWF creado con ActionScript 1.0 o ActionScript 2.0 para Flash Player 8 o una versión anterior.

AVM2 SWF archivo SWF creado con ActionScript 3.0 para Adobe Flash Player 9 (o versión posterior) o Adobe AIR.

Archivo SWF externo Archivo SWF que se crea de forma independiente del archivo SWF del proyecto y que debe cargarse en el archivo SWF del proyecto y reproducirse en dicho archivo SWF.

Fotograma División de tiempo más pequeña en la línea de tiempo. Al igual que en una tira de película animada, cada fotograma es como una instantánea de la animación en el tiempo; cuando se reproduce rápidamente la secuencia de los fotogramas, se crea el efecto de animación.

Línea de tiempo Representación metafórica de una serie de fotogramas que forman una secuencia de animación de un clip de película. La línea de tiempo de un objeto MovieClip es equivalente a la línea de tiempo de la herramienta de edición Flash.

Cabeza lectora Marcador que identifica la ubicación (fotograma) en la línea de tiempo que se está mostrando en un momento determinado.

Trabajo con objetos MovieClip

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se publica un archivo SWF, Flash convierte de forma predeterminada todas las instancias de símbolo de clip de película del escenario en objetos MovieClip. Para hacer que un símbolo de clip de película esté disponible para ActionScript, se le asigna un nombre de instancia en el campo Nombre de instancia del inspector de propiedades. Cuando se crea el archivo SWF, Flash genera el código que crea la instancia de MovieClip en el escenario y declara una variable con el nombre de instancia. Si se tienen clips de película con nombre anidados en otros clips de película con nombre, los clips de película secundarios se tratarán como propiedades del clip de película principal (se puede acceder al clip de película secundario con la sintaxis de punto). Por ejemplo, si un clip de película con el nombre de la instancia `childClip` está anidado dentro de otro clip con el nombre de la instancia `parentClip`, se puede hacer que se reproduzca la animación de la línea de tiempo del clip secundario llamando a este código:

```
parentClip.childClip.play();
```

Nota: *no se puede acceder a las instancias secundarias colocadas en el escenario en la herramienta de edición de Flash por código del constructor de una instancia principal, ya que no se han creado en ese punto en la ejecución del código. Antes de acceder al valor secundario, el principal debe crear la instancia secundaria por código o acceder con retraso a la función callback que detecta el valor secundario para que distribuya el evento* `Event.ADDED_TO_STAGE`.

Aunque se conservan algunos métodos y propiedades de la clase MovieClip de ActionScript 2.0, otros han cambiado. Todas las propiedades que empiezan por un carácter de subrayado han cambiado de nombre. Por ejemplo, las propiedades `_width` y `_height` son ahora `width` y `height`, mientras que las propiedades `_xscale` y `_yscale` son ahora `scaleX` y `scaleY`. Para obtener una lista completa de las propiedades y métodos de la clase MovieClip, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash .

Control de la reproducción de clips de película

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash utiliza la metáfora de una línea de tiempo para expresar animación o un cambio de estado. Cualquier elemento visual que emplee una línea de tiempo debe ser un objeto MovieClip o una ampliación de la clase MovieClip. Aunque se puede utilizar el código ActionScript para ordenar a cualquier clip de película que se detenga, se reproduzca o pase a otro punto de la línea de tiempo, no se puede utilizar para crear dinámicamente una línea de tiempo ni añadir contenido en fotogramas específicos; esto solo es posible en la herramienta de edición Flash.

Cuando se reproduce un clip de película, avanza por su línea de tiempo a la velocidad indicada en la velocidad de fotogramas del archivo SWF. Como alternativa, se puede sustituir esta configuración estableciendo la propiedad `Stage.frameRate` en ActionScript.

Reproducción de clips de película y detención de la reproducción

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `play()` y `stop()` permiten realizar el control básico de un clip de película a lo largo de su línea de tiempo. Por ejemplo, sea un símbolo de clip de película en el escenario que contiene una animación de una bicicleta moviéndose por la pantalla, con el nombre de instancia establecido en `bicycle`. Si se añade el código siguiente a un fotograma clave de la línea de tiempo principal,

```
bicycle.stop();
```

la bicicleta no se moverá (su animación no se reproducirá). El movimiento de la bicicleta puede empezar a través de alguna interacción del usuario. Por ejemplo, si hubiera un botón denominado `startButton`, el código siguiente en un fotograma clave de la línea de tiempo principal lo haría de forma que al hacer clic en el botón se reproduce la animación:

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

Avance rápido y rebobinado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `play()` y `stop()` no son la única manera de controlar la reproducción en un clip de película. También se puede avanzar o rebobinar manualmente la cabeza lectora a lo largo de la línea de tiempo con los métodos `nextFrame()` y `prevFrame()`. Cuando se llama a cualquiera de estos métodos, se detiene la reproducción, y la cabeza lectora avanza o rebobina un fotograma, respectivamente.

Utilizar el método `play()` equivale a llamar a `nextFrame()` cada vez que se activa el evento `enterFrame` del objeto de clip de película. De esta manera, se podría reproducir hacia atrás el clip de película `bicycle` añadiendo un detector de eventos para el evento `enterFrame` e indicando a `bicycle` que retroceda al fotograma anterior en la función de detector, como se muestra a continuación:

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

En la reproducción normal, si un clip de película contiene más de un fotograma, se reproducirá indefinidamente; es decir, que volverá al fotograma 1 si avanza más allá del último fotograma. Al utilizar `prevFrame()` o `nextFrame()`, este comportamiento no se produce automáticamente (llamar a `prevFrame()` cuando la cabeza lectora está en el Fotograma 1 no mueve la cabeza lectora al último fotograma). La condición `if` del ejemplo anterior comprueba si la cabeza lectora ha retrocedido al primer fotograma y la avanza hasta el último fotograma, creando un bucle continuo del clip de película que se reproduce hacia atrás.

Salto a otro fotograma y utilización de etiquetas de fotogramas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Enviar un clip de película a un nuevo fotograma es muy sencillo. Mediante una llamada a `gotoAndPlay()` o `gotoAndStop()`, el clip de película saltará al número de fotograma especificado como parámetro. Como alternativa, se puede pasar una cadena que coincida con el nombre de una etiqueta de fotograma. Se puede asignar una etiqueta a cualquier fotograma de la línea de tiempo. Para ello, hay que seleccionar un fotograma de la línea de tiempo e introducir un nombre en el campo Etiqueta de fotograma del inspector de propiedades.

Las ventajas de utilizar etiquetas de fotogramas en lugar de números se aprecian especialmente cuando se crea un clip de película complejo. Cuando el número de fotogramas, capas e interpolaciones de una animación es elevado, resulta útil etiquetar los fotogramas importantes con descripciones explicativas que representan los cambios en el comportamiento del clip de película; por ejemplo, "off" (fuera), "walking" (caminando) o "running" (corriendo). De esta forma se mejora la legibilidad del código y se logra una mayor flexibilidad, ya que las llamadas de código ActionScript para ir a un fotograma con etiqueta señalan a una sola referencia, la etiqueta, en lugar de a un número de fotograma específico. Si posteriormente se decide mover la cabeza lectora a un segmento específico de la animación en un fotograma diferente, será necesario modificar el código ActionScript manteniendo la misma etiqueta para los fotogramas en la nueva ubicación.

Para representar etiquetas de fotograma en el código, ActionScript 3.0 incluye la clase FrameLabel. Cada instancia de esta clase representa una sola etiqueta de fotograma y tiene una propiedad `name`, que representa el nombre de la etiqueta de fotograma especificado en el inspector de propiedades, y una propiedad `frame` que representa el número de fotograma del fotograma en el que está colocada la etiqueta en la línea de tiempo.

Para acceder a las instancias de FrameLabel asociadas con una instancia de clip de película, la clase MovieClip incluye dos propiedades que devuelven objetos FrameLabel directamente. La propiedad `currentLabels` devuelve un conjunto formado por todos los objetos FrameLabel de toda la línea de tiempo de un clip de película. La propiedad `currentLabel` devuelve una cadena que contiene el nombre de la etiqueta de fotograma encontrada más reciente en la línea de tiempo.

Supongamos que se crea un clip de película denominado `robot` y que se asignan etiquetas a los distintos estados de animación. Se podría configurar una condición que comprobara la etiqueta `currentLabel` para acceder al estado actual del robot, como en el código siguiente:

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Flash Player 11.3 y AIR 3.3 han añadido el evento `frameLabel` a la clase FrameLabel. Puede asignar un controlador de eventos a la instancia de FrameLabel que represente una etiqueta de fotograma. El evento se distribuye cuando la cabeza lectora accede al fotograma.

El siguiente ejemplo crea una instancia de FrameLabel para la etiqueta del segundo fotograma del conjunto de etiquetas de fotograma de MovieClip. Seguidamente, registra un controlador de eventos para el evento `frameLabel`:

```
var myFrameLabel:FrameLabel = robot.currentLabels[1];
myFrameLabel.addEventListener(Event.FRAME_LABEL, onFrameLabel);

function onFrameLabel(e:Event):void {
    //do something
}
```

Trabajo con escenas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el entorno de edición de Flash, se pueden utilizar las escenas para delimitar una serie de líneas de tiempo en las que avanzaría un archivo SWF. En el segundo parámetro de los métodos `gotoAndPlay()` o `gotoAndStop()`, se puede especificar una escena a la que enviar la cabeza lectora. Todos los archivos FLA comienzan con la escena inicial únicamente, pero se pueden crear nuevas escenas.

La utilización de escenas no es siempre el mejor enfoque, ya que presenta varios inconvenientes. Un documento de Flash que contenga varias escenas puede ser difícil de mantener, especialmente en entornos de varios autores. La utilización de varias escenas puede no ser eficaz en términos de ancho de banda, ya que el proceso de publicación combina todas las escenas en una sola línea de tiempo. Esto provoca una descarga progresiva de todas las escenas, incluso si no se reproducen nunca. Por estos motivos, a menudo se desaconseja utilizar varias escenas, a menos que se necesiten para organizar varias animaciones largas basadas en la línea de tiempo.

La propiedad `scenes` de la clase MovieClip devuelve un conjunto de objetos Scene que representan todas las escenas del archivo SWF. La propiedad `currentScene` devuelve un objeto Scene que representa la escena que se está reproduciendo actualmente.

La clase Scene tiene varias propiedades que ofrecen información sobre una escena. La propiedad `labels` devuelve un conjunto de objetos FrameLabel que representan las etiquetas de fotograma en dicha escena. La propiedad `name` devuelve el nombre de la escena como una cadena. La propiedad `numFrames` devuelve un entero que representa el número total de fotogramas en la escena.

Creación de objetos MovieClip con ActionScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una forma de añadir contenido a la pantalla en Flash consiste en arrastrar los activos de la biblioteca al escenario. Sin embargo, no es el único modo. En proyectos completos, los desarrolladores con experiencia suelen preferir crear clips de película mediante programación. Este enfoque implica varias ventajas: reutilización de código más sencilla, velocidad de tiempo de compilación más rápida y modificaciones más sofisticadas que solo están disponibles en ActionScript.

La API de lista de visualización de ActionScript 3.0 optimiza el proceso de creación dinámica de objetos MovieClip. La capacidad de crear una instancia de MovieClip directamente, por separado del proceso de añadirla a la lista de visualización, proporciona flexibilidad y simplicidad sin sacrificar control.

En ActionScript 3.0, al crear una instancia de clip de película (o cualquier otro objeto de visualización) mediante programación, no estará visible en la pantalla hasta que se añada a la lista de visualización llamando al método `addChild()` o `addChildAt()` en un contenedor de objeto de visualización. Esto permite crear un clip de película, establecer sus propiedades e incluso llamar a métodos antes de representarlo en la pantalla. Para obtener más información sobre la utilización de la lista de visualización, consulte "Trabajo con contenedores de objetos de visualización" en la página 163.

Exportación de símbolos de biblioteca para ActionScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De manera predeterminada, las instancias de símbolos de clip de película de una biblioteca de un documento Flash no se pueden crear dinámicamente (es decir, mediante ActionScript). El motivo de ello es que cada símbolo que se exporta para utilizarlo en ActionScript se suma al tamaño del archivo SWF y se detecta que no todos los símbolos se van a utilizar en el escenario. Por esta razón, para que un símbolo esté disponible en ActionScript, hay que especificar que se debe exportar el símbolo para ActionScript.

Para exportar un símbolo para ActionScript:

1. Seleccione el símbolo en el panel Biblioteca y abra su cuadro de diálogo Propiedades de símbolo.
2. Si es necesario, active la configuración avanzada.
3. En la sección Vinculación, active la casilla de verificación Exportar para ActionScript.

   De este modo, se activan los campos Clase y Clase base.

   De manera predeterminada, el campo Clase se llena con el nombre del símbolo, eliminando los espacios (por ejemplo, un símbolo denominado "Tree House" se convertiría en "TreeHouse"). Para especificar que el símbolo debe utilizar una clase personalizada para su comportamiento, hay que escribir el nombre completo de la clase, incluido el paquete, en este campo. Si se desea crear instancias del símbolo en ActionScript y no es necesario añadir ningún comportamiento adicional, se puede dejar el nombre de la clase tal cual.

   El valor predeterminado del campo Clase base es `flash.display.MovieClip`. Si se desea que el símbolo amplíe la funcionalidad de otra clase de cliente, se puede especificar el nombre de la clase, siempre y cuando dicha clase amplíe la clase Sprite (o MovieClip).
4. Presione el botón Aceptar para guardar los cambios.

   Si Flash no encuentra un archivo SWC vinculado o un archivo de ActionScript externo con una definición para la clase especificada (por ejemplo, si no se necesita un comportamiento adicional para el símbolo), se muestra una advertencia:

   No se pudo encontrar una definición de esta clase en la ruta de clases, por lo que se generará una automáticamente en el archivo SWF al exportar.

   Se puede pasar por alto esta advertencia si el símbolo de la biblioteca no requiere funcionalidad exclusiva más allá de la funcionalidad de la clase MovieClip.

Si no se proporciona una clase para el símbolo, Flash creará una clase para el símbolo equivalente a la siguiente:

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Si se desea añadir funcionalidad de ActionScript adicional al símbolo, hay que añadir las propiedades y métodos adecuados a la estructura de código. Por ejemplo, sea un símbolo de clip de película que contiene un círculo con una anchura de 50 píxeles y una altura de 50 píxeles, y se especifica que el símbolo debe exportarse para ActionScript con una clase denominada Circle. El siguiente código, incluido en un archivo Circle.as, amplía la clase MovieClip y proporciona al símbolo los métodos adicionales `getArea()` y `getCircumference()`:

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

El código siguiente, colocado en un fotograma clave en el Fotograma 1 del documento de Flash, creará una instancia del símbolo y la mostrará en pantalla:

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Este código muestra la creación de instancias basada en ActionScript como una alternativa a arrastrar activos individuales al escenario. Crea un círculo que tiene todas las propiedades de un clip de película, además de los métodos personalizados definidos en la clase Circle. Es un ejemplo muy básico: el símbolo de la biblioteca puede especificar varias propiedades y métodos en su clase.

La creación de instancias basada en ActionScript es eficaz, ya que permite crear de forma dinámica grandes cantidades de instancias, una tarea tediosa si tuviera que realizarse manualmente. También es flexible, pues permite personalizar las propiedades de cada instancia a medida que se crea. Para comprobar ambas ventajas, utilice un bucle para crear de forma dinámica varias instancias de Circle. Con el símbolo y la clase Circle descritos previamente en la biblioteca del documento Flash, coloque el código siguiente en un fotograma clave en el Fotograma 1:

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Esto ilustra la forma de crear y personalizar múltiples instancias de un símbolo rápidamente mediante código. La posición de cada instancia se modifica en función del recuento actual dentro del bucle y cada instancia recibe un color aleatorio mediante la propiedad `transform` (que hereda Circle a través de la ampliación de la clase MovieClip).

Carga de un archivo SWF externo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 3.0, los archivos SWF se cargan mediante la clase Loader. Para cargar un archivo SWF externo, el código ActionScript debe hacer cuatro cosas:

1. Crear un nuevo objeto URLRequest con el URL del archivo.
2. Crear un nuevo objeto Loader.
3. Llamar al método `load()` del objeto Loader pasando la instancia de URLRequest como parámetro.
4. Llamar al método `addChild()` en un contendor de objeto de visualización (como la línea de tiempo principal de un documento de Flash) para añadir la instancia de Loader a la lista de visualización.

Una vez finalizado, el código tiene el siguiente aspecto:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Este mismo código se puede utilizar para cargar un archivo de imagen externo como una imagen JPEG, GIF o PNG especificando el URL del archivo de imagen en lugar del URL del archivo SWF. Un archivo SWF, a diferencia de un archivo de imagen, puede contener código ActionScript. Por lo tanto, aunque el proceso de carga de un archivo SWF puede ser idéntico a la carga de una imagen, al cargar un archivo SWF externo, tanto el SWF que realiza la carga como el SWF cargado deben residir en el mismo entorno limitado de seguridad si Flash Player o AIR van a reproducir el archivo SWF y ActionScript se va a emplear para comunicarse de cualquier forma con el archivo SWF externo. Además, si el archivo SWF externo contiene clases que comparten el mismo espacio de nombres que las clases del archivo SWF que realiza la carga, es posible que sea necesario crear un nuevo dominio de aplicación para el archivo SWF cargado a fin de evitar conflictos de espacio de nombres. Para obtener más información sobre las consideraciones de seguridad y de dominio de aplicación, consulte "Trabajo con dominios de aplicación" en la página 150 y "Carga de contenido" en la página 1105.

Cuando el archivo SWF externo se carga correctamente, se puede acceder a él a través de la propiedad `Loader.content`. Si el archivo SWF externo se publica para ActionScript 3.0, será un objeto MovieClip o Sprite, en función de la clase que amplíe.

La carga de archivos SWF en Adobe AIR para iOS es algo diferente del procedimiento utilizado en otras plataformas. Para obtener más información, consulte "Carga de archivos SWF en AIR para iOS" en la página 208.

Consideraciones sobre la carga de un archivo SWF antiguo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si el archivo SWF externo se ha publicado con una versión anterior de ActionScript, hay que tener en cuenta algunas limitaciones importantes. A diferencia de un archivo SWF de ActionScript 3.0 que se ejecuta en AVM2 (máquina virtual ActionScript 2), un archivo SWF publicado para ActionScript 1.0 o 2.0 se ejecuta en AVM1 (máquina virtual ActionScript 1).

Existen diferencias importantes al cargar un archivo SWF 1.0 o de ActionScript 2.0 en un archivo SWF de ActionScript 3.0 (en comparación con cargar un archivo SWF de ActionScript 3.0). Como siempre, Flash Player proporciona compatibilidad completa con el contenido publicado previamente con versiones anteriores. Cualquier contenido que se ejecute en versiones anteriores de Flash Player se ejecuta en versiones de Flash Player que admiten ActionScript 3.0. Sin embargo, se aplican las siguientes limitaciones:

- El código ActionScript 3.0 puede cargar un archivo SWF escrito en ActionScript 1.0 o 2.0. Si un archivo SWF de ActionScript 1.0 o 2.0 se carga correctamente, el objeto cargado (propiedad `Loader.content`) es un objeto AVM1Movie. Una instancia de AVM1Movie no es lo mismo que una instancia de MovieClip. Es un objeto de visualización que, a diferencia de un clip de película, no incluye propiedades ni métodos relacionados con la línea de tiempo. El archivo SWF AVM2 principal no tendrá acceso a las propiedades, métodos ni objetos del objeto AVM1Movie cargado.
- Los archivos SWF escritos en ActionScript 1.0 o 2.0 no pueden cargar archivos SWF escritos en ActionScript 3.0. Esto significa que los archivos SWF creados en Flash 8 o Flex Builder 1.5 o versiones anteriores no pueden cargar archivos SWF de ActionScript 3.0.

La única excepción a esta regla es que un archivo SWF de ActionScript 2.0 puede sustituirse a sí mismo por un archivo SWF de ActionScript 3.0, siempre y cuando el archivo SWF de ActionScript 2.0 no haya cargado ningún elemento en ninguno de sus niveles. Para ello, el archivo SWF de ActionScript 2.0 debe realizar una llamada a `loadMovieNum()`, pasando un valor 0 al parámetro `level`.

- En general, los archivos SWF escritos en ActionScript 1.0 o 2.0 se deben migrar si van a funcionar de forma conjunta con los archivos SWF escritos en ActionScript 3.0. Por ejemplo, supongamos que se ha creado un reproductor de medios utilizando ActionScript 2.0. El reproductor carga distinto contenido que también se creó utilizando ActionScript 2.0. No es posible crear nuevo contenido en ActionScript 3.0 y cargarlo en el reproductor de medios. Es necesario migrar el reproductor de vídeo a ActionScript 3.0.

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

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

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

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

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

Ejemplo de clip de película: RuntimeAssetsExplorer

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La funcionalidad Exportar para ActionScript puede ser especialmente ventajosa para las bibliotecas que pueden utilizarse en más de un proyecto. Si Flash Player o AIR ejecutan un archivo SWF, los símbolos que se han exportado a ActionScript están disponibles en cualquier archivo SWF del mismo entorno limitado de seguridad que el archivo SWF que lo carga. De este modo, un solo documento de Flash puede generar un archivo SWF cuyo único propósito sea contener activos gráficos. Esta técnica es especialmente útil en proyectos grandes donde los diseñadores que trabajan en activos visuales pueden trabajar en paralelo con los desarrolladores que crean un archivo SWF "envolvente", que carga los archivos SWF de activos gráficos en tiempo de ejecución. Puede utilizarse este método para mantener una serie de archivos de versiones cuyos activos gráficos no dependan del progreso de desarrollo de programación.

La aplicación RuntimeAssetsExplorer carga cualquier archivo SWF que sea una subclase de RuntimeAsset y permite examinar los activos disponibles de dicho archivo SWF. El ejemplo muestra lo siguiente:

- Carga de un archivo SWF mediante `Loader.load()`
- Creación dinámica de un símbolo de la biblioteca exportado para ActionScript
- Control de la reproducción de MovieClip en el código ActionScript

Antes de comenzar, se debe que tener en cuenta que los archivos SWF que se van a ejecutar en Flash Player se deben ubicar en el mismo entorno limitado de seguridad. Para obtener más información, consulte "Entornos limitados de seguridad" en la página 1087.

Para obtener los archivos de aplicación para esta muestra, descargue las muestras de Flash Professional. Los archivos de la aplicación RuntimeAssetsExplorer se encuentran en la carpeta Samples/RuntimeAssetsExplorer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
RuntimeAssetsExample.mxml o RuntimeAssetsExample.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
RuntimeAssetsExample.as	Clase de documento para la aplicación de Flash (FLA).
GeometricAssets.as	Una clase de ejemplo que implementa la interfaz de RuntimeAsset.
GeometricAssets.fla	Un archivo FLA vinculado a la clase GeometricAssets (la clase de documento del archivo FLA), que contiene símbolos exportados para ActionScript.
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	Una interfaz que define los métodos necesarios que se esperan de todos los archivos SWF de activos de tiempo de ejecución que se cargarán en el contenedor del explorador.
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	La clase del símbolo de la biblioteca con la forma de un cuadro giratorio.
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	La clase del símbolo de la biblioteca con la forma de una estrella giratoria.

Establecimiento de una interfaz de biblioteca de tiempo de ejecución

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para que el explorador pueda interactuar correctamente con la biblioteca de un archivo SWF, debe formalizarse la estructura de las bibliotecas de activos de tiempo de ejecución. Con este fin se creará una interfaz, que se asemeja a una clase en que es un plano de los métodos que delimitan una estructura esperada, pero a diferencia de una clase no incluye el cuerpo de los métodos. La interfaz proporciona una forma de comunicación entre la biblioteca de tiempo de ejecución y el explorador. Cada archivo SWF de activos de tiempo de ejecución que se cargue en el navegador implementará esta interfaz. Para obtener más información sobre las interfaces y su utilidad, consulte *Aprendizaje de ActionScript 3.0*.

La interfaz de RuntimeLibrary será muy sencilla; solo se necesita una función que pueda proporcionar al explorador un conjunto de rutas de clases para los símbolos que se exportarán y estarán disponibles en la biblioteca de tiempo de ejecución. Para este fin, la interfaz tiene un solo método(): `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

Creación del archivo SWF de biblioteca de activos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Mediante la definición de la interfaz de RuntimeLibrary, es posible crear varios archivos SWF de biblioteca de activos que se pueden cargar en otro archivo SWF. La creación de un archivo SWF de biblioteca de activos implica cuatro tareas:

- Crear una clase para el archivo SWF de biblioteca de activos
- Crear clases para activos individuales contenidos en la biblioteca
- Crear los activos gráficos reales
- Asociar los elementos gráficos a las clases y publicar el archivo SWF de biblioteca

Crear una clase para implementar la interfaz RuntimeLibrary

A continuación, se creará la clase GeometricAssets que implementará la interfaz de RuntimeLibrary. Esta será la clase de documento del archivo FLA. El código para esta clase es muy similar a la interfaz RuntimeLibrary. Se diferencian en que en la definición de clase el método `getAssets()` no tiene cuerpo de método.

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

Si se tuviera que crear una segunda biblioteca de tiempo de ejecución, se crearía otro archivo FLA basado en otra clase (por ejemplo, AnimationAssets) que proporcionara su propia implementación de `getAssets()`.

Creación de clases para cada activo de MovieClip

En este ejemplo, simplemente se amplía la clase MovieClip sin añadir ninguna funcionalidad a los activos personalizados. El siguiente código de AnimatingStar equivale al de AnimatingBox:

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

Publicación de la biblioteca

Ahora se conectarán los activos basados en MovieClip con la nueva clase; para ello, se creará un nuevo archivo FLA y se introducirá GeometricAssets en el campo Clase de documento del inspector de propiedades. Para este ejemplo, se crearán dos formas muy básicas que utilizan una interpolación de la línea de tiempo para girar en el sentido de las agujas del reloj a través de 360 fotogramas. Los símbolos `animatingBox` y `animatingStar` están configurados en Exportar para ActionScript y tienen como valor del campo Clase las rutas de clases respectivas especificadas en la implementación de `getAssets()`. La clase base predeterminada de `flash.display.MovieClip` se conserva, ya que se va a crear una subclase de los métodos estándar de MovieClip.

Después de establecer la configuración de exportación del símbolo, hay que publicar el archivo FLA para obtener la primera biblioteca de tiempo de ejecución. Este archivo SWF podría cargarse en otro archivo SWF AVM2, y los símbolos AnimatingBox y AnimatingStar estarían disponibles para el nuevo archivo SWF.

Carga de la biblioteca en otro archivo SWF

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La última parte funcional que hay que resolver es la interfaz del usuario para el explorador de activos. En este ejemplo, la ruta a la biblioteca de tiempo de ejecución se especifica en el código como una variable denominada `ASSETS_PATH`. También se puede utilizar la clase FileReference, por ejemplo, para crear una interfaz que busque un determinado archivo SWF en el disco duro.

Cuando la biblioteca de tiempo de ejecución se carga correctamente, Flash Player llama al método `runtimeAssetsLoadComplete()`:

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

En este método, la variable rl representa el archivo SWF cargado. El código llama al método `getAssets()` del archivo SWF cargado, que obtiene la lista de activos disponibles, y los utiliza para llenar un componente ComboBox con una lista de activos disponibles llamando al método `populateDropDown()`. Este método almacena la ruta de clase completa de cada activo. Si se hace clic en el botón Añadir de la interfaz de usuario se activa el método `addAsset()`:

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

que obtiene la ruta de clase del activo que esté seleccionado actualmente en el ComboBox (`assetNameCbo.selectedItem.data`) y utiliza la función `getDefinitionByName()` (del paquete flash.utils) para obtener una referencia a la clase del activo para crear una nueva instancia de dicho activo.

Capítulo 17: Trabajo con interpolaciones de movimiento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

En la sección "Animación de objetos" en la página 198 se describe cómo implementar animaciones con scripts en ActionScript.

En esta sección describiremos una técnica distinta para crear animaciones en: las interpolaciones de movimiento. Esta técnica permite crear movimiento estableciéndolo interactivamente en un archivo FLA con Adobe® Flash® Professional. Seguidamente, se puede utilizar el movimiento en la animación basada en ActionScript en tiempo de ejecución.

Flash genera automáticamente el código ActionScript que implementa la interpolación de movimiento y permite que el usuario pueda copiarlo y reutilizarlo.

Para poder crear interpolaciones de movimiento es preciso disponer de una licencia de Flash Professional.

Más temas de ayuda

Paquete fl.motion

Fundamentos de interpolaciones de movimiento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

Las interpolaciones de movimiento son una forma sencilla de crear animaciones.

Una interpolación de movimiento modifica las propiedades de los objetos de visualización (como la posición o la rotación) fotograma por fotograma. Una interpolación de movimiento también puede cambiar el aspecto de un objeto de visualización mientras se mueve aplicando diversos filtros y otras propiedades. Es posible crear la interpolación de movimiento interactivamente con Flash Professional, que genera el código ActionScript para la interpolación de movimiento. Desde Flash, utilice el comando Copiar movimiento como ActionScript 3.0 para copiar el código ActionScript que creó la interpolación de movimiento. Posteriormente, puede reutilizar el código ActionScript para crear movimientos en su propia animación dinámica en tiempo de ejecución.

Consulte la sección Interpolaciones de movimiento en el manual *Uso de Flash Professional* para obtener más información sobre la creación de interpolaciones de movimiento.

Conceptos y términos importantes

A continuación se incluye un término importante que es relevante para esta función:

Interpolación de movimiento Construcción que genera fotogramas intermedios de un objeto de visualización en distintos estados y tiempos. Produce un efecto por el cual el primer estado evoluciona suavemente hacia el segundo. Se utiliza para mover un objeto de visualización en el escenario, así como para agrandarlo, encogerlo, girarlo, difuminarlo o cambiar su color con el paso del tiempo.

Copia de scripts de interpolación de movimiento en Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

Una interpolación genera fotogramas intermedios que muestran un objeto de visualización en distintos estados en dos fotogramas diferentes de una línea de tiempo. Produce un efecto por el cual la imagen del primer fotograma evoluciona suavemente hasta la imagen del segundo fotograma. En una interpolación de movimiento, el cambio de aspecto suele implicar también un cambio de posición del objeto de visualización, creándose así el movimiento. Además de cambiar la posición del objeto de visualización, una interpolación de movimiento también puede girar, sesgar, cambiar de tamaño o aplicar filtros al objeto.

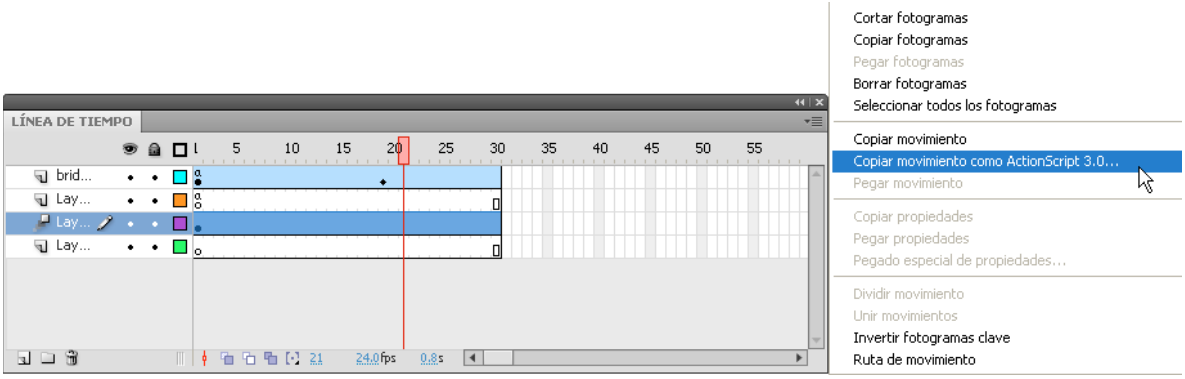
Para crear una interpolación de movimiento en Flash, mueva un objeto de visualización entre fotogramas clave de la línea de tiempo. Flash genera automáticamente el código ActionScript que describe la interpolación. Este código se puede copiar y guardar en un archivo. Consulte la sección Interpolaciones de movimiento en el manual *Uso de Flash Professional* para obtener más información sobre la creación de interpolaciones de movimiento.

Puede acceder al comando Copiar movimiento como ActionScript 3.0 en Flash de dos formas. La primera es desde el menú contextual de la interpolación en el escenario:

1 Seleccione la interpolación de movimiento en el escenario.

2 Haga clic con el botón derecho (Windows) o haga clic con la tecla Control pulsada (Macintosh).

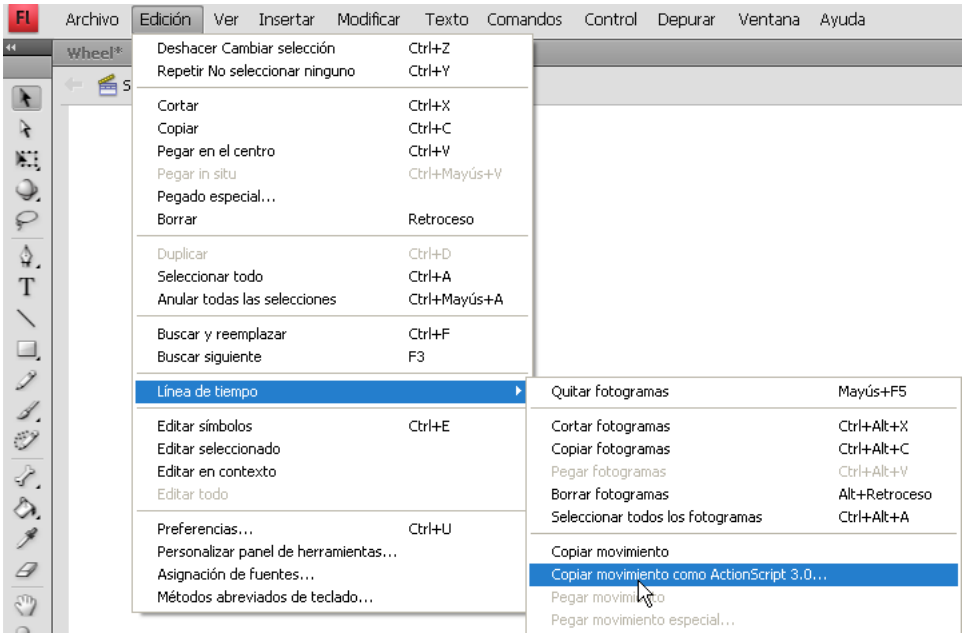
3 Seleccione Copiar movimiento como ActionScript 3.0...



La segunda posibilidad es elegir el comando directamente en el menú Edición de Flash

1 Seleccione la interpolación de movimiento en el escenario.

2 Seleccione Edición > Línea de tiempo > Copiar movimiento como ActionScript 3.0.



Una vez copiado el script, péguelo en un archivo y guárdelo.

Tras haber creado una interpolación de movimiento y haber copiado y guardado el script, puede reutilizarlo y modificarlo en su propia animación dinámica basada en ActionScript.

Incorporación de scripts de interpolación de movimiento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

El encabezado del código ActionScript copiado desde Flash contiene los módulos necesarios para que se admita la interpolación de movimiento.

Clases de interpolación de movimiento

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

Las clases fundamentales son AnimatorFactory, MotionBase y Motion, pertenecientes al paquete `fl.motion`. Puede necesitar clases adicionales, dependiendo de las propiedades que manipule la interpolación de movimiento. Por ejemplo, si la interpolación de movimiento transforma o gira el objeto de visualización, deberá importar las clases adecuadas de `flash.geom`. Si aplica filtros, deberá importar las clases de `flash.filter`. En ActionScript, una interpolación de movimiento es una instancia de la clase Motion. La clase Motion guarda una secuencia de animación de fotogramas clave que se puede aplicar a un objeto visual. La información de animación incluye datos sobre ubicación, escala, rotación, sesgado, color, filtros y aceleración.

El siguiente código ActionScript se ha copiado de una interpolación de movimiento creada para animar un objeto de visualización con nombre de instancia `Symbol1_2`. Declara una variable para un objeto MotionBase llamada `__motion_Symbol1_2`. La clase MotionBase es el elemento principal de la clase Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Seguidamente, el script crea el objeto Motion:

```
__motion_Symbol1_2 = new Motion();
```

Nombres de objetos Motion

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

En el caso anterior, Flash genera automáticamente el nombre `__motion_Symbol1_2` para el objeto Motion. Añade el prefijo `__motion_` al nombre del objeto de visualización. De este modo, el nombre generado automáticamente se basa en el nombre de instancia del objeto de destino de la interpolación de movimiento en la herramienta de edición de Flash. La propiedad `duration` del objeto Motion indica el número total de fotogramas en la interpolación de movimiento:

```
__motion_Symbol1_2.duration = 200;
```

De forma predeterminada, Flash asigna automáticamente un nombre a la instancia del objeto de visualización cuya interpolación de movimiento se está creando (si aún no tiene ningún nombre de instancia).

Cuando se vuelve a utilizar ActionScript creado por Flash en una animación propia, se puede conservar el nombre que genera Flash automáticamente para la interpolación, o bien, se puede sustituir por un nombre diferente. Si cambia el nombre de la interpolación, debe asegurarse de que lo hace también en todo el script.

Como alternativa, en Flash es posible asignar un nombre de su elección y asignarlo al objeto de destino de la interpolación de movimiento. Posteriormente, cree la interpolación de movimiento y copie el script. Puede utilizar la nomenclatura que desee, pero es importante que todos los objetos Motion del código ActionScript tengan un nombre exclusivo.

Descripción de la animación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

El método `addPropertyArray()` de la clase MotionBase añade un conjunto de valores para describir cada propiedad interpolada.

Potencialmente, el conjunto contiene un elemento de conjunto en cada fotograma clave de la interpolación de movimiento. A menudo, algunos conjuntos contienen menos elementos que el número total de fotogramas clave de la interpolación de movimiento. Esta situación se produce cuando el último valor del conjunto no cambia en los fotogramas restantes.

Si la longitud del argumento del conjunto es superior a la propiedad `duration` del objeto Motion, `addPropertyArray()` ajusta el valor de la propiedad `duration` en consecuencia. No añade fotogramas clave a las propiedades añadidas previamente. Los fotogramas clave recién añadidos se conservan en los fotogramas adicionales de la animación.

Las propiedades `x` e `y` del objeto Motion describen la posición variable del objeto interpolado a medida que se ejecuta la animación. Estas coordenadas son los valores que, con toda probabilidad, cambiarán en cada fotograma clave si cambia la posición del objeto de visualización. Puede añadir propiedades de movimiento adicionales con el método `addPropertyArray()`. Por ejemplo, puede añadir los valores `scaleX` y `scaleY` si el objeto interpolado cambia de tamaño. Añada los valores `scewX` y `skewY` si se sesga. Añada la propiedad `rotationConcat` si se gira.

Utilice el método `addPropertyArray()` para definir las siguientes propiedades de interpolación:

`x`	posición horizontal del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`y`	posición vertical del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`z`	posición de profundidad (eje z) del punto de transformación del objeto en el espacio de coordenadas de su elemento principal
`scaleX`	escala horizontal (en forma de porcentaje) del objeto aplicada desde el punto de transformación
`scaleY`	escala vertical (en forma de porcentaje) del objeto aplicada desde el punto de transformación
`skewX`	ángulo de sesgado horizontal (en grados) del objeto de destino aplicado desde el punto de transformación
`skewY`	ángulo de sesgado vertical (en grados) del objeto de destino aplicado desde el punto de transformación
`rotationX`	rotación del objeto alrededor del eje x desde su orientación original
`rotationY`	rotación del objeto alrededor del eje y desde su orientación original
`rotationConcat`	valores de rotación (eje z) del objeto en el movimiento relativo a la orientación anterior (según se aplica desde el punto de transformación)
`useRotationConcat`	si se establece, hace que el objeto de destino gire cuando `addPropertyArray()` proporciona datos de movimiento
`blendMode`	valor de la clase BlendMode que especifica la mezcla de colores del objeto con gráficos subyacentes
`matrix3D`	propiedad matrix3D (si existe en el fotograma clave). Se utiliza para interpolaciones 3D; si se usa, todas las propiedades de transformación anteriores se omiten
`rotationZ`	rotación del objeto alrededor del eje z (en grados) desde su orientación original relativa al contendor principal 3D. Se utiliza en interpolaciones 3D para sustituir a rotationConcat

Las propiedades que se añaden en el script generado automáticamente dependen de las propiedades asignadas a la interpolación de movimiento en Flash. Es posible añadir, eliminar o modificar algunas de estas propiedades al personalizar su propia versión del script.

El siguiente código asigna valores a las propiedades de una interpolación de movimiento llamada `__motion_Wheel`. En este caso, el objeto de visualización interpolado no cambia de posición, sino que gira sobre sí mismo en los 29 fotogramas de la interpolación de movimiento. Los diversos valores asignados al conjunto `rotationConcat` definen la rotación. El resto de valores de las propiedades de la interpolación de movimiento no varían.

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

En el siguiente ejemplo, el objeto de visualización llamado `Leaf_1` se mueve por el escenario. Sus conjuntos de propiedades `x` e `y` contienen distintos valores en cada uno de los 100 fotogramas de la animación. Además, el objeto gira alrededor de su eje `z` a medida que se mueve por el escenario. Los distintos elementos del conjunto de la propiedad `rotationZ` determinan la rotación.

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
```

```
__motion_Symbol1_4.addPropertyArray("rotationX", [64.0361]);
__motion_Symbol1_4.addPropertyArray("rotationY", [41.9578]);
__motion_Symbol1_4.addPropertyArray("rotationZ",
    [
        -18.0336,-17.5536,-17.0736,-16.5936,-16.1136,-15.6336,-15.1536,-14.6736,
        -14.1936,-13.7136,-13.2336,-12.7536,-12.2736,-11.7936,-11.3136,-10.8336,
        -10.3536,-9.8736,-9.3936,-8.9136,-8.4336,-7.9536,-7.4736,-6.9936,-6.5136,
        -6.0336,-7.21542,-8.39723,-9.57905,-10.7609,-11.9427,-13.1245,-14.3063,
        -15.4881,-16.67,-17.8518,-19.0336,-20.2154,-21.3972,-22.5791,-23.7609,
        -24.9427,-26.1245,-27.3063,-28.4881,-29.67,-30.8518,-32.0336,-31.0771,
        -30.1206,-29.164,-28.2075,-27.251,-26.2945,-25.338,-24.3814,-23.4249,
        -22.4684,-21.5119,-20.5553,-19.5988,-18.6423,-17.6858,-16.7293,-15.7727
        -14.8162,-13.8597,-12.9032,-11.9466,-10.9901,-10.0336,-10.9427,-11.8518,
        -12.7609,-13.67,-14.5791,-15.4881,-16.3972,-17.3063,-18.2154,-19.1245,
        -20.0336,-20.9427,-21.8518,-22.7609,-23.67,-24.5791,-25.4881,-26.3972,
        -27.3063,-28.2154,-29.1245,-30.0336,-28.3193,-26.605,-24.8907,-23.1765,
        -21.4622,-19.7479,-18.0336
    ]
);
__motion_Symbol1_4.addPropertyArray("blendMode", ["normal"]);
```

Añadir filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

Si el objeto de destino de una interpolación de movimiento contiene filtros, estos se añaden con los métodos `initFilters()` y `addFilterPropertyArray()` de la clase Motion.

Inicialización del conjunto de filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

El método `initFilters()` inicializa los filtros. Su primer argumento es un conjunto con los nombres de clases cualificados de todos los filtros aplicados al objeto de visualización. Este conjunto de nombres de filtros se genera a partir de la lista de filtros de la interpolación de movimiento en Flash. En su copia del script puede eliminar o añadir cualquiera de los filtros del paquete `flash.filters` a este conjunto. La siguiente llamada inicializa la lista de filtros del objeto de visualización de destino. Aplica los efectos `DropShadowFilter`, `GlowFilter` y `BevelFilter`. Además, copia la lista en todos los fotogramas clave del objeto Motion.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

Añadir filtros

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

El método `addFilterPropertyArray()` describe las propiedades de un filtro inicializado con los siguientes argumentos:

1. El primer argumento identifica un filtro por su índice. El índice hace referencia a la posición del nombre del filtro en el conjunto de nombres de la clase Filter transferidos en una llamada previa a `initFilters()`.

2 El segundo argumento es la propiedad del filtro que se guarda para dicho filtro en cada fotograma clave.

3 El tercer argumento es el valor de la propiedad del filtro especificada.

Con una llamada previa a `initFilters()`, las siguientes llamadas a `addFilterPropertyArray()` asignan un valor de 5 a las propiedades `blurX` y `blurY` de `DropShadowFilter`. El elemento `DropShadowFilter` es el primero (índice 0) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

Las siguientes tres llamadas asignan valores a las propiedades de calidad, alfa y color de `GlowFilter`, el segundo elemento (índice 1) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

Las siguientes cuatro llamadas asignan valores a las propiedades `shadowAlpha`, `shadowColor`, `highlightAlpha` y `highlightColor` de `BevelFilter`, el tercer elemento (índice 2) del conjunto de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

Ajuste del color con ColorMatrixFilter

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

Una vez inicializado el filtro `ColorMatrixFilter`, es posible definir las propiedades adecuadas de `AdjustColor` para ajustar el brillo, el contraste, la saturación y el tono del objeto de visualización interpolado. Generalmente, el filtro `AdjustColor` se aplica cuando la interpolación de movimiento se crea en Flash; puede precisarse en su copia de ActionScript. El siguiente ejemplo transforma el tono y la saturación del objeto de visualización a medida que se mueve.

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

Asociación de una interpolación de movimiento con sus objetos de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior, requiere Flash CS3 o posterior

La última tarea es asociar la interpolación de movimiento al objeto u objetos de visualización que manipula.

La clase AnimatorFactory gestiona la asociación entre una interpolación de movimiento y sus objetos de visualización de destino. El argumento del constructor AnimatorFactory es el objeto Motion:

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

Utilice el método `addTarget()` de la clase AnimatorFactory para asociar el objeto de destino con su interpolación de movimiento. El código ActionScript copiado desde Flash comenta la línea `addTarget()` y no especifica ningún nombre de instancia:

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

En su copia, especifique el objeto de visualización que se asociará con la interpolación de movimiento. En el siguiente ejemplo, los destinos se especifican como `greenWheel` y `redWheel`:

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

Puede asociar varios objetos de visualización con la misma interpolación de movimiento llamando varias veces a `addTarget()`.

Capítulo 18: Trabajo con cinemática inversa

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

La cinemática inversa (IK) es una gran técnica para crear movimientos realistas.

La IK permite crear movimientos coordinados dentro de una cadena de partes conectadas denominada esqueleto IK, de modo que las partes se mueven juntas como en la vida real. Las partes del esqueleto son sus huesos y sus uniones. Dado el punto final del esqueleto, IK calcula los ángulos necesarios de las uniones para alcanzar dicho punto final.

Calcular estos ángulos manualmente sería todo un reto de programación. Lo atractivo de esta función es la posibilidad de crear esqueletos interactivamente con Adobe® Flash® Professional. Y, posteriormente, animarlos con ActionScript. La herramienta IK incluida con Flash Professional realiza los cálculos para describir el movimiento del esqueleto. Es posible limitar el movimiento a determinados parámetros del código ActionScript.

Como novedad la versión de IK de Flash Professional CS5, se presenta el concepto de muelle de huelo, normalmente asociado a aplicaciones con animaciones de alto nivel. Esta función, empleada con el nuevo motor de física dinámica, permite configurar movimientos similares a los de la vida real. Este efecto es visible en tiempo de ejecución y durante la edición.

Para poder crear esqueletos de cinemática inversa, es necesario disponer de una licencia para Flash Professional.

Más temas de ayuda

Paquete fl.ik

Fundamentos de cinemática inversa

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

La cinemática inversa (IK) permite crear animaciones reales uniendo distintas piezas de modo que puedan moverse entre sí con realismo.

Por ejemplo, se puede utilizar la cinemática inversa para mover una pierna hasta una posición determinada articulando los movimientos necesarios de las uniones de la pierna para lograr la postura deseada. La cinemática inversa utiliza una arquitectura de huesos unidos entre sí en una estructura denominada esqueleto IK. El paquete `fl.ik` sirve para crear animaciones de movimientos naturales y realistas. Permite animar varios esqueletos IK a la perfección sin necesidad de tener conocimientos físicos de algoritmos de cinemática inversa.

Puede crear el esqueleto IK con sus huesos y uniones correspondientes en Flash Professional. Luego puede acceder a las clases de IK para animarlos en tiempo de ejecución.

Consulte la sección Uso de cinemática inversa en el manual *Uso de Flash Professional* para obtener instrucciones detalladas sobre la creación de un esqueleto IK.

Conceptos y términos importantes

La siguiente lista de referencias contiene términos importantes que resultan relevantes para esta función:

Esqueleto Cadena cinemática formada por huesos y uniones que se utiliza en animación por ordenador para simular movimientos con realismo.

Hueso Segmento rígido de un esqueleto (análogo al hueso de un esqueleto animal)

Cinemática inversa (IK) Proceso por el cual se determinan los parámetros de un objeto flexible unido denominado cadena cinemática o esqueleto.

Unión Lugar en el que dos huesos entran en contacto, creado para permitir el movimiento de los huesos; es similar a la articulación de un animal.

Motor de física Se utiliza un paquete de algoritmos relacionados con la física para proporcionar acciones reales de animación.

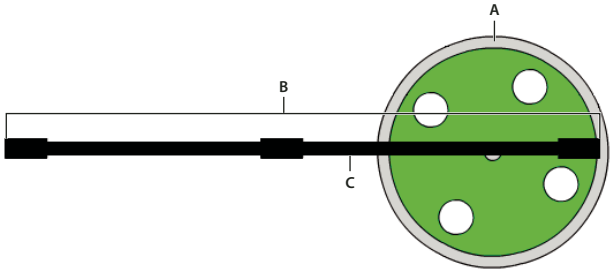
Muelle Cualidad de un hueso que se mueve y reacciona cuando el hueso principal se mueve y posteriormente disminuye de forma incremental con el tiempo.

Información general sobre animación de esqueletos IK

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

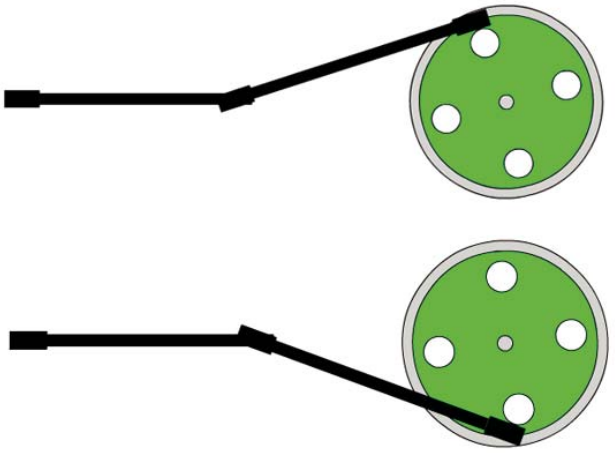
Una vez creado el esqueleto IK en Flash Professional, utilice las clases de `fl.ik` para limitar su movimiento, realizar un seguimiento de sus eventos o animarlo en tiempo de ejecución.

En la siguiente figura se muestra un clip de película denominado `Rueda`. El eje es una instancia de un objeto IKArmature denominado `Eje`. La clase IKMover mueve el esqueleto en sincronía con la rotación de la rueda. El hueso IKBone (`ikBone2`) del esqueleto se une a la rueda en su unión de cola.



A. *Rueda* ***B.*** *Eje* ***C.*** *ikBone2*

En tiempo de ejecución, la rueda gira asociada a la interpolación de movimiento `__motion_Wheel` descrita en "Descripción de la animación" en la página 354. Un objeto IKMover inicia y controla el movimiento del eje. En la siguiente figura se muestran dos instantáneas del esqueleto del eje asociado a la rueda giratoria en distintos fotogramas de la rotación.



En tiempo de ejecución, el siguiente código ActionScript:

- Obtiene información sobre el esqueleto y sus componentes
- Crea una instancia del objeto IKMover
- Mueve el eje junto con la rotación de la rueda

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Las clases IK utilizadas para mover el eje son:

- IKArmature: describe el esqueleto, una estructura en árbol formada por huesos y uniones. Se debe crear con Flash Professional.
- IKManager: clase contenedora de todos los esqueletos IK del documento. Se debe crear con Flash Professional.
- IKBone: segmento de un esqueleto IK
- IKJoint: conexión entre dos huesos IK
- IKMover: inicia y controla el movimiento IK de los esqueletos

Para obtener información detallada y completa sobre estas clases, consulte el paquete ik.

Obtener información sobre un esqueleto IK

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

En primer lugar, declare variables para el esqueleto, el hueso y la unión que formen parte del grupo de piezas que desea mover.

El siguiente código utiliza el método `getArmatureByName()` de la clase IKManager para asignar el valor del esqueleto Axle a la variable `tree` de IKArmature. El esqueleto Axle se ha creado previamente con Flash Professional.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

De forma similar, el siguiente código utiliza el método `getBoneByName()` de la clase IKArmature para asignar a la variable IKBone el valor del hueso `ikBone2`.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

La unión de cola del hueso `ikBone2` es la parte del esqueleto que se une a la rueda giratoria.

La siguiente línea declara la variable `endEffector` y le asigna la propiedad `tailjoint` del hueso `ikBone2`:

```
var endEffector:IKJoint = home.tailjoint;
```

La variable `pos` es un punto que almacena la posición actual de la unión `endEffector`.

```
var pos:Point = endEffector.position;
```

En este ejemplo, `pos` corresponde a la posición de la unión al final del eje, donde se conecta con la rueda. El valor original de esta variable se obtiene a partir de la propiedad `position` del objeto IKJoint.

Creación de una instancia de la clase IKMover y limitación del movimiento

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

Una instancia de la clase IKMover mueve el eje.

La siguiente línea crea una instancia del objeto `ik` de IKMover, transfiere a su constructor el elemento que desea mover y el punto de inicio del movimiento:

```
var ik:IKMover = new IKMover(endEffector, pos);
```

Las propiedades de la clase IKMover permiten limitar el movimiento de un esqueleto. Puede limitar el movimiento basado en la distancia, en las iteraciones o en el tiempo del movimiento.

Las siguientes parejas de propiedades refuerzan estos límites. La pareja consta de una valor booleano que indica si el movimiento está limitado y un entero que especifica dicho límite:

Propiedad booleana	Propiedad de entero	Límite establecido
`limitByDistance:Boolean`	`distanceLimit:int`	Establece la distancia máxima (en píxeles) que se mueve el motor IK en cada iteración.
`limitByIteration:Boolean`	`iterationLimit:int`	Especifica el número máximo de iteraciones que realiza el motor IK en cada movimiento.
`limitByTime:Boolean`	`timeLimit:int`	Establece el tiempo máximo (en milisegundos) asignado al motor IK para realizar el movimiento.

De forma predeterminada, todos los valores booleanos se establecen en `false`, por lo que el movimiento no se limita a no ser que se defina explícitamente un valor booleano como `true`. Para aplicar un límite, establezca la propiedad adecuada en `true` y después especifique un valor para la propiedad de entero correspondiente. Si establece el límite en un valor sin definir su propiedad Boolean correspondiente, el límite se obviará. En este caso, el motor IK sigue moviendo el objeto hasta llegar a otro límite o a la posición de destino del objeto IKMover.

En el siguiente ejemplo, la distancia máxima del movimiento del esqueleto se establece en 0,1 píxeles por iteración. El número máximo de iteraciones de cada movimiento se establece en diez.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

Desplazamiento de un esqueleto IK

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

El objeto IKMover mueve el eje dentro del detector de eventos de la rueda. En cada evento enterFrame de la rueda, se calcula una nueva posición de destino para el esqueleto. Si se utiliza su método `moveTo()`, el objeto IKMover mueve la unión de cola hasta su posición de destino o tan lejos como lo permitan las limitaciones definidas por las propiedades `limitByDistance`, `limitByIteration` y `limitByTime`.

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Uso de muelles

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS5 o posterior

La cinemática inversa de Flash Professional CS5 admite muelles de hueso. El muelle de hueso se puede establecer durante la edición y los atributos del muelle del hueso se pueden añadir o modificar en tiempo de ejecución. El muelle es una propiedad del hueso y sus uniones. Cuenta con dos atributos: `IKJoint.springStrength`, que establece la cantidad de muelle y `IKJoint.springDamping`, que añade resistencia al valor de intensidad y afecta al grado de descomposición del muelle.

La intensidad del muelle es un valor porcentual desde 0 (totalmente rígido, predeterminado) a 100 (muy suelto y controlado por física). Los huesos con muelle reaccionan al movimiento de su unión. Si no se activa ninguna otra traslación (rotación, x o y), los valores del muelle no se aplicarán.

La atenuación del muelle es un valor porcentual desde 0 (sin resistencia, predeterminado) a 100 (mucha atenuación). La atenuación cambia la cantidad de tiempo que transcurre entre el movimiento inicial del hueso y su vuelta a la posición de reposo.

Se puede verificar si los muelles están habilitados para un objeto IKArmature comprobando su propiedad `IKArmature.springsEnabled`. Las demás propiedades y métodos del muelle pertenecen a los objetos IKJoint independientes. Se puede habilitar una unión para la rotación angular y traslación a lo largo de los ejes x e y. Se puede situar un ángulo del muelle de la unión de rotación con `IKJoint.setSpringAngle` y una posición del muelle de la unión de traslación con `IKJoint.setSpringPt`.

Este ejemplo selecciona un hueso por nombre e identifica su valor tailJoint. El código prueba el esqueleto para ver si los muelles están habilitados y posteriormente establece las propiedades del muelle para la unión.

```
var arm:IKArmature = IKManager.getArmatureAt(0);
var bone:IKBone = arm.getBoneByName("c");
var joint:IKJoint = bone.tailJoint;
if (arm.springsEnabled) {
    joint.springStrength = 50; //medium spring strength
    joint.springDamping = 10; //light damping resistance
    if (joint.hasSpringAngle) {
        joint.setSpringAngle(30); //set angle for rotational spring
    }
}
```

Uso de eventos IK

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior, requiere Flash CS4 o posterior

La clase IKEvent permite crear un objeto de evento que contenga información sobre eventos IK. La información de IKEvent describe un movimiento que ha finalizado por haberse superado el tiempo especificado, la distancia o el límite de iteraciones.

El siguiente código muestra un detector de eventos y un controlador para realizar el seguimiento de los eventos de límite de tiempo. Este controlador de eventos informa sobre el tiempo, la distancia, el número de iteraciones y las propiedades de unión de un evento activado cuando se supera el límite de tiempo de IKMover.

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

Capítulo 19: Trabajo en tres dimensiones (3D)

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Los motores de ejecución de Flash Player y AIR admiten gráficos 3D de dos formas. Puede utilizar los objetos de visualización tridimensionales de la lista de visualización de Flash. Esto resulta adecuado para añadir efectos tridimensionales a contenido de Flash y para objetos con bajo recuento de polígonos. En Flash Player 11 y AIR 3, o versiones posteriores, puede procesar escenas complejas en 3D con la API Stage3D.

Una vista Stage3D no es un objeto de visualización. Los gráficos 3D se procesan en una vista que se muestra debajo de la lista de visualización de Flash (y encima de cualquier plano de vista StageVideo). En vez de utilizar las clases DisplayObject de Flash para crear una escena, puede utilizar un conducto 3D programable (similar a OpenGL y Direct3D). Este conducto toma datos de triángulo y texturas como entrada y procesa la escena con programas de sombreado facilitados por el usuario. Se recurre a aceleración por hardware si hay disponible en el equipo cliente una unidad con procesamiento de gráficos (GPU) compatible con controladores admitidos.

Stage3D proporciona una API de muy bajo nivel. En una aplicación, se recomienda utilizar una estructura 3D que admita Stage3D. Puede crear su propia estructura o utilizar una de las muchas estructuras de pago o de código abierto.

Para obtener más información sobre el desarrollo de aplicaciones 3D con Stage3D y sobre las estructuras 3D disponibles, visite el Centro de desarrollo de Flash Player: Stage 3D.

Fundamentos de objetos de visualización 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La principal diferencia entre un objeto bidimensional (2D) y otro tridimensional (3D) proyectado en una pantalla bidimensional es la adición de una tercera dimensión en el objeto. La tercera dimensión permite que el objeto se acerque y se aleje del punto de vista del usuario.

Si la propiedad `z` de un objeto de visualización se establece de forma explícita en un valor numérico, el objeto crea automáticamente una matriz de transformación 3D. Esta matriz se puede alterar para modificar la configuración de la transformación 3D del objeto.

Asimismo, la rotación 3D difiere de la rotación bidimensional. En 2D, el eje de rotación siempre es perpendicular al plano x/y; es decir, en el eje z. En 3D, el eje de rotación puede estar alrededor de cualquiera de los ejes x, y o z. La definición de las propiedades de escala y rotación de un objeto de visualización permite moverlo en un espacio 3D.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al programar gráficos tridimensionales:

Perspectiva En un plano 2D, representación de líneas paralelas convergentes en un punto de fuga para crear la ilusión de profundidad y distancia.

Proyección Producción de una imagen 2D de un objeto de mayores dimensiones; los mapas en proyección 3D señalan a un plano 2D.

Rotación Cambio de la orientación (y a menudo de la posición) de un objeto moviendo todos los puntos incluidos en el objeto en un movimiento circular.

Transformación Alteración de puntos 3D o de conjuntos de puntos mediante la traslación, escala, rotación y sesgo o una combinación de estas operaciones.

Traslación Cambio de posición de un objeto moviendo todos los puntos incluidos en el objeto mediante la misma cantidad en la misma dirección.

Punto de fuga Punto en el que las líneas paralelas que se alejan parecen coincidir cuando se representan en una perspectiva lineal.

Vector Un vector 3D representa un punto o una ubicación en el espacio tridimensional mediante el uso de las coordenadas cartesianas x, y y z.

Vértice Punto de esquina.

Malla con textura Cualquier punto que defina un objeto en el espacio 3D.

Mapeado UV Una forma de aplicar una textura o mapa de bits a una superficie 3D. El mapeado UV asigna valores a las coordenadas de una imagen como porcentajes del eje (U) horizontal y el eje (V) vertical.

Valor T Factor de escala para determinar el tamaño de un objeto 3D conforme el objeto se acerca o se aleja del punto de vista actual.

Técnica "culling" Representación, o no, de superficies, con una dirección de trazado específica. Con el uso de la técnica "culling" se pueden ocultar superficies que no se encuentran visibles en el punto de vista actual.

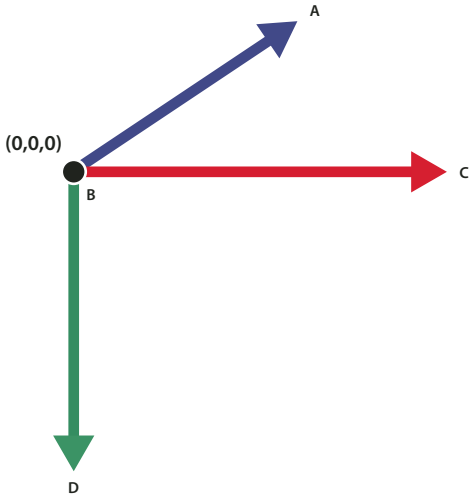
Información sobre los objetos de visualización 3D en el motor de ejecución de Flash Player y AIR

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

En versiones de Flash Player anteriores a Flash Player 10 y en versiones de Adobe AIR anteriores a Adobe AIR 1.5, los objetos de visualización tienen dos propiedades, `x` e `y`, que permiten colocarlos en un plano bidimensional. A partir de Flash Player 10 y Adobe AIR 1.5, todos los objetos de visualización de ActionScript tienen una propiedad `z` que permite colocarlos a lo largo del eje z (eje que suele usarse para indicar profundidad o distancia).

Flash Player 10 y Adobe AIR 1.5 incorporan compatibilidad para efectos 3D. No obstante, los objetos de visualización son esencialmente planos. Finalmente todos los objetos de visualización como, por ejemplo, MovieClip o Sprite, se representan a sí mismos en dos dimensiones, en un solo plano. Las funciones de 3D permiten situar, mover, rotar y transformar estos objetos planos en tres dimensiones. También permiten administrar puntos 3D y convertirlos en coordenadas x, y 2D para que pueda proyectar objetos 3D en una vista bidimensional. Con el uso de estas funciones puede simular diversos tipos de experiencias en 3D.

El sistema de coordenadas 3D utilizado por ActionScript difiere de otros sistemas. Cuando se utilizan coordenadas 2D en ActionScript, el valor de x aumenta conforme se desplaza hacia la derecha por el eje x y el valor de y aumenta conforme desciende por el eje y. El sistema de coordenadas 3D conserva estas convenciones y añade un eje z cuyo valor aumenta conforme se aleja del punto de vista.



Direcciones positivas de los ejes x, y y z en el sistema de coordenadas 3D de ActionScript.
A. *Eje + Z* ***B.*** *Origen* ***C.*** *+Eje X* ***D.*** *+Eje Y*

Nota: *se debe tener en cuenta que Flash Player y AIR siempre representan 3D en capas. Esto significa que si el objeto A está delante del objeto B en la lista de visualización, Flash Player o AIR siempre representan A delante de B independientemente de los valores del eje z de los dos objetos. Para resolver este conflicto entre el orden de la lista de visualización y el orden del eje z, utilice el método* `transform.getRelativeMatrix3D()` *para guardar y posteriormente reordenar las capas de los objetos de visualización 3D. Para obtener más información, consulte "Uso de objetos Matrix3D para reordenar la visualización" en la página 380.*

Las siguientes clases de ActionScript admiten las nuevas funciones relacionadas con 3D:

1. La clase flash.display.DisplayObject contiene la propiedad `z` y nuevas propiedades de escala y rotación para manipular objetos de visualización en el espacio 3D. El método `DisplayObject.local3DToGlobal()` ofrece una forma sencilla de proyectar geometría 3D en un plano 2D.
2. La clase flash.geom.Vector3D se puede utilizar como estructura de datos para administrar puntos 3D. También admite matemáticas vectoriales.
3. La clase flash.geom.Matrix3D admite transformaciones complejas de geometría 3D como, por ejemplo, rotación, escala y traslación.
4. La clase flash.geom.PerspectiveProjection controla los parámetros para el mapeado de geometría 3D en una vista 2D.

Existen dos enfoques diferentes en la simulación de imágenes 3D en ActionScript:

1. Organización y animación de objetos planos en un espacio 3D. Este enfoque implica la animación de objetos de visualización utilizando las propiedades x, y y z de los objetos, o bien, estableciendo propiedades de rotación y escala mediante la clase DisplayObject. Se puede obtener un movimiento más complejo utilizando el objeto DisplayObject.transform.matrix3D. El objeto DisplayObject.transform.perspectiveProjection personaliza el modo en que los objetos de visualización se dibujan en perspectiva 3D. Utilice este enfoque cuando desee animar objetos 3D que consten de planos fundamentalmente. Los ejemplos de este enfoque incluyen galerías de imágenes 3D u objetos de animación 2D organizados en un espacio 3D.

2 Generación de triángulos 2D a partir de geometría 3D y representación de estos triángulos con texturas. Para utilizar este enfoque, en primer lugar debe definir y administrar datos sobre objetos 3D y posteriormente convertir estos datos en triángulos 2D para la representación. Las texturas de mapa de bits se pueden asignar a estos triángulos y a continuación los triángulos se dibujan en un objeto graphics utilizando el método `Graphics.drawTriangles()`. Los ejemplos de este enfoque incluyen la carga de datos del modelo 3D desde un archivo y la representación del modelo en pantalla, o bien, la generación y el dibujo de terreno 3D como mallas de triángulo.

Creación y movimiento de objetos de visualización 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Para convertir un objeto de visualización 2D en un objeto de visualización 3D, puede establecer su propiedad `z` de forma explícita en un valor numérico. Cuando se asigna un valor a la propiedad `z`, se crea un nuevo objeto Transform para el objeto de visualización. Con la definición de las propiedades `DisplayObject.rotationX` o `DisplayObject.rotationY` también se crea un nuevo objeto Transform. El objeto Transform contiene una propiedad `Matrix3D` que determina el modo en que se representa el objeto de visualización en el espacio 3D.

En el siguiente código se establecen las coordenadas para un objeto de visualización denominado “leaf”:

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Estos valores, así como las propiedades que se derivan de los mismos, se pueden ver en la propiedad `matrix3D` del objeto Transform de la hoja:

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

Para obtener información sobre las propiedades del objeto Transform, consulte la clase Transform. Para obtener información sobre las propiedades del objeto Matrix3D, consulte la clase Matrix3D.

Movimiento de un objeto en el espacio 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Puede mover un objeto en un espacio 3D cambiando los valores de sus propiedades `x, y o z`. Cuando se cambia el valor de la propiedad `z`, el objeto parece acercarse o alejarse del espectador.

El siguiente código mueve dos elipses hacia delante y hacia atrás a lo largo de sus ejes z, cambiando el valor de sus propiedades `z` como respuesta a un evento. `ellipse2` se mueve más rápido que `ellipse1`: su propiedad `z` se vé aumentada por un múltiplo de 20 en cada evento Frame, mientras que la propiedad `z` de `ellipse1` se vé aumentada por un múltiplo de 10:

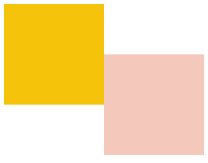
```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

Rotación de un objeto en el espacio 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Se puede girar un objeto de tres formas distintas, dependiendo del modo en que se establezcan las propiedades de rotación del objeto: `rotationX`, `rotationY` y `rotationZ`.

La siguiente figure muestra dos cuadrados que no se giran:

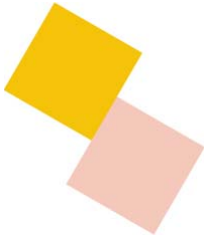


La siguiente figura muestra los dos cuadrados cuando se incrementa la propiedad `rotationY` del contenedor de los cuadrados para girarlos en el eje y. Con la rotación del contenedor, u objeto de visualización principal, de los dos cuadrados giran ambos cuadrados:

```
container.rotationY += 10;
```

En la siguiente figura se muestra qué sucede cuando se establece la propiedad `rotationX` del contenedor de los cuadrados. Con ello se giran los cuadrados en el eje x.

En la siguiente figura se muestra qué sucede cuando se incrementa la propiedad `rotationZ` del contenedor de los cuadrados. Con ello se giran los cuadrados en el eje z.



Un objeto de visualización se puede mover y girar simultáneamente en un espacio 3D.

Proyección de objetos 3D en una vista bidimensional

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La clase PerspectiveProjection del paquete `flash.geom` proporciona una forma sencilla de aplicar una perspectiva básica al mover objetos de visualización en un espacio 3D.

Si no crea de forma explícita una proyección en perspectiva para el espacio 3D, el motor 3D utiliza un objeto PerspectiveProjection predeterminado que se encuentra en la raíz y se propaga a todos sus elementos secundarios.

Las tres propiedades que definen cómo muestra un objeto PerspectiveProjection el espacio 3D son las siguientes:

- `fieldOfView`
- `projectionCenter`
- `focalLength`

Con la modificación del valor de `fieldOfView`, cambia automáticamente el valor de `focalLength` y viceversa, ya que son interdependientes.

La fórmula utilizada para calcular `focalLength` dado el valor de `fieldOfView` es la siguiente:

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

Generalmente se modificaría la propiedad `fieldOfView` de forma explícita.

Campo visual

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Con la manipulación de la propiedad `fieldOfView` de la clase PerspectiveProjection, puede hacer que un objeto de visualización 3D que se acerca al espectador parezca más grande y que un objeto que se aleja del espectador parezca más pequeño.

La propiedad `fieldOfView` especifica un ángulo entre 0 y 180 grados que determina la intensidad de la proyección en perspectiva. Cuanto mayor es el valor, mayor es la distorsión aplicada a un objeto de visualización que se mueve por su eje z. Un valor `fieldOfView` bajo implica un ajuste de escala muy pequeño y hace que objeto solo parezca moverse ligeramente hacia atrás en el espacio. Un valor `fieldOfView` alto causa más distorsión y la apariencia de mayor movimiento. El valor máximo de 179,9999... tiene como resultado un efecto de lente de cámara "ojo de pez" extremo. El valor máximo de `fieldOfView` es 179.9999... y el mínimo es de 0,00001... Exactamente 0 y 180 son valores no permitidos.

Centro de proyección

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `projectionCenter` representa el punto de fuga en la proyección en perspectiva. Se aplica como desplazamiento en el punto de registro predeterminado (0,0) en la esquina superior izquierda del escenario.

Cuando un objeto parece alejarse del espectador, se sesga hacia el punto de fuga y finalmente desaparece. Imagine un pasillo infinitamente largo. Cuando mira el pasillo hacia delante, los bordes de las paredes convergen en un punto de fuga hacia el final del pasillo.

Si el punto de fuga está en el centro del escenario, el pasillo desaparece hacia un punto en el centro. El valor predeterminado de la propiedad `projectionCenter` es el centro del escenario. Si, por ejemplo, desea que los elementos aparezcan en la parte izquierda del escenario y un área 3D a la derecha, establezca `projectionCenter` en un punto en la derecha del escenario para que ese sea el punto de fuga del área de visualización 3D.

Distancia focal

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `focalLength` representa la distancia entre el origen del punto de vista (0,0,0) y la ubicación del objeto de visualización en su eje z.

Una distancia focal larga es como una lente telefoto con una vista estrecha y distancias comprimidas entre objetos. Una distancia focal corta es como una lente gran angular, con la que se obtiene una vista amplia con mucha distorsión. La distancia focal media se aproxima a lo que vé el ojo humano.

Generalmente el valor `focalLength` se vuelve a calcular de forma dinámica durante la transformación de perspectiva conforme se mueve el objeto de visualización, aunque se puede establecer de forma explícita.

Valores de proyección en perspectiva predeterminados

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El objeto predeterminado PerspectiveProjection creado en la raíz tiene los siguientes valores:

- `fieldOfView: 55`
- `perspectiveCenter: stagewidth/2, stageHeight/2`
- `focalLength: stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Estos son los valores que se utilizan si no crea su propio objeto PerspectiveProjection.

Puede crear una instancia de su propio objeto PerspectiveProjection con el fin realizar una modificación propia de `projectionCenter` y `fieldOfView`. En este caso, los valores predeterminados del objeto recién creado son los siguientes, en función del tamaño del escenario predeterminado de 500 por 500:

- `fieldOfView: 55`
- `perspectiveCenter: 250,250`
- `focalLength: 480.24554443359375`

Ejemplo: Proyección en perspectiva

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

En el siguiente ejemplo se muestra el uso de la proyección en perspectiva para crear espacio 3D. Muestra cómo se puede modificar el punto de fuga y cambiar la proyección en perspectiva del espacio mediante la propiedad `projectionCenter`. Esta modificación implica volver a calcular los valores `focalLength` y `fieldOfView` con su distorsión añadida del espacio 3D.

En este ejemplo:

1. Se crea el objeto denominado `center`, como un círculo con cruz.
2. Asigna la coordenadas del objeto Sprite `center` a la propiedad`projectionCenter` de la propiedad `perspectiveProjection` de `transform` de la raíz.
3. Se añaden detectores de eventos para eventos de ratón que llaman a controladores que modifican el componente `projectionCenter` de forma que este siga la ubicación del objeto `center`.
4. Se crean cuatro cuadros de estilo Accordion que forman las paredes del espacio en perspectiva.

Cuando pruebe este ejemplo, ProjectionDragger.swf, arrastre el círculo alrededor de las distintas ubicaciones. El punto de fuga sigue el círculo, situándose donde lo coloque. Observe los cuadros que delimitan la extensión del espacio y aplique la distorsión cuando aleje el centro de proyección del centro del escenario.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación ProjectionDragger se encuentran en la carpeta Samples/ProjectionDragger.

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCCCFF));
```

```
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int
= 50, color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            inDrag = false;
        }
    }
}
```

Para obtener proyecciones en perspectiva más complejas, utilice la clase Matrix3D.

Transformaciones 3D complejas

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La clase Matrix3D permite transformar puntos 3D en un espacio de coordenadas o mapear puntos 3D de un espacio de coordenadas a otro.

Para utilizar la clase Matrix3D no es necesario conocer las matemáticas de matrices. La mayoría de las operaciones comunes de transformación se pueden controlar utilizando los métodos de la clase. No tiene que preocuparse por establecer o calcular de forma explícita los valores de todos los elementos de la matriz.

Una vez establecida la propiedad `z` de un objeto de visualización en un valor numérico, puede recuperar la matriz de transformación del objeto utilizando la propiedad Matrix3D del objeto Transform del objeto de visualización:

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

Puede utilizar los métodos del objeto Matrix3D para realizar la traslación, rotación, escala y proyección en perspectiva del objeto de visualización.

Utilice la clase Vector3D con sus propiedades `x`, `y`, y `z` para administrar puntos 3D. También puede representar un vector espacial en física, que cuenta con una dirección y una magnitud. Los métodos de la clase Vector3D permiten realizar cálculos comunes con vectores espaciales como, por ejemplo, cálculos de producto escalar, producto cruzado y suma.

Nota: *la clase Vector3D no se relaciona con la clase Vector de ActionScript. La clase Vector3D contiene propiedades y métodos para definir y manipular puntos 3D, mientras que la clase Vector admite conjuntos de objetos con tipo.*

Creación de objetos Matrix3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Existen tres formas principales de crear o recuperar objetos `Matrix3D`:

- Utilice el método constructor `Matrix3D()` para crear una instancia de una nueva matriz. El constructor `Matrix3D()` adopta un objeto `Vector` que contiene 16 valores numéricos y sitúa cada valor en una celda de la matriz. Por ejemplo:

  ```
  var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
  ```

- Establezca el valor de la propiedad `z` de un objeto de visualización. A continuación recupere la matriz de transformación de la propiedad `transform.matrix3D` de ese objeto.
- Recupere el objeto Matrix3D que controla la visualización de objetos 3D en el escenario mediante una llamada al método `perspectiveProjection.toMatrix3D()` en el objeto de visualización raíz.

Aplicación de varias transformaciones 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Puede aplicar varias transformaciones 3D a la vez utilizando un objeto Matrix3D. Por ejemplo, si desea girar, escalar y mover un cubo, puede aplicar tres transformaciones independientes a cada punto del cubo. No obstante, resulta mucho más eficaz calcular previamente varias transformaciones en un objeto Matrix3D y posteriormente realizar una transformación de matriz en cada uno de los puntos.

Nota: *el orden en que se aplican las transformaciones de matriz es importante. Los cálculos de matriz no son conmutativos. Por ejemplo, con la aplicación de una rotación seguida de una traslación se obtiene un resultado diferente de la aplicación de la misma traslación seguida de la misma rotación.*

En el siguiente ejemplo se muestran dos formas de realizar varias transformaciones 3D.

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);
```

```
            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

En el método `doTransforms()` el primer bloque de código utiliza las propiedades DisplayObject para cambiar la rotación, escala y posición de una forma de rectángulo. El segundo bloque de código utiliza los métodos de la clase Matrix3D para realizar las mismas transformaciones.

La principal ventaja del uso del método `Matrix3D` es que todos los cálculos se realizan en la matriz en primer lugar. Después se aplican al objeto de visualización solo una vez, cuando se establece su propiedad `transform.matrix3D`. La definición de las propiedades DisplayObject hace que el código fuente resulte un poco más sencillo de leer. No obstante, cada vez que se establece una propiedad de escala o rotación, esto causa varios cálculos y cambia varias propiedades del objeto de visualización.

Si el código aplica las mismas transformaciones completas a los objetos de visualización varias veces, guarde el objeto Matrix3D como una variable y posteriormente vuelva a aplicarlo una y otra vez.

Uso de objetos Matrix3D para reordenar la visualización

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Tal y como se indicó anteriormente, el orden de la disposición en capas de los objetos en la lista de visualización determina el orden de las capas de visualización, independientemente de sus ejes z relativos. Si la animación transforma las propiedades de los objetos de visualización en un orden que difiere del de la lista de visualización, el espectador podría ver una disposición en capas de los objetos que no se corresponde con las capas del eje z. Por lo tanto, un objeto que deba aparecer alejado del espectador puede aparecer frente a un objeto que esté más cerca del espectador.

Para garantizar que la disposición en capas de los objetos de visualización 3D se corresponde con las profundidades relativas de los objetos, utilice un enfoque como el siguiente:

1. Utilice el método `getRelativeMatrix3D()` del objeto Transform para obtener los `ejes z` relativos de los objetos de visualización 3D secundarios.
2. Use el método `removeChild()` para eliminar los objetos de la lista de visualización.
3. Ordene los objetos de visualización en función de sus valores de eje z.
4. Utilice el método `addChild()` para volver a añadir los elementos secundarios a la lista de visualización en orden inverso.

Este reordenamiento garantiza que los objetos se muestren conforme a sus ejes z relativos.

El siguiente código muestra la correcta visualización de las seis caras de un cuadro 3D. Reordena las caras del cuadro tras haber aplicado las rotaciones:

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación se encuentran en la carpeta Samples/ReorderByZ.

Uso de triángulos para efectos 3D

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

En ActionScript, las transformaciones de mapa de bits se realizan utilizando el método `Graphics.drawTriangles()`, ya que los modelos 3D se representan mediante una colección de triángulos en el espacio. (No obstante, Flash Player y AIR no admiten un búfer de profundidad, por lo que los objetos de visualización aún son esencialmente planos o bidimensionales. Esto se describe en "Información sobre los objetos de visualización 3D en el motor de ejecución de Flash Player y AIR" en la página 369.) El método `Graphics.drawTriangles()` es similar a `Graphics.drawPath()`, ya que adopta un conjunto de coordenadas para dibujar un trazado de triángulo.

Para familiarizarse con el uso de `Graphics.drawPath()`, consulte "Trazados de dibujo" en la página 244.

El método `Graphics.drawTriangles()` utiliza Vector.<Number> para especificar las ubicaciones de punto para el trazado del triángulo:

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

El primer parámetro de `drawTriangles()` es el único necesario: el parámetro `vertices`. Este parámetro es un vector de números que define las coordenadas mediante las que se dibujan los triángulos. Cada tres conjuntos de coordenadas (seis números) se representa un trazado del triángulo. Sin el parámetro `indices`, la longitud del vector siempre debe ser un factor de seis, ya que cada triángulo requiere tres pares de coordenadas(tres conjuntos de dos valores x/y). Por ejemplo:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

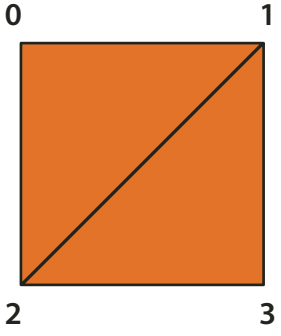
Ninguno de estos triángulos comparte puntos, pero si lo hicieran, el segundo parámetro `drawTriangles()`, `indices`, podría utilizarse para volver a usar los valores en el vector `vertices` para varios triángulos.

Al utilizar el parámetro `indices`, se debe tener en cuenta que los valores `indices` son índices de punto, y no índices que se relacionan directamente con los elementos del conjunto `vertices`. Es decir, un índice del vector `vertices` tal y como se define mediante `indices` es realmente el verdadero índice dividido por 2. Para el tercer punto de un vector `vertices` utilice, por ejemplo, un valor `indices` de 2, aunque el primer valor numérico de ese punto comience en el índice vectorial de 4.

Por ejemplo, combine dos triángulos para que compartan el borde diagonal utilizando el parámetro `indices`:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Se debe tener en cuenta que aunque ahora se haya dibujado un cuadrado utilizando dos triángulos, solo se especificaron cuatro puntos en el vector `vertices`. Con el uso de `indices`, los dos puntos compartidos por los dos triángulos se volverán a utilizar para cada triángulo. Con ello se reduce el recuento total de vértices de 6 (12 números) a 4 (8 números):



Cuadrado dibujado con dos triángulos utilizando el parámetro de vértices

Esta técnica resulta útil con mallas de triángulo más grandes donde varios triángulos comparten la mayoría de los puntos.

Todos los rellenos se pueden aplicar a triángulos. Los rellenos se aplican a la malla de triángulo resultante tal y como se haría en cualquier otra forma.

Transformación de mapas de bits

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Las transformaciones de mapa de bits proporcionan la ilusión de perspectiva o "textura" en un objeto tridimensional. Concretamente, es posible distorsionar un mapa de bits hacia un punto de fuga de forma que la imagen parezca encogerse conforme se acerca al punto de fuga. O bien, puede utilizar un mapa de bits bidimensional para crear una superficie para un objeto tridimensional, ofreciendo la ilusión de textura o “envoltura” en dicho objeto.

Superficie bidimensional donde se utiliza un punto de fuga y tres objetos tridimensionales ajustados con un mapa de bits

Mapeado UV

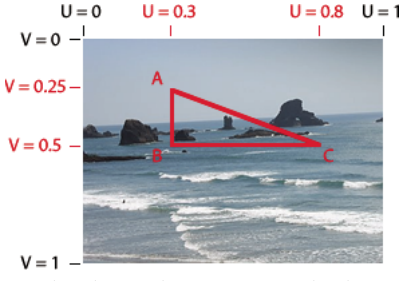
Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Una vez que comience a trabajar con texturas, deseará utilizar el parámetro uvtData de `drawTriangles()`. Este parámetro permite configurar el mapeado UV para rellenos de mapa de bits.

El mapeado UV es un método para aplicar textura a los objetos. Se basa en dos valores: un valor U horizontal (x) y un valor V vertical (y). En lugar de basarse en valores de píxel, se basan en porcentajes. 0 U y 0 V representan la parte superior izquierda de una imagen y 1 U y 1 V la parte inferior derecha:

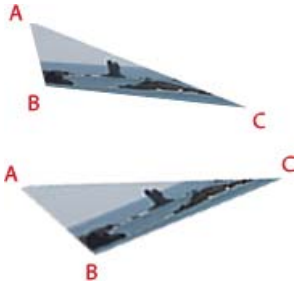
Ubicaciones UV 0 y 1 en una imagen de mapa de bits

A los vectores de un triángulo se les puede proporcionar coordenadas UV para asociarse con las ubicaciones respectivas en una imagen:



Coordenadas UV de un área triangular de una imagen de mapa de bits

Los valores UV coinciden con los puntos del triángulo:



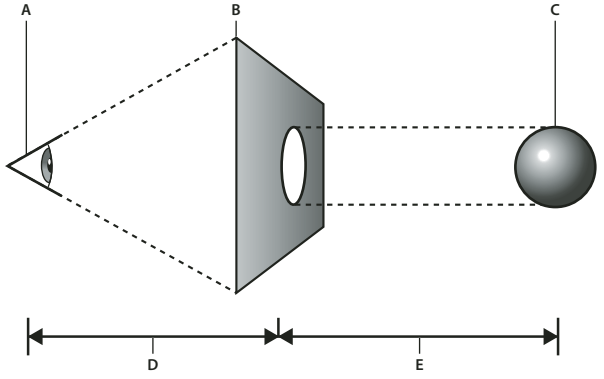
Los vértices del triángulo se mueven y el mapa de bits se distorsiona para mantener igual los valores UV para un punto individual

Debido a que las transformaciones 3D de ActionScript se aplican al triángulo asociado con el mapa de bits, la imagen del mapa de bits se aplica al triángulo en función de los valores UV. Por lo tanto, en lugar de utilizar cálculos de matriz, establezca o ajuste los valores UV para crear un efecto tridimensional.

El método `Graphics.drawTriangles()` también acepta una parte de información opcional para transformaciones tridimensionales: el valor T. El valor T en uvtData representa la perspectiva 3D, o más concretamente, el factor de escala del vértice asociado. El mapeado UVT añade una corrección de la perspectiva al mapeado UV. Por ejemplo, si un objeto se coloca en un espacio 3D alejado del punto de vista de forma que parezca ser un 50% de su tamaño "original", el valor T del objeto podría ser 0,5. Debido a que los triángulos se dibujan para representar objetos en el espacio 3D, sus ubicaciones en el eje z determinan sus valores T. La ecuación que determina el valor T es la siguiente:

```
T = focalLength/(focalLength + z);
```

En esta ecuación, focalLength representa una distancia focal o ubicación de "pantalla" calculada que determina la cantidad de perspectiva que se ofrece en la vista.



Distancia focal y valor z
***A.** Punto de vista **B.** pantalla **C.** Objeto 3D **D.** valor focalLength **E.** valor z*

El valor de T se utiliza para realizar un ajuste de escala de formas básicas para hacerlas parecer más alejadas en la distancia. Suele ser el valor utilizado para convertir puntos 3D en puntos 2D. En el caso de datos UVT, también se utiliza para escalar un mapa de bits entre los puntos en un triángulo con perspectiva.

Cuando se definen valores UVT, el valor T sigue directamente los valores UV definidos para un vértice. Con la inclusión de T, cada tres valores en el parámetro `uvtData` (U, V y T) coinciden con cada dos valores del parámetro `vertices` (x e y). Solo con los valores UV, uvtData.length == vertices.length. Con la inclusión de un valor T, uvtData.length = 1.5*vertices.length.

En el siguiente ejemplo se muestra un plano giratorio en un espacio 3D utilizando datos UVT. Este ejemplo utiliza una imagen llamada ocean.jpg y una clase "auxiliar", ImageLoader, para cargar la imagen ocean.jpg de modo que se pueda asignar al objeto BitmapData.

A continuación se muestra el código fuente de la clase ImageLoader (guarde este código en un archivo denominado ImageLoader.as):

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

Y a continuación se muestra el código ActionScript que utiliza triángulos, mapeado UV y valores T para hacer que la imagen parezca que está rotando y encogiéndose hacia un punto de fuga. Guarde este código en un archivo denominado Spinning3dOcean.as:

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);

            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
```

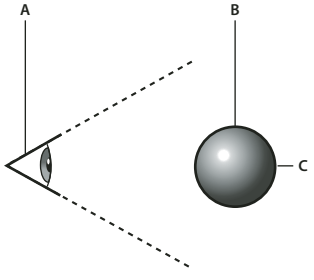
```
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

Para probar este ejemplo, guarde los dos archivos de clase en el mismo directorio que una imagen llamada “ocean.jpg”. Puede ver cómo se transforma el mapa de bits original para que parezca que se aleja en la distancia y gira en un espacio 3D.

Técnica "culling"

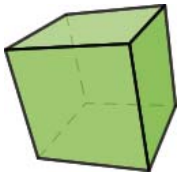
Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La técnica "culling" es el proceso que determina qué superficies de un objeto tridimensional no debe representar el procesador porque están ocultas desde el punto de vista actual. En el espacio 3D, la superficie de la parte “posterior” de un objeto tridimensional queda oculta desde el punto de vista:



La parte posterior de un objeto 3D se oculta desde el punto de vista.
***A.** Punto de vista **B.** Objeto 3D **C.** Parte posterior de un objeto tridimensional*

Los triángulos siempre se representan independientemente de su tamaño, forma o posición. La técnica "culling" garantiza que Flash Player o AIR representen los objetos 3D correctamente. Asimismo, para guardar los ciclos de representación, a veces se desea que el procesador omita algunos triángulos. Pongamos como ejemplo un cubo giratorio en el espacio. En un momento dado, nunca verá más de tres caras del cubo, ya que las caras ocultas estarán frente a la otra dirección del otro lado del cubo. Debido a que estas caras no se van a ver, el procesador no debe dibujarlas. Sin "culling", Flash o AIR representan tanto las caras frontales como las posteriores.



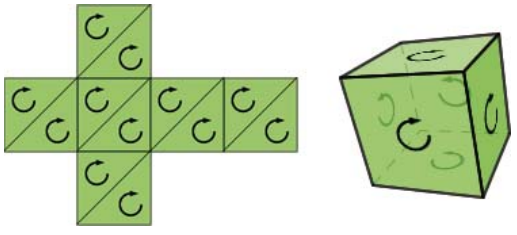
Un cubo tiene caras no visibles desde el punto de vista actual

Por lo tanto, el método `Graphics.drawTriangles()` tiene un cuarto parámetro para establecer un valor de "culling":

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

El parámetro de "culling" es un valor de la clase de enumeración `TriangleCulling`: `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` y `TriangleCulling.NEGATIVE`. Estos valores son dependientes de la dirección del trazado del triángulo que define la superficie del objeto. La API de ActionScript para determinar el valor "culling" da por hecho que todos los triángulos exteriores de una forma 3D se dibujan con la misma dirección de trazado. Cuando una cara del triángulo cambia de dirección, la dirección del trazado también se modifica. En este momento, al triángulo se le puede aplicar el proceso de "culling" (eliminarse de la representación).

Por lo tanto, un valor `TriangleCulling` de `POSITIVE` elimina triángulos con una dirección de trazado positiva (en el sentido de las agujas del reloj). Un valor `TriangleCulling` de `NEGATIVE` elimina triángulos con una dirección de trazado negativa (en sentido contrario a las agujas del reloj). En el caso de un cubo, mientras que las caras frontales tienen una dirección de trazado positiva, las posteriores cuentan con una dirección de trazado negativa:



Cubo "sin ajustar" para mostrar la dirección de trazado. Cuando se "ajusta", se invierte la dirección de trazado de la cara posterior

Para ver cómo funciona la técnica "culling", comience con el ejemplo anterior de "Mapeado UV" en la página 383, establezca el parámetro de "culling" del método`drawTriangles()` en `TriangleCulling.NEGATIVE`:

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Observe que la parte "posterior" de la imagen no se procesa conforme gira el objeto.

Capítulo 20: Fundamentos de la utilización de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para mostrar texto en pantalla en Adobe® Flash® Player o Adobe® AIR™, se utiliza una instancia de la clase TextField o las clases de Flash Text Engine. Estas clases permiten crear texto, presentarlo y aplicarle formato. Como alternativa, puede utilizar Text Layout Framework (TLF), una biblioteca de componentes basada en las clases de Flash Text Engine, pero diseñada para facilidad de uso. En dispositivos móviles es posible utilizar la clase StageText para introducir texto.

Se puede establecer el contenido específico para los campos de texto, o bien, designar el origen del texto y posteriormente definir su aspecto. También se puede responder a eventos de usuario, como cuando el usuario introduce texto o hace clic en un vínculo de hipertexto.

Tanto la clase TextField como la clases de Flash Text Engine permiten mostrar y administrar texto en Flash Player y AIR. La clase TextField se puede utilizar para crear objetos de texto para visualización y entrada. La clase TextField proporciona la base para otros componentes basados en texto como, por ejemplo, TextArea y TextInput. La clase TextFormat se puede utilizar para establecer el formato de párrafo y caracteres para los objetos TextField y se pueden aplicar hojas de estilos en cascada (CSS) con el uso de la propiedad Textfield.styleSheet y de la clase StyleSheet. Se puede asignar un texto con formato HTML, que puede contener elementos multimedia incorporados (clips de pelí-cula y archivos SWF, GIF, PNG y JPEG), directamente a un campo de texto.

Flash Text Engine, disponible comenzando con Flash Player 10 y Adobe AIR 1.5, ofrece una compatibilidad de bajo nivel para realizar un sofisticado control de las medidas y el formato del texto, además de admitir texto bidireccional. También proporciona una compatibilidad para idiomas y un flujo de texto mejorados. Aunque Flash Text Engine se puede utilizar para crear y administrar elementos de texto, principalmente está diseñado como base para la creación de componentes de gestión de texto y requiere una experta labor de tareas de programación. Text Layout Framework, que incluye un componente de gestión de texto basado en FTE, ofrece una forma más segura de utilizar sus funciones avanzadas. Text Layout Framework es una biblioteca ampliable incorporada por completo en ActionScript 3.0. Es posible utilizar el componente TLF existente o la arquitectura para crear un componente de texto propio.

La clase StageText, disponible desde AIR 3, proporciona un campo de entrada de texto nativo. Como este campo lo proporciona el sistema operativo del dispositivo, permite que la experiencia de los usuarios sea mucho más familiar al integrarse en el dispositivo. Una instancia de StageText no es un objeto de visualización. En vez de añadirse a la lista de visualización, se debe asignar un escenario a la instancia y un área de visualización en dicho escenario denominada puerto. La instancia de StageText se visualiza delante de cualquier objeto de visualización.

Para obtener más información sobre estos temas, consulte:

- "Uso de la clase TextField" en la página 391
- "Uso de Flash Text Engine" en la página 416
- "Uso de Text Layout Framework" en la página 446
- Native text input with StageText (Introducción de texto nativo con StageText, en inglés)

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes relacionados con el control de texto:

Hojas de estilos en cascada Sintaxis estándar para especificar estilos y formato para contenido estructurado con formato XML (o HTML).

Fuente de dispositivo Fuente instalada en el equipo del usuario.

Campo de texto dinámico Campo de texto cuyo contenido se puede modificar mediante código ActionScript pero no con una entrada de usuario.

Fuente incorporada Fuente que tiene los datos de contorno de caracteres almacenados en el archivo SWF de la aplicación.

Texto HTML Contenido de texto introducido en un campo de texto mediante código ActionScript que incluye etiquetas de formato HTML junto con el contenido de texto.

Campo de texto de entrada Campo de texto cuyo contenido se puede modificar mediante una entrada de usuario o código ActionScrip.

Espaciado manual Ajuste del espaciado entre pares de caracteres para que el espaciado de las palabras sea más proporcional y se facilite la lectura del texto.

Campo de texto estático Campo de texto creado en la herramienta de edición, cuyo contenido no se puede modificar mientras se ejecuta el archivo SWF.

Medidas de lí-neas de texto Medidas de las diversas partes del contenido de texto de un campo de texto, como la lí-nea de base del texto, la altura de la parte superior de los caracteres, el tamaño de los trazos descendentes (la parte de algunas letras minúsculas que se extiende por debajo de la línea de base), etc.

Espaciado entre caracteres Ajuste del espaciado entre grupos de letras o bloques de texto para aumentar o disminuir la densidad y hacer el texto más legible.

Capítulo 21: Uso de la clase TextField

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede utilizar una instancia de la clase TextField para mostrar texto o crear un campo de entrada de texto en la pantalla en Adobe® Flash® Player o Adobe® AIR™. La clase TextField es la base para otros componentes basados en texto como, por ejemplo, los componentes TextArea o TextInput.

El contenido de un campo de texto se puede especificar previamente en el archivo SWF, se puede cargar desde un archivo de texto o una base de datos o puede ser introducido por un usuario que interactúe con la aplicación. En un campo de texto, el texto puede aparecer como contenido HTML representado con imágenes incorporadas. Una vez creada una instancia de un campo de texto, se pueden utilizar las clases flash.text, como TextFormat y StyleSheet, para controlar la apariencia del texto. El paquete flash.text contiene casi todas las clases relacionadas con la creación, administración y formato de texto en ActionScript.

Se puede aplicar formato a texto definiendo el formato con un objeto TextFormat y asignando dicho objeto al campo de texto. Si el campo de texto contiene texto HTML, se puede aplicar un objeto StyleSheet al campo de texto para asignar estilos a partes específicas del contenido del campo de texto. Los objetos TextFormat o StyleSheet contienen propiedades que definen la apariencia del texto, como el color, el tamaño y el grosor. El objeto TextFormat asigna las propiedades a todo el contenido de un campo de texto o a un rango de texto. Por ejemplo, en un mismo campo de texto una frase puede estar en negrita y de color rojo y la frase siguiente en cursiva y de color azul.

Además de las clases del paquete flash.text, se puede utilizar la clase flash.events.TextEvent para responder a acciones del usuario relacionadas con texto.

Más temas de ayuda

"Asignación de formatos de texto" en la página 399

"Visualización de texto HTML" en la página 393

"Aplicación de hojas de estilos en cascada" en la página 399

Visualización de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Aunque las herramientas de edición como Adobe Flex Builder y Flash Professional proporcionan varias opciones para visualizar texto, como componentes relacionados con texto o herramientas de texto, la principal manera de mostrar texto mediante programación es utilizar un campo de texto.

Tipos de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El tipo de texto de un campo de texto se caracteriza por su origen:

- Texto dinámico

  El texto dinámico incluye contenido que se carga desde un origen externo, como un archivo de texto, un archivo XML o incluso un servicio web remoto.

- Texto de entrada

  El texto de entrada es cualquier texto introducido por un usuario o texto dinámico que un usuario puede editar. Se puede establecer una hoja de estilos para aplicar formato al texto de entrada o utilizar la clase flash.text.TextFormat para asignar propiedades al campo de texto para el contenido de la entrada. Para obtener más información, consulte "Captura de entradas de texto" en la página 397.

- Texto estático

  El texto estático solo se crea mediante Flash Professional. Con ActionScript 3.0 no se pueden crear instancias de texto estático. Sin embargo, se pueden usar clases de ActionScript como StaticText y TextSnapshot para manipular instancias de texto estático existentes. Para obtener más información, consulte "Trabajo con texto estático" en la página 405.

Modificación del contenido de un campo de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El usuario puede definir texto dinámico asignando una cadena a la propiedad `flash.text.TextField.text`. Se puede asignar una cadena directamente a la propiedad de la siguiente manera:

```
myTextField.text = "Hello World";
```

También se puede asignar a la propiedad `text` un valor de una variable definida en el script, tal y como se muestra en el siguiente ejemplo:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

Como alternativa, se puede asignar a la propiedad `text` un valor de una variable remota. Existen tres opciones para cargar valores de texto desde orí-genes remotos:

- Las clases flash.net.URLLoader y flash.net.URLRequest cargan variables para el texto desde una ubicación local o remota.
- El atributo `FlashVars` está incorporado en la página HTML que aloja el archivo SWF y puede incluir valores para variables de texto.
- La clase flash.net.SharedObject administra el almacenamiento persistente de valores. Para obtener más información, consulte "Almacenamiento de datos locales" en la página 731.

Visualización de texto HTML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase flash.text.TextField cuenta con una propiedad `htmlText` que se puede utilizar para identificar la cadena de texto como cadena que contiene etiquetas HTML para aplicar formato al contenido. Tal y como se muestra en el ejemplo siguiente, se debe asignar el valor de cadena a la propiedad `htmlText` (no a la propiedad `text`) para que Flash Player o AIR representen el texto en formato HTML:

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player y AIR admiten un subconjunto de etiquetas y entidades HTML para la propiedad `htmlText`. La descripción de la propiedad `flash.text.TextField.htmlText` en Referencia de ActionScript 3.0 proporciona información detallada sobre las etiquetas y entidades HTML admitidas.

Una vez designado el contenido mediante la propiedad `htmlText`, puede utilizar hojas de estilos o la etiqueta `textformat` para administrar el formato del contenido. Para obtener más información, consulte "Formato de texto" en la página 399.

Uso de imágenes en campos de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Otra ventaja de mostrar el contenido como texto HTML es que se pueden incluir imágenes en el campo de texto. Se puede hacer referencia a una imagen, local o remota, mediante la etiqueta `img` y hacer que aparezca dentro del campo de texto asociado.

En el ejemplo siguiente se crea un campo de texto denominado `myTextBox` y se incluye una imagen JPG de un ojo, almacenada en el mismo directorio que el archivo SWF, dentro del texto mostrado:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and
<i>see</i></p><p><img src='eye.jpg' width='20' height='20'></p><p>what can be
rendered.</p><p>You should see an eye image and some <u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

La etiqueta `img` es compatible con archivos JPEG, GIF, PNG y SWF.

Desplazamiento de texto en un campo de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En muchos casos, el texto puede ser más largo que el campo de texto que lo muestra. O bien, se puede tener un campo de entrada que permita a un usuario introducir más texto que el que se puede mostrar de una sola vez. Se pueden utilizar las propiedades relacionadas con el desplazamiento de la clase flash.text.TextField para administrar contenido extenso, tanto verticalmente como horizontalmente.

Entre las propiedades relacionadas con el desplazamiento se incluyen `TextField.scrollV`, `TextField.scrollH`, `maxScrollV` y `maxScrollH`. Estas propiedades pueden utilizarse para responder a eventos, como un clic de ratón o una pulsación de tecla.

En el ejemplo siguiente se crea un campo de texto con un tamaño establecido y que contiene más texto que el campo puede mostrar de una sola vez. A medida que el usuario hace clic en el campo de texto, el texto se desplaza verticalmente.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to
meet you. Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go
home now. Don't forget to tip your waiter. There are mints in the bowl by the door. Thank you.
Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

Selección y manipulación de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede seleccionar texto dinámico o de entrada. Debido a que las propiedades y los métodos de selección de texto de la clase TextField utilizan posiciones de í-ndice para establecer el rango de texto que se va a manipular, se puede seleccionar mediante programación texto dinámico o de entrada aunque no se conozca su contenido.

Nota: *en Flash Professional, si se elige la opción seleccionable en un campo de texto estático, el campo de texto que se exporta y se coloca en la lista de visualización es un campo de texto dinámico normal.*

Selección de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El valor de la propiedad `flash.text.TextField.selectable` es `true` de forma predeterminada y se puede seleccionar texto mediante programación a través del método `setSelection()`.

Por ejemplo, se puede establecer texto especí-fico de un campo de texto que se debe seleccionar cuando el usuario haga clic en el campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

Del mismo modo, si se desea que se seleccione texto de un campo como el texto que se mostrará inicialmente, se debe crear una función de controlador de eventos a la que se llama cuando se añade texto a la lista de visualización.

Captura de texto seleccionado por el usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las propiedades `selectionBeginIndex` y `selectionEndIndex` de TextField, que son de "solo lectura" y por tanto no se pueden establecer para que seleccionen texto mediante programación, pueden utilizarse para capturar lo que el usuario haya seleccionado actualmente. Asimismo, los campos de entrada de texto pueden utilizar la propiedad `caretIndex`.

Por ejemplo, el código siguiente hace un seguimiento de los valores de índice de texto seleccionado por el usuario:

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first
and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

Se puede aplicar una colección de propiedades de objeto TextFormat a la selección para modificar el aspecto del texto. Para obtener más información sobre la aplicación de una colección de propiedades de TextFormat a texto seleccionado, consulte “Formato de rangos de texto en un campo de texto” en la página 402.

Captura de entradas de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De forma predeterminada, la propiedad `type` de un campo de texto se establece como `dynamic`. Si la propiedad `type` se establece en `input` mediante la clase TextFieldType, se puede obtener la entrada del usuario y guardar el valor para utilizarlo en otras partes de la aplicación. Los campos de entrada de texto resultan útiles para los formularios y cualquier aplicación que requieran que el usuario defina un valor de texto para utilizarlo en otro lugar del programa.

Por ejemplo, el código siguiente crea un campo de entrada de texto denominado `myTextBox`. A medida que el usuario escribe texto en el campo, se activa el evento `textInput`. Un controlador de eventos denominado `textInputCapture` captura la cadena de texto introducido y se la asigna a una variable. Flash Player o AIR muestran el nuevo texto en otro campo de texto, llamado `myOutputBox`.

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
```

```
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

Restricción de entradas de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Debido a que los campos de entrada de texto se suelen utilizar para formularios o cuadros de diálogo en aplicaciones, es posible que se desee limitar los tipos de caracteres que puede introducir el usuario o incluso mantener el texto oculto (por ejemplo, para una contraseña). La clase flash.text.TextField dispone de las propiedades `displayAsPassword` y `restrict` que se pueden establecer para controlar la entrada del usuario.

La propiedad `displayAsPassword` simplemente oculta el texto (mostrándolo como una serie de asteriscos) conforme lo escribe el usuario. Cuando `displayAsPassword` se establece en `true`, los comandos Cortar y Copiar y sus correspondientes métodos abreviados de teclado no funcionan. Tal y como se muestra en el siguiente ejemplo, la propiedad `displayAsPassword` se asigna de la misma manera que otras propiedades, como background y color:

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

La propiedad `restrict` es algo más compleja, ya que es necesario especificar qué caracteres puede escribir el usuario en un campo de entrada de texto. Se pueden permitir letras, números o intervalos de letras, números y caracteres específicos. El código siguiente permite al usuario escribir únicamente letras mayúsculas (y no números ni caracteres especiales) en el campo de texto:

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 utiliza guiones para definir rangos y caracteres ^ para definir caracteres excluidos. Para obtener más información sobre cómo definir restricciones en un campo de entrada de texto, consulte la entrada sobre la propiedad `flash.text.TextField.restrict` en la Referencia de ActionScript 3.0.

Nota: *si se utiliza la propiedad `flash.text.TextField.restrict`, el motor de ejecución convierte automáticamente las letras con caracteres restringidos a caracteres permitidos. Si utiliza la propiedad `fl.text.TLFTextField.restrict` (es decir, si utiliza un campo de texto TLF), el motor de ejecución no tendrá en cuenta las letras restringidas.*

Formato de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen varias opciones para aplicar formato a la visualización del texto mediante programación. Se pueden establecer propiedades directamente en la instancia de TextField; por ejemplo, las propiedades `TextFIeld.thickness`, `TextField.textColor` y `TextField.textHeight` .O bien, se puede designar el contenido del campo de texto mediante la propiedad `htmlText` y utilizar las etiquetas HTML admitidas, como `b`, `i` y `u`. Pero también se pueden aplicar objetos TextFormat a campos de texto que contienen texto sin formato u objetos StyleSheet a campos de texto que contienen la propiedad `htmlText` . El uso de objetos TextFormat y StyleSheet proporciona el mayor control y la mayor uniformidad de la apariencia del texto en toda la aplicación. Es posible definir un objeto TextFormat o StyleSheet y aplicarlo a diversos campos de texto de la aplicación o a todos ellos.

Asignación de formatos de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase TextFormat se puede utilizar para establecer varias propiedades de visualización de texto distintas y aplicarlas a todo el contenido de un objeto TextField o a un rango de texto.

En el ejemplo siguiente se aplica un objeto TextFormat a un objeto TextField completo y se aplica un segundo objeto TextFormat a un rango de texto dentro de ese objeto TextField:

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

El método `TextField.setTextFormat()` solo afecta al texto que ya se ha visualizado en el campo de texto. Si cambia el contenido del objeto TextField, la aplicación tendrá que volver a llamar al método `TextField.setTextFormat()` para aplicar de nuevo el formato. También se puede establecer la propiedad `defaultTextFormat` de TextField para especificar el formato que se debe utilizar para el texto introducido por el usuario.

Aplicación de hojas de estilos en cascada

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los campos de texto pueden incluir texto sin formato o texto con formato HTML. El texto sin formato se almacena en la propiedad `text` de la instancia y el texto HTML se almacena en la propiedad `htmlText`.

Se pueden utilizar declaraciones de estilos CSS para definir estilos de texto que pueden aplicarse a varios campos de texto distintos. Las declaraciones de estilos CSS pueden crearse en el código de la aplicación o cargarse en tiempo de ejecución desde un archivo CSS externo.

La clase flash.text.StyleSheet administra estilos CSS. La clase StyleSheet reconoce un conjunto limitado de propiedades CSS. Para ver una lista detallada de propiedades de estilos admitidas por la clase StyleSheet, consulte la entrada de flash.textStylesheet en la Referencia de ActionScript 3.0.

Tal y como se indica en el ejemplo siguiente, se pueden crear hojas de estilos CSS en el código y aplicar dichos estilos a texto HTML mediante un objeto StyleSheet:

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

Tras crear un objeto StyleSheet, el código de ejemplo crea un objeto simple para almacenar un conjunto de propiedades de declaración de estilos. Posteriormente llama al método `StyleSheet.setStyle()`, que añade el nuevo estilo a la hoja de estilos con el nombre ".darkred". Por último, aplica el formato de la hoja de estilos asignando el objeto StyleSheet a la propiedad `styleSheet`.

Para que los estilos CSS surtan efecto, la hoja de estilos debe aplicarse al objeto TextField antes de establecer la propiedad `htmlText`.

Nota: por diseño, los campos de texto con hojas de estilos no se pueden editar. Si se dispone de un campo de entrada de texto y se le asigna una hoja de estilos, el campo muestra las propiedades de la hoja de estilos, pero el campo de texto no permitirá a los usuarios escribir texto nuevo en él. Asimismo, las siguientes API de ActionScript no se pueden utilizar en un campo de texto con una hoja de estilos asignada:

- Método `TextField.replaceText()`
- Método `TextField.replaceSelectedText()`
- Propiedad `TextField.defaultTextFormat`
- Método `TextField.setTextFormat()`

Si se ha asignado una hoja de estilos a un campo de texto, pero después se establece la propiedad `TextField.styleSheet` con el valor `null`, el contenido de las propiedades `TextField.text` y `TextField.htmlText` añade etiquetas y atributos a su contenido para incorporar el formato de la hoja de estilos asignada anteriormente. Para conservar la propiedad `htmlText` original, debe guardarse en una variable antes de establecer la hoja de estilos en `null`.

Carga de un archivo CSS externo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El enfoque de utilizar CSS para el formato es más eficaz si se carga información CSS desde un archivo externo en tiempo de ejecución. Cuando los datos CSS son externos a la propia aplicación, se puede cambiar el estilo visual de texto en la aplicación sin tener que modificar el código fuente de ActionScript 3.0. Una vez implementada la aplicación, es posible modificar un archivo CSS externo para cambiar el aspecto de la aplicación sin tener que volver a implementar su archivo SWF.

El método `StyleSheet.parseCSS()` convierte una cadena que contiene datos CSS en declaraciones de estilo del objeto StyleSheet. En el siguiente ejemplo se muestra cómo leer un archivo CSS externo y aplicar sus declaraciones de estilo a un objeto TextField.

En primer lugar, este es el contenido del archivo CSS que se va a cargar, denominado example.css:

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

A continuación se muestra el código ActionScript de una clase que carga el archivo example.css y aplica los estilos al contenido de TextField:

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
```

```
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

Una vez cargados los datos CSS, se ejecuta el método `onCSSFileLoaded()` y se llama al método `StyleSheet.parseCSS()` para transferir las declaraciones de estilo al objeto StyleSheet.

Formato de rangos de texto en un campo de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un método especialmente útil de la clase flash.text.TextField es `setTextFormat()`. Con `setTextFormat()` se pueden asignar propiedades específicas al contenido de una parte de un campo de texto para responder a una entrada de usuario, como en el caso de los formularios que deben recordar a los usuarios que algunas entradas son obligatorias o para resaltar una parte de un texto dentro de un campo de texto a medida que el usuario selecciona partes del texto.

En el siguiente ejemplo se utiliza `TextField.setTextFormat()` en un rango de caracteres para cambiar el aspecto de una parte del contenido de `myTextField` cuando el usuario hace clic en el campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes
format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

Representación avanzada de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 proporciona diversas clases en el paquete flash.text para controlar las propiedades del texto mostrado, incluidas las fuentes incorporadas, la configuración de suavizado, el control del canal alfa y otras configuraciones específicas. La Referencia de ActionScript 3.0 proporciona descripciones detalladas de estas clases y propiedades, incluidas las clases CSMSettings, Font y TextRenderer.

Uso de fuentes incorporadas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se especifica una fuente específica para un objeto TextField en la aplicación, Flash Player o AIR buscarán una fuente de dispositivo (una fuente que resida en el equipo del usuario) que tenga el mismo nombre. Si no encuentra esa fuente en el sistema o si el usuario tiene una versión algo distinta de una fuente con ese nombre, el aspecto del texto visualizado puede ser muy distinto del esperado. De forma predeterminada, el texto aparece en fuente Times Roman.

Para asegurarse de que el usuario ve la fuente correcta, se puede incorporar dicha fuente al archivo SWF de la aplicación. Las fuentes incorporadas ofrecen varias ventajas:

- Los caracteres de las fuentes incorporadas se suavizan, por lo que sus bordes parecen más lisos, especialmente en textos grandes.
- Se puede girar textos con fuentes incorporadas.
- El texto de una fuente incorporada se puede convertir en transparente o semitransparente.
- Se puede utilizar el estilo CSS de `kerning` (ajuste entre caracteres) con fuentes incorporadas.

La mayor limitación del uso de fuentes incorporadas es que aumentan el tamaño del archivo o el tiempo de descarga de la aplicación.

El método preciso de incorporar un archivo de fuente en el archivo SWF de la aplicación varía de un entorno de desarrollo a otro.

Tras incorporar una fuente, es necesario comprobar que un objeto TextField utiliza la fuente incorporada correcta:

- Establezca el valor de la propiedad `embedFonts` del objeto TextField en `true`.
- Cree un objeto TextFormat, establezca su propiedad `fontFamily` en el nombre de la fuente incorporada y aplique el objeto TextFormat al objeto TextField. Al especificar una fuente incorporada, la propiedad `fontFamily` solo debe contener un único nombre; no se puede utilizar una lista de nombres de fuentes delimitados por comas.
- Si se utilizan estilos CSS para definir fuentes de objetos TextField o componentes, hay que establecer la propiedad CSS `font-family` en el nombre de la fuente incorporada. Si se desea especificar una fuente incorporada, la propiedad `font-family`debe contener un solo nombre, no una lista de nombres.

Incorporación de una fuente en Flash

La herramienta de edición de Flash permite incorporar prácticamente cualquier fuente que esté instalada en el sistema, incluidas las fuentes TrueType y las fuentes Postscript de tipo 1.

Existen varias formas de incorporar fuentes en una aplicación, entre las que se incluyen:

- Establecer las propiedades de fuente y estilo de un objeto TextField en el escenario y hacer clic en la casilla de verificación Fuentes incorporadas.

- Crear un sí-mbolo de fuente y hacer referencia al mismo.
- Crear y utilizar una biblioteca compartida en tiempo de ejecución que contenga sí-mbolos de fuente incorporados.

Para obtener detalles sobre cómo incorporar fuentes en aplicaciones, consulte "Incorporación de fuentes para campos de texto dinámico o de entrada" en *Uso de Flash*.

Incorporación de una fuente en Flex

Existen varias formas de incorporar fuentes en una aplicación Flex, entre las que se incluyen:

- Utilizar la etiqueta de metadatos `[Embed]` en un script
- Uso de la declaración de estilos `@font-face`
- Establezca una clase para la fuente y utilice la etiqueta `[Embed]` para incorporarla.

En una aplicación Flex solo se pueden incorporar directamente fuentes TrueType. Las fuentes que están en otro formato, como las PostScript de tipo 1, se pueden incorporar primero en un archivo SWF de Flash Professional mediante la herramienta de edición de Flash y, posteriormente, ese archivo SWF se puede usar en la aplicación Flex. Para obtener más información sobre la utilización de fuentes en Flex incorporadas desde archivos SWF, consulte la *Guía del desarrollador de Flex 4*.

Más temas de ayuda

Incorporación de fuentes para obtener un aspecto de texto uniforme

Peter deHaan: Embedding fonts (Incorporación de fuentes; en inglés)

Divillysausages.com: AS3 Font embedding masterclass

Control de la nitidez, el grosor y el suavizado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De forma predeterminada, Flash Player o AIR determinan la configuración de los controles de la visualización del texto, como la nitidez, el grosor y el suavizado, cuando el texto cambia de tamaño, de color o se muestra con distintos fondos. En algunos casos, como cuando se dispone de un texto muy pequeño o muy grande, o un texto en diversos fondos únicos, es posible que se desee controlar esta configuración. Es posible reemplazar la configuración de Flash Player o AIR mediante la clase `flash.text.TextRenderer` y sus clases asociadas, como CSMSettings. Estas clases ofrecen un control preciso de la calidad de representación del texto incorporado. Para obtener más información sobre las fuentes incorporadas, consulte "Uso de fuentes incorporadas" en la página 403.

Nota: *la propiedad `flash.text.TextField`.antiAliasType debe contar con el valor `AntiAliasType.ADVANCED` para que se pueda establecer la nitidez, el grosor o la propiedad gridFitType o para utilizar el método `TextRenderer.setAdvancedAntiAliasingTable()`.*

En el ejemplo siguiente se aplican propiedades de modulación de trazo continua y formato personalizados a texto visualizado con una fuente incorporada denominada `myFont`. Cuando el usuario hace clic en el texto mostrado, Flash Player o Adobe AIR aplican la configuración personalizada:

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
TextColorType.DARK_COLOR, myAliasTable);
}
```

Trabajo con texto estático

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El texto estático solo se crea en Flash Professional. No se puede crear texto estático mediante programación con ActionScript. El texto estático resulta útil si el texto es breve y no va a cambiar (a diferencia del texto dinámico). Se puede considerar que el texto estático es un tipo de elemento gráfico, como un círculo o un cuadrado, dibujado en el escenario de Flash Professional. Aunque el texto estático presenta más limitaciones que el texto dinámico, ActionScript 3.0 permite leer los valores de propiedades de texto estático mediante la clase flash.text.StaticText. También se puede utilizar la clase TextSnapshot para leer valores del texto estático.

Acceso a campos de texto estático mediante la clase StaticText

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Generalmente, se usa la clase flash.text.StaticText en el panel Acciones de la herramienta de edición de Flash para interactuar con una instancia de texto estático colocada en el escenario. También se puede trabajar en archivos de ActionScript que interactúen con un archivo SWF que contenga texto estático. En cualquier caso, no se puede crear una instancia de texto estático mediante programación. El texto estático se crea en Flash Professional.

Para crear una referencia a un campo de texto estático, se pueden repetir los elementos de la lista de visualización y asignar una variable. Por ejemplo:

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

Una vez que se tiene una referencia a un campo de texto estático, se pueden usar las propiedades de ese campo en ActionScript 3.0. El código siguiente se adjunta a un fotograma de la línea de tiempo y presupone que una variable denominada `myFieldLabel` está asignada a una referencia de texto estático. Un campo de texto dinámico denominado `myField` se coloca con respecto a los valores `x` e `y` de `myFieldLabel` y vuelve a mostrar el valor de `myFieldLabel`.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

Uso de la clase TextSnapshot

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se desea trabajar mediante programación con una instancia de texto estático existente, se puede utilizar la clase flash.text.TextSnapshot para utilizar la propiedad `textSnapshot` de flash.display.DisplayObjectContainer. En otras palabras, se crea una instancia de TextSnapshot a partir de la propiedad `DisplayObjectContainer.textSnapshot`. Después se pueden aplicar métodos a esa instancia para recuperar valores o seleccionar partes del texto estático.

Por ejemplo, coloque en el escenario un campo de texto estático que contenga el texto "TextSnapshot Example". Añada el siguiente código ActionScript al fotograma 1 de la lí-nea de tiempo:

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

La clase TextSnapshot resulta útil para obtener el texto de campos de texto estático de un archivo SWF cargado, si se desea utilizar el texto como un valor en otra parte de una aplicación.

Ejemplo de TextField: Formato de texto con estilo periodístico

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El ejemplo News Layout aplica formato a texto para que parezca una noticia de un periódico impreso. El texto de entrada puede contener un titular, un subtí-tulo y el cuerpo de la noticia. Dadas la anchura y la altura de visualización, el ejemplo News Layout aplica formato al titular y al subtítulo para que ocupen toda la anchura del área de visualización. El texto de la noticia se distribuye en dos o más columnas.

En este ejemplo se ilustran las siguientes técnicas de programación en ActionScript:

- Ampliación de la clase TextField
- Cargar y aplicar un archivo CSS externo
- Convertir estilos de CSS en objetos TextFormat
- Utilizar la clase TextLineMetrics para obtener información sobre el tamaño de la visualización de texto

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación News Layout se encuentran en la carpeta Samples/NewsLayout. La aplicación consta de los siguientes archivos:

Archivo	Descripción
NewsLayout.mxml o NewsLayout.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
com/example/programmingas3/newslayout/StoryLayoutComponent.as	Una clase UIComponent de Flex que coloca la instancia de StoryLayout.
com/example/programmingas3/newslayout/StoryLayout.as	La clase principal de ActionScript que dispone todos los componentes de una noticia para mostrarla.
com/example/programmingas3/newslayout/FormattedTextField.as	Una subclase de la clase TextField que administra su propio objeto TextFormat.
com/example/programmingas3/newslayout/HeadlineTextField.as	Una subclase de la clase FormattedTextField que ajusta el tamaño de las fuentes a una anchura deseada.
com/example/programmingas3/newslayout/MultiColumnTextField.as	Una clase de ActionScript que divide el texto en dos o más columnas.
story.css	Un archivo CSS que define estilos de texto para el diseño.

Lectura del archivo CSS externo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación News Layout empieza por leer en un archivo XML local el texto de la noticia. A continuación, lee el archivo CSS externo que proporciona la información de formato para el titular, el subtí-tulo y el texto principal.

El archivo CSS define tres estilos, un estilo de párrafo estándar para la noticia y los estilos h1 y h2 para el titular y el subtítulo respectivamente.

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

La técnica usada para leer el archivo CSS externo es la misma que la descrita en "Carga de un archivo CSS externo" en la página 400. Cuando el archivo CSS se ha cargado, la aplicación ejecuta el método `onCSSFileLoaded()`, tal y como se muestra a continuación.

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

El método `onCSSFileLoaded()` crea un objeto StyleSheet y hace que analice los datos CSS de entrada. El texto principal de la historia se muestra en un objeto MultiColumnTextField que puede utilizar un objeto StyleSheet directamente. No obstante, los campos de titular usan la clase HeadlineTextField, que utiliza un objeto TextFormat para su formato.

El método `onCSSFileLoaded()` llama dos veces a `getTextStyle()` para convertir una declaración de estilos CSS en un objeto TextFormat para utilizarlo con cada uno de los dos objetos HeadlineTextField.

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

Los nombres de propiedad y el significado de los valores de propiedad difieren entre las declaraciones de estilos CSS y los objetos TextFormat. El método `getTextStyle()` traduce los valores de propiedades CSS en los valores que espera el objeto TextFormat.

Disposición de los elementos de la noticia en la página

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase StoryLayout aplica formato a los campos de titular, subtí-tulo y texto principal y los dispone como en un periódico. El método `displayText()` crea y sitúa inicialmente los distintos campos.

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

Cada uno de los campos se coloca bajo el anterior estableciendo su propiedad `y` en un valor igual al de la propiedad `y` del campo anterior más su altura. Este cálculo dinámico de la posición es necesario, ya que los objetos HeadlineTextField y MultiColumnTextField pueden modificar su altura para ajustarla a su contenido.

Modificación del tamaño de fuente para ajustar el tamaño de un campo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Dados una anchura en píxeles y un número máximo de líneas para mostrar, HeadlineTextField modifica el tamaño de fuente para ajustar el texto al campo. Si el texto es corto, el tamaño de fuente es muy grande, y se creará un titular de tipo tabloide. Si el texto es largo, tamaño de fuente es más pequeño.

El método `HeadlineTextField.fitText()` que se muestra a continuación ajusta el tamaño de fuente:

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false,
targetWidth:Number = -1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
```

```
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

El método `HeadlineTextField.fitText()` utiliza una técnica recursiva sencilla para ajustar el tamaño de la fuente. En primer lugar calcula un número de píxeles por carácter en el texto y con ese dato calcula un tamaño de punto inicial. A continuación cambia el tamaño de la fuente y comprueba si el texto se ha ajustado para crear un número de líneas de texto superior al máximo. Si hay demasiadas lí-neas, llama al método `shrinkText()` para reducir el tamaño de fuente y lo intenta de nuevo. Si no hay demasiadas lí-neas, llama al método `growText()` para aumentar el tamaño de fuente y lo intenta de nuevo. El proceso se detiene en el punto en el que, al aumentar un punto el tamaño de fuente, se crearán demasiadas líneas.

División del texto en varias columnas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase MultiColumnTextField divide el texto en varios objetos TextField que se disponen como las columnas de un periódico.

El constructor `MultiColumnTextField()` crea primero un conjunto de objetos TextField, uno por cada columna, tal y como se muestra a continuación:

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

Todos los objetos TextField se añaden al conjunto y a la lista de visualización con el método `addChild()`.

Siempre que cambien las propiedades `text` o `styleSheet` del objeto StoryLayout, se llama al método `layoutColumns()` para volver a mostrar el texto. El método `layoutColumns()` llama al método `getOptimalHeight()` method, para calcular la altura en pí-xeles necesaria para ajustar todo el texto en la anchura de diseño especificada.

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

En primer lugar, el método `getOptimalHeight()` calcula la anchura de cada columna. Posteriormente se establece la anchura y la propiedad `htmlText` del primer objeto TextField en el conjunto. El método `getOptimalHeight()` utiliza ese primer objeto TextField para descubrir el número total de líneas ajustadas en el texto y, a partir de ese número, determina cuántas líneas debe haber en cada columna. Posteriormente llama al método `TextField.getLineMetrics()` para recuperar un objeto TextLineMetrics que contiene detalles sobre el tamaño del texto en la primera lí-nea. La propiedad `TextLineMetrics.height` representa la altura total de una -línea de texto, en píxeles, incluidos los valores ascendente, descendente y de interlineado. La altura óptima para el objeto MultiColumnTextField es la altura de línea multiplicada por el número de líneas por columna, más 4 para tener en cuenta el borde de dos píxeles que hay por encima y por debajo de un objeto TextField.

A continuación se muestra el código del método `layoutColumns()`:

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
```

```
field = this.fieldArray[i] as TextField;

field.height = this.preferredHeight;
field.text = remainder;

field.setTextFormat(this.format);

var lineLen:int;
if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
        }
    }

field.x = i * (colWidth + gutter);
field.y = 0;

remainder = "";
fieldText = "";

var linesRemaining:int = field.numLines;
var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

for (var j:int = 0; j < linesRemaining; j++)
{
    if (j < linesVisible)
    {
        fieldText += field.getLineText(j);
    }
    else
    {
        remainder +=field.getLineText(j);
    }
}

field.text = fieldText;

field.setTextFormat(this.format);

if (indent > 0 && !lastLineEndedPara)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
    }
}

var lastLine:String = field.getLineText(field.numLines - 1);
var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);
```

```
        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

Una vez establecida la propiedad `preferredHeight` al llamar al método `getOptimalHeight()`, el método `layoutColumns()` recorre los objetos TextField y establece la altura de cada uno de ellos en el valor de `preferredHeight`. Posteriormente el método `layoutColumns()` distribuye a cada campo las líneas de texto suficientes para que no se produzca desplazamiento en ningún campo y el texto de cada campo empiece donde acabó el texto del campo anterior. Si el estilo de alineación del texto está establecido en "justify" (justificado), se llama al método `justifyLastLine()` para justificar la última lí-nea del texto de un campo. De lo contrario, esa última lí-nea se tratará como una línea de final de párrafo y no se justificará.

Capítulo 22: Uso de Flash Text Engine

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Adobe® Flash® Text Engine (FTE), disponible comenzando con Flash Player 10 y Adobe® AIR™1.5, ofrece una compatibilidad de bajo nivel para realizar un sofisticado control de las medidas y el formato del texto, además de admitir texto bidireccional. Ofrece una compatibilidad de idiomas y un flujo de texto mejorados. Aunque se puede usar para crear y administrar elementos de texto sencillos, FTE está diseñado principalmente como base sobre la que los desarrolladores puedan crear componentes de gestión del texto. Como tal, para Flash Text Engine se presupone un nivel más avanzado de conocimientos de programación. Para ver elementos de texto sencillos, consulte "Uso de la clase TextField" en la página 391.

Text Layout Framework, que incluye un componente de gestión de texto basado en FTE, ofrece una forma más segura de utilizar sus funciones avanzadas. Text Layout Framework es una biblioteca ampliable incorporada por completo en ActionScript 3.0. Es posible utilizar el componente TLF existente o la arquitectura para crear un componente de texto propio. Para obtener más información, consulte "Uso de Text Layout Framework" en la página 446.

Más temas de ayuda

Paquete flash.text.engine

Creación y visualización de texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Las clases que conforman Flash Text Engine permiten crear texto, aplicarle formato y controlarlo. Las clases siguientes son los componentes esenciales para crear y mostrar texto con Flash Text Engine:

- TextElement/GraphicElement/GroupElement: incluyen el contenido de una instancia de TextBlock.
- ElementFormat: especifica los atributos de formato para el contenido de una instancia de TextBlock.
- TextBlock - componente fundamental para crear un párrafo de texto.
- TextLine: una línea de texto creada a partir de TextBlock

Para mostrar el texto, cree un objeto TextElement a partir de una cadena y usando un objeto ElementFormat para especificar las características de formato. Asigne TextElement a la propiedad `content` de un objeto TextBlock. Las líneas de texto que se van a mostrar se crean llamando al método `TextBlock.createTextLine()`. El método `createTextLine()` devuelve un objeto TextLine que contiene parte de la cadena que se ajustará en la anchura especificada. Llame al método varias veces hasta que a toda la cadena se le haya aplicado el formato en líneas. Cuando no haya más líneas que crear, a la propiedad textLineCreationResult del objeto se le asigna el valor: `TextLineCreationResult.COMPLETE`. Para mostrar las líneas, añádalas a la lista de visualización (con los valores de posición `x` e `y` apropiados).

Por ejemplo, el código siguiente utiliza estas clases de FTE para mostrar el texto "Hello World! This is Flash Text Engine!", usando el formato y los valores de fuente predeterminados. En este sencillo ejemplo, únicamente se crea una sola línea de texto.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

Los parámetros para `createTextLine()` especifican la línea a partir de la que comienza la nueva lí-nea y su anchura en píxeles. La línea a partir de la que comienza la nueva línea suele ser la anterior pero, en caso de que sea la primera línea, es `null`.

Cómo añadir objetos GraphicElement y GroupElement

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Para mostrar una imagen o un elemento gráfico, se puede asignar un objeto GraphicElement a un objeto TextBlock. Únicamente se debe crear una instancia de la clase GraphicElement a partir de un gráfico o una imagen y asignar la instancia a la propiedad `TextBlock.content`. Cree la línea de texto llamando a `TextBlock.createTextline()` tal y como lo haría normalmente. El ejemplo siguiente crea dos líneas de texto, una con un objeto GraphicElement y otra con un objeto TextElement.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

Se puede crear un objeto GroupElement para crear un grupo de objetos TextElement, GraphicElement y otros objetos de tipo GroupElement. GroupElement se puede asignar a la propiedad `content` de un objeto TextBlock. El parámetro pasado al constructor `GroupElement()` es un Vector, que apunta a los elementos de texto, gráficos y grupo que conforman el grupo. El ejemplo siguiente agrupa dos elementos gráficos y un elemento de texto y los asigna como una unidad a un bloque de texto.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1,
triangle1.width, triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2,
triangle2.width, triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

Reemplazo de texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Se puede reemplazar texto en una instancia de TextBlock llamando a `TextElement.replaceText()` para reemplazar el texto del elemento TextElement que asignó a la propiedad `TextBlock.content`.

En el ejemplo siguiente se utiliza `repaceText()` para insertar texto al principio de la línea, añadir texto al final de la lí-nea y reemplazar texto en medio de la línea.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

El método `replaceText()` reemplaza el texto especificado por los parámetros `beginIndex` y `endIndex` con el texto que especifica el parámetro `newText`. Si los valores de los parámetros `beginIndex` y `endIndex` son los mismos, `replaceText()` inserta el texto especificado en esa ubicación. De lo contrario, reemplaza los caracteres especificados mediante `beginIndex` y `endIndex` con el nuevo texto.

Gestión de eventos en FTE

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Se pueden añadir detectores de eventos a una instancia de TextLine igual que sucede con otros objetos de visualización. Por ejemplo, se puede detectar el momento en que un usuario desplaza el ratón sobre una lí-nea de texto o hace clic en la línea. En el siguiente ejemplo se detectan ambos eventos. Cuando el ratón se desplaza sobre la línea, el cursor cambia a un cursor de botón y cuando se hace clic en la línea, este cambia de color.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
```

```
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

Reflejo de eventos

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

También se pueden reflejar los eventos de un bloque de texto, o de una parte de un bloque de texto, en un distribuidor de eventos. En primer lugar, se crea una instancia de EventDispatcher y se asigna a la propiedad `eventMirror` de una instancia de TextElement. Si el bloque de texto consta de un solo elemento de texto, el motor de texto refleja los eventos de todo el bloque. Si el bloque de texto está formado por varios elementos de texto, el motor de texto solo refleja los eventos de las instancias de TextElement que tienen establecida la propiedad `eventMirror`. El texto del siguiente ejemplo consta de tres elementos: la palabra "Click", la palabra "here" y la cadena "to see me in italic". El ejemplo asigna un distribuidor de eventos al segundo elemento de texto, la palabra "here", y añade un detector de eventos, el método `clickHandler()`. El método `clickHandler()` aplica al texto formato de cursiva. Asimismo, reemplaza el contenido del tercer elemento de texto para que se lea, "Click here to see me in normal font!".

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }
```

```
        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

Las funciones `mouseOverHandler()` y `mouseOutHandler()` cambian el cursor a un cursor de botón cuando se encuentra sobre la palabra "here" y le devuelven la forma de flecha cuando no lo está.

Formato de texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

A `TextBlock` es la base para crear líneas de texto. El contenido de `TextBlock` se asigna a través del objeto `TextElement`. Un objeto `ElementFormat` controla el formato del texto. La clase ElementFormat define propiedades como la alineación de la línea de base, el ajuste y espaciado entre caracteres, la rotación de texto y el tamaño, el color y el formato de mayúsculas de la fuente. También incluye un elemento `FontDescription`, que se analiza detalladamente en "Trabajo con fuentes" en la página 429.

Uso del objeto ElementFormat

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El constructor del objeto `ElementFormat` adopta cualquiera de una larga lista de parámetros opcionales, incluido el parámetro `FontDescription`. Estas propiedades también se pueden establecer fuera del constructor. El ejemplo siguiente indica la relación de los distintos objetos en la definición y visualización de una lí-nea de texto sencillo:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Color de fuente y transparencia (alfa)

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `color` del objeto `ElementFormat` define el color de la fuente. El valor es un entero que representa los componentes RGB del color; por ejemplo, 0xFF0000 para el rojo y 0x00FF00 para el verde. El valor predeterminado es el negro (0x000000).

La propiedad `alpha` define el valor de transparencia alfa de un elemento (`TextElement` y `GraphicElement`). Los valores se encuentran entre 0 (totalmente transparente) y 1 (completamente opaco, que es el valor predeterminado). Los elementos con un valor `alpha` de 0 son invisibles, pero siguen estando activos. Este valor se multiplica por los valores de alpha heredados, haciendo de este modo que el elemento sea más transparente.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

Alineación y desplazamiento de la lí-nea de base

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La fuente y el tamaño del texto más grande de una lí-nea determinan su línea de base dominante. Estos valores se pueden reemplazar definiendo las propiedades `TextBlock.baselineFontDescription` y `TextBlock.baselineFontSize`. La línea de base dominante se puede alinear con una de las diversas líneas de base del texto. Entre ellas se encuentran la lí-nea ascendente y la línea descendente, o la línea superior, central o inferior de los pictogramas.

A. *Ascendente* ***B.*** *Línea de base* ***C.*** *Descendente* ***D.*** *Altura x*

En el objeto `ElementFormat` las características de línea de base y alineación se definen mediante tres propiedades. La propiedad `alignmentBaseline` establece la línea de base principal de un objeto `TextElement` o `GraphicElement`. Esta lí-nea de base es la línea "de ajuste" para el elemento y respecto a su posición se alinea la línea de base dominante de todo el texto.

La propiedad `dominantBaseline` especifica cuál de las distintas lí-neas de base del elemento se debe usar, lo que determina la posición vertical del elemento en la línea. El valor predeterminado es `TextBaseline.ROMAN`, pero también se pueden establecer como dominantes las lí-neas de base `IDEOGRAPHIC_TOP` o `IDEOGRAPHIC_BOTTOM`.

La propiedad `baselineShift` desplaza la línea de base un número determinado de píxeles a lo largo del eje y. En el caso de texto normal (no rotado), un valor positivo desplaza la línea de base hacia abajo, mientras que un valor negativo lo hace hacia arriba.

Formato tipográfico de mayúsculas y minúsculas

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `TypographicCase` de `ElementFormat` especifica el uso de mayúsculas, minúsculas o versalitas del texto.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

Rotación del texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Los bloques de texto o los pictogramas incluidos en un segmento de texto se pueden rotar en incrementos de 90 grados°. La clase TextRotation define las constantes siguientes para establecer la rotación de los bloques de texto y los pictogramas:

Constante	Valor	Descripción
AUTO	"auto"	Especifica una rotación de 90 grados en sentido contrario a las agujas del reloj. Se utiliza normalmente con textos asiáticos, para rotar únicamente los pictogramas que necesitan rotarse.
ROTATE_0	"rotate_0"	No especifica ninguna rotación.
ROTATE_180	"rotate_180"	Especifica una rotación de 180 grados.
ROTATE_270	"rotate_270"	Especifica una rotación de 270 grados.
ROTATE_90	"rotate_90"	Especifica una rotación de 90 grados en sentido horario.

Para rotar las lí-neas de texto de un bloque de texto, establezca la propiedad `TextBlock.lineRotation` antes de llamar al método `TextBlock.createTextLine()` para crear la línea de texto.

Para rotar los pictogramas incluidos en un bloque de texto o un segmento, establezca la propiedad `ElementFormat.textRotation` en el número de grados que desea girar los pictogramas. Un pictograma es la forma que constituye un carácter o una parte de un carácter que consta de varios pictogramas. Por ejemplo, la letra a y el punto de la i son pictogramas.

Rotar pictogramas es relevante en algunos idiomas asiáticos en los que puede ser necesario rotar las líneas hasta el sentido vertical, pero no los caracteres incluidos en las líneas. Para obtener información sobre la rotación de texto en idiomas asiáticos, consulte "Justificación de texto asiático" en la página 433.

A continuación se muestra un ejemplo de rotación tanto de un bloque de texto como de los pictogramas que incluye, tal y como se llevará a cabo con textos asiáticos. El ejemplo también utiliza una fuente japonesa:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Bloqueo y clonación de ElementFormat

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Si un objeto `ElementFormat` se asigna a cualquier tipo de elemento `ContentElement`, su propiedad `locked` se establece automáticamente en el valor `true`. Si se intenta modificar un objeto `ElementFormat` bloqueado, se genera un error de tipo `IllegalOperationError`. La práctica más recomendable consiste en definir completamente un objeto así antes de asignarlo a una instancia de `TextElement`.

Si desea modificar una instancia existente de `ElementFormat`, compruebe antes su propiedad `locked`. Si es `true`, utilice el método `clone()` para crear una copia no bloqueada del objeto. Las propiedades de este objeto desbloqueado se pueden cambiar y, posteriormente, asignarlo a la instancia de `TextElement`. Las nuevas lí-neas que se creen a partir de él tendrán el nuevo formato. Las lí-neas anteriores creadas a partir de este mismo objeto y con el formato antiguo no se modifican.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Trabajo con fuentes

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El objeto `FontDescription` se utiliza junto con `ElementFormat` para identificar una fuente y definir algunas de sus características. Entre estas características se encuentran el nombre de la fuente, grosor, postura, representación y el modo de localizar la fuente (en el dispositivo en lugar de incorporada).

Nota: *FTE no admite fuentes de Tipo 1 o fuentes de mapa de bits, como las de Tipo 3, ATC, sfnt-wrapped CID o Naked CID.*

Definición de características de fuentes (objeto FontDescription)

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `fontName` del objeto `FontDescription` puede ser un único nombre o una lista de nombres separadas por comas. Por ejemplo, en una lista como "Arial, Helvetica, _sans", el motor de texto busca primero "Arial", después "Helvetica" y, por último, "_sans", si no encuentran alguna de las dos primeras fuentes. El conjunto de nombres de fuente incluye tres nombres genéricos de fuentes de dispositivo: “_sans”, “_serif” y “_typewriter”. Dependiendo del sistema de reproducción, se corresponden con fuentes de dispositivo específicas. Se recomienda especificar nombres predeterminados como estos en todas las descripciones de fuentes que utilizan fuentes de dispositivo. Si no se especifica `fontName`, se utiliza “_serif” de forma predeterminada.

La propiedad `fontPosture` se puede establecer en su valor predeterminado (`FontPosture.NORMAL`) o en cursiva (`FontPosture.ITALIC`). La propiedad `fontWeight` se puede establecer en su valor predeterminado (`FontWeight.NORMAL`) o en negrita (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

Fuentes incorporadas frente a fuentes de dispositivo

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `fontLookup` del objeto `FontDescription` especifica si el motor de texto debe buscar una fuente de dispositivo o una fuente incorporada para procesar el texto. Si se especifica una fuente de dispositivo (`FontLookup.DEVICE`), el tiempo de ejecución buscará la fuente en el sistema de reproducción. La especificación de una fuente incorporada (`FontLookup.EMBEDDED_CFF`) hace que el tiempo de ejecución busque una fuente incorporada con el nombre especificado en el archivo SWF. Con esta configuración solo funcionan las fuentes CFF (de formato de fuente compacto) incorporadas. Si no se encuentra la fuente especificada, se utiliza una fuente de dispositivo alternativa.

Las fuentes de dispositivo suponen un tamaño de archivo SW menor. Las fuentes incorporadas ofrecen una mayor fidelidad entre plataformas.

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

Modo de representación e interpolación

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La representación de CFF (Formato de fuente compacto) se encuentra disponible a partir de Flash Player 10 y Adobe AIR 1.5. Este tipo de representación de fuentes hace que el texto sea más legible y permite obtener una visualización de mayor calidad de las fuentes a tamaños reducidos. Esta configuración solo se aplica a las fuentes incorporadas. `FontDescription` toma como predeterminada esta configuración (`RenderingMode.CFF`) para la propiedad `renderingMode` . Esta propiedad se puede establecer en `RenderingMode.NORMAL` para que coincida con el tipo de representación utilizado por Flash Player 7 o versiones anteriores.

Cuando está seleccionada la representación CFF, una segunda propiedad, `cffHinting`, controla cómo se ajustan los trazos horizontales de una fuente a la cuadrícula de subpí-xeles. El valor predeterminado, `CFFHinting.HORIZONTAL_STEM`, utiliza la interpolación CFF. Al establecer esta propiedad en `CFFHinting.NONE` se elimina la interpolación, lo que resulta adecuado para animaciones o para tamaños de fuente grandes.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

Bloqueo y clonación de FontDescription

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Si un objeto `FontDescription` se asigna a `ElementFormat`, su propiedad `locked` se establece automáticamente en `true`. Si se intenta modificar un objeto `FontDescription` bloqueado, se generar un error de tipo `IllegalOperationError`. Se recomienda definir completamente un objeto así antes de asignarlo a una instancia de `ElementFormat`.

Si desea modificar un elemento `FontDescription` existente, compruebe antes su propiedad `locked`. Si es `true`, utilice el método `clone()` para crear una copia no bloqueada del objeto. Las propiedades de este objeto desbloqueado se pueden cambiar y, posteriormente, asignarlo a `ElementFormat`. Las nuevas líneas que se creen a partir de este `TextElement` tendrán el nuevo formato. Las líneas anteriores creadas a partir de este mismo objeto no cambian.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Control del texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

FTE ofrece un nuevo conjunto de controles de formato del texto para controlar la justificación y el espaciado (ajuste y espaciado entre caracteres). También existen propiedades para controlar si hay lí-neas rotas y para establecer tabulaciones dentro de las líneas.

Justificación del texto

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Si se justifica el texto, se proporciona la misma longitud a todas las lí-neas de un párrafo ajustando el espaciado entre las palabras y, a veces, entre las letras. El efecto consiste en alinear el texto a ambos lados, mientras que se varía el espaciado entre las palabras y las letras. Las columnas de texto de los periódicos y las revistas suelen estar justificadas.

La propiedad `lineJustfication` de la clase SpaceJustifier permite controlar la justificación de las líneas de un bloque de texto. La clase LineJustification define constantes que se pueden usar para especificar una opción de justificación: `ALL_BUT_LAST` justifica todas las lí-neas de texto excepto la última; `ALL_INCLUDING_LAST` justifica todo el texto, incluyendo la última línea; `UNJUSTIFIED`, que es el valor predeterminado, deja el texto sin justificar.

Para justificar texto, establezca la propiedad `lineJustification` en una instancia de la clase SpaceJustifier y asigne dicha instancia a la propiedad `textJustifier` de una instancia de TextBlock. En el ejemplo siguiente se crea un párrafo en el que todas las líneas de texto están justificadas, excepto la última.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new
SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

Para modificar el espaciado entre las letras y entre las palabras, la propiedad `SpaceJustifier.letterspacing` se establece en`true`. Activando el espaciado entre las letras se puede reducir la aparición de antiestéticos espacios entre las palabras, lo que a veces ocurre con la justificación simple.

Justificación de texto asiático

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La justificación de textos en idiomas de Extremo Oriente implica otras consideraciones. Se puede escribir de arriba a abajo y algunos caracteres, conocidos como kinsoku, no pueden aparecer al principio o al final de una línea. La clase JustificationStyle define las siguientes constantes, que especifican las opciones para gestionar estos caracteres. `PRIORITIZE_LEAST_ADJUSTMENT` basa la justificación en expandir o comprimir la lí-nea, dependiendo de qué opción ofrece el resultado más deseable. `PUSH_IN_KINSOKU` basa la justificación en la compresión kinsoku al final de la línea o su aumento si no hay kinsoku o si el espacio es insuficiente.

`PUSH_OUT_ONLY` basa la justificación en expandir la lí-nea. Para crear un bloque de texto asiático vertical, establezca la propiedad `TextBlock.lineRotation` en `TextRotation.ROTATE_90` y la propiedad `ElementFormat.textRotation` en `TextRotation.AUTO`, que es el valor predeterminado. Si la propiedad `textRotation` se establece en `AUTO`, los pictogramas del texto se mantienen en vertical en lugar de girarse sobre un lado cuando se rota la línea. El valor `AUTO` rota 90 grados en sentido contrario al de las agujas del reloj solo los pictogramas de anchura completa y los anchos, según lo determinen las propiedades Unicode del pictograma. El ejemplo siguiente muestra un bloque vertical de texto en japonés y lo justifica utilizando la opción `PUSH_IN_KINSOKU`.

```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
```

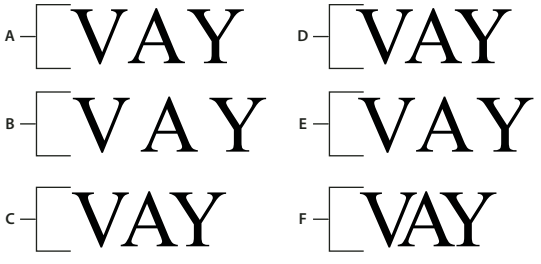
```
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

Ajuste y espaciado entre caracteres

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El ajuste y espaciado entre caracteres afectan a la distancia existente entre pares adyacentes de caracteres de un bloque de texto. Con el ajuste entre caracteres se controla cómo se "ajustan" entre sí- los pares de caracteres como, por ejemplo, "WA" o "Va". El ajuste entre caracteres se establece en el objeto `ElementFormat`. Se encuentra activado de forma predeterminada (`Kerning.ON`) y se puede establecer en OFF o en AUTO, en cuyo caso el espaciado manual solo se aplica entre los caracteres si ninguno de ellos es Kanji, Hiragana o Katakana.

Con el espaciado entre caracteres se añade o retira un número determinado de pí-xeles entre todos los caracteres de un bloque de texto y también se establece en el objeto `ElementFormat`. Funciona tanto con fuentes de dispositivo como incorporadas. FTE admite dos propiedades de espaciado entre caracteres, `trackingLeft`, que añade o retira pí-xeles por el lado izquierdo de un carácter, y `trackingRight`, que los añade o retira por el lado derecho. Cuando se utiliza el ajuste entre caracteres, el valor de espaciado se añade o se resta de los valores de ajuste para cada par de caracteres.



A. *Kerning.OFF* ***B.*** *TrackingRight=5, Kerning.OFF* ***C.*** *TrackingRight=-5, Kerning.OFF* ***D.*** *Kerning.ON* ***E.*** *TrackingRight=-5, Kerning.ON* ***F.*** *TrackingRight=-5, Kerning.ON*

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

Saltos de línea para texto ajustado

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La propiedad `breakOpportunity` del objeto `ElementFormat` determina qué caracteres se pueden usar para realizar el salto de lí-nea cuando el texto ajustado ocupa varias lí-neas. La configuración predeterminada, `BreakOpportunity.AUTO`, utiliza propiedades estándar de Unicode como, por ejemplo, dividir las lí-neas entre palabras o cuando hay guiones. Con el uso de `BreakOpportunity.ALL` cualquier carácter se puede tratar como oportunidad de salto de línea, lo que resulta útil para crear efectos como texto distribuido a lo largo de un trazado.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Tabulaciones

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Para establecer tabulaciones en un bloque de texto, dichas tabulaciones se definen creando instancias de la clase TabStop. Los parámetros pasados al constructor `TabStop()` especifican cómo se alinea el texto con respecto a la tabulación. Estos parámetros especifican la posición de la tabulación y, para la alineación decimal, el valor respecto al que realizarla, expresado como una cadena. Normalmente, este valor es un punto decimal, pero también podr-á ser una coma, un símbolo de dólar o el sí-mbolo del yen o el euro, por ejemplo. La siguiente línea de código crea una tabulación denominada tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

Una vez creadas las tabulaciones para un bloque de texto, se asignan a la propiedad `tabStops` de una instancia de TextBlock. Debido a que la propiedad `tabStops` requiere un vector, primero debe crearse un vector y después añadirle las tabulaciones. El vector permite asignarle un juego de tabulaciones al bloque de texto. En el siguiente ejemplo se crea una instancia de `Vector<TabStop>` y se añade un conjunto de objetos TabStop. A continuación se asignan las tabulaciones a la propiedad `tabStops` de una instancia de TextBlock.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Para obtener más información sobre los vectores, consulte “Trabajo con conjuntos” en la página 25.

El ejemplo siguiente muestra los efectos que se obtienen con las distintas opciones de alineación de TabStop.

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDescription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Ejemplo de Flash Text Engine: News Layout

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Este ejemplo de programación muestra el uso de Flash Text Engine con el diseño una sencilla página de periódico. La página incluye un titular grande, un subtí-tulo y una sección de cuerpo de varias columnas.

En primer lugar, cree un nuevo archivo FLA y añada el siguiente código al fotograma #2 de la capa predeterminada:

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as es el script del controlador de este ejemplo. Establece el contenido, lee la información de estilo de una hoja de estilos externa y asigna estos estilos a objetos ElementFormat. Posteriormente crea el titular, el subtí-tulo y los elementos de texto de varias columnas.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;
```

```
        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the
Flash Text Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should
consider him a friend to mankind; and as our acquaintance commenced on that ground, it would
have been more agreeable to me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament
against the French Revolution and the National Assembly, I was in Paris, and had written to him
but a short time before to inform him how prosperously matters were going on. Soon after this I
saw his advertisement of the Pamphlet he intended to publish: As the attack was to be made in a
language but little studied, and less understood in France, and as everything suffers by
translation, I promised some of the friends of the Revolution in that country that whenever Mr.
Burke's Pamphlet came forth, I would answer it. This appeared to me the more necessary to be
done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and that
while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is
an imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the
circumstances I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence
in the world, and that some other mode might be found out to settle the differences that should
occasionally arise in the neighbourhood of nations. This certainly might be done if Courts were
disposed to set honesty about it, or if countries were enlightened enough not to be made the
dupes of Courts. The people of America had been bred up in the same prejudices against France,
which at that time characterised the people of England; but experience and an acquaintance with
the French Nation have most effectually shown to the Americans the falsehood of those prejudices;
and I do not believe that a more cordial and confidential intercourse exists between any two
countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int =
10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
```

```
    h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
    h2_ElFormat = getElFormat("h2", this.sheet);
    p_ElFormat = getElFormat("p", this.sheet);
    displayText();
}

public function drawBackground():void
{
    var h:Number = this.storyTxt.y + this.storyTxt.height +
                    this.paddingTop + this.paddingBottom;
    var g:Graphics = this.graphics;
    g.beginFill(this.bgColor);
    g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
    g.endFill();
}

/**
 * Reads a set of style properties for a named style and then creates
 * a TextFormat object that uses the same properties.
 */
public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
{
var style:Object = ss.getStyle(styleName);
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                            style.fontFamily,
                            style.fontWeight,
                            FontPosture.NORMAL,
                            FontLookup.DEVICE,
                            RenderingMode.NORMAL,
                            CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                      style.fontSize,
                      style.color,
                      1,
                      TextRotation.AUTO,
                      TextBaseline.ROMAN,
                      TextBaseline.USE_DOMINANT_BASELINE,
                      0.0,
                      Kerning.ON,
                      0.0,
                      0.0,
                      "en",
                              BreakOpportunity.AUTO,
                              DigitCase.DEFAULT,
                              DigitWidth.DEFAULT,
                      LigatureLevel.NONE,
                              TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
```

```
            format.trackingRight = style.letterSpacing;
        }
        }
        return format;
        }

        public function displayText():void
        {
        headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
        headlineTxt.x = this.paddingLeft;
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as se emplea como clase base para la creación de bloques de texto. También incluye funciones de utilidad para cambiar el tamaño de fuente y las mayúsculas o minúsculas.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
```

```
        tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
        te = new TextElement(blockText,this.ef1);
        tb.content = te;
        this.breakLines();
    }

    private function breakLines()
    {
        var textLine:TextLine = null;
        var y:Number = 0;
        var lineNum:int = 0;
        while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
        {
            textLine.x = 0;
            textLine.y = y;
            y += this.leading*textLine.height;
            this.addChild(textLine);
        }
        for (var i:int = 0; i < this.numChildren; i++)
        {
            TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
        }
        this.totalTextLines = this.numChildren;
    }

    private function rebreakLines()
    {
        this.clearLines();
        this.breakLines();
    }

    private function clearLines()
    {
        while(this.numChildren)
        {
            this.removeChildAt(0);
        }
    }
```

```
        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as amplía la clase FormattedTextBlock y se utiliza para crear titulares. Incluye una función para ajustar texto con un espacio definido en la página.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                            Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
```

```
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
            {
                return growText(pointSize + 1, maxLines);
            }
        }

        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as gestiona el formato del texto con un diseño de varias columnas. Muestra el uso flexible de un objeto TextBlock como base para crear, dar formato y situar líneas de texto.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
```

```
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

Capítulo 23: Uso de Text Layout Framework

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Información general sobre Text Layout Framework

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Text Layout Framework (TLF) es una biblioteca ampliable de ActionScript. TLF está incorporada en el motor de texto de Adobe® Flash® Player 10 y Adobe® AIR® 1.5. TLF proporciona funciones de diseño de texto y tipográficas avanzadas para una tipografía innovadora en la web. La arquitectura se puede utilizar con Adobe® Flex® o Adobe® Flash® Professional. Los desarrolladores pueden utilizar o ampliar los componentes existentes, o bien, pueden utilizar la arquitectura para crear sus propios componentes de texto.

TLF incluye las siguientes capacidades:

- Texto bidireccional, texto vertical y unos 30 scripts de escritura, incluyendo árabe, hebreo, chino, japonés, coreano, tailandés, lao y vietnamita, entre otros.
- Selección, edición y flujo de texto en varias columnas y contenedores vinculados.
- Texto vertical, Tate-Chu-Yoko (texto vertical con horizontal) y justificador para la tipografía del este asiático.
- Controles tipográficos entre los que se incluyen ajuste entre caracteres, ligaduras, formato tipográfico de mayúsculas y minúsculas, caja de dígitos, anchura de dígitos y guiones discrecionales.
- Cortar, copiar, pegar, deshacer y gestos de ratón y teclado estándar para edición.
- APIs para el desarrollador para manipular contenido de texto, diseño y marcado y crear componentes de texto personalizados.
- Compatibilidad sólida con la lista, incluyendo marcadores personalizados y formatos de numeración.
- Imágenes en línea y reglas de posicionamiento.

TLF es una biblioteca de ActionScript 3.0 incorporada en Flash Text Engine (FTE) introducido en Flash Player 10. A FTE se puede acceder mediante el paquete `flash.text.engine`, que forma parte de la Interfaz de programación de aplicaciones (API) de Flash Player 10.

Sin embargo, la API de Flash Player ofrece un acceso de muy bajo nivel al motor de texto, lo que significa que algunas tareas pueden requerir una enorme cantidad de código. TLF encapsula el código de bajo nivel en APIs más sencillas. TLF también ofrece una arquitectura conceptual que organiza los bloques de creación básicos definidos por FTE en un sistema que sea más fácil de utilizar.

A diferencia de FTE, TLF no se basa en Flash Player. Se trata de una biblioteca de componentes independiente escrita en ActionScript 3.0 en su totalidad. Debido a que la arquitectura se puede ampliar, también se puede personalizar para entorno específicos. Tanto Flash Professional como el SDK de Flex incluyen componentes que se basan en el marco TLF.

Más temas de ayuda

Aplicación de marcado TLF "Flow"

Compatibilidad con scripts complejos

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

TLF proporciona compatibilidad con scripts complejos. La compatibilidad con scripts complejos incluye la capacidad para mostrar y editar scripts de derecha a izquierda. TLF también proporciona la capacidad para mostrar y editar una mezcla de scripts de izquierda a derecha y de derecha a izquierda como, por ejemplo, árabe y hebreo. La arquitectura no solo admite diseño de texto vertical para chino, japonés y coreano, sino que también es compatible con tate-chu-yoko (elementos TCY). Los elementos TCY son bloques de texto horizontal incorporados en extensiones de texto vertical. Se admiten los siguientes scripts:

- Latinos (inglés, español, francés, vietnamita, etc)
- Griego, cirílico, armenio, georgiano y etíope
- Árabe y hebreo
- Ideogramas de Han y Kana (chino, japonés y coreano) y Hangul Johab (coreano)
- Thai, Lao y Khmer
- Devanagari, Bengali, Gurmukhi, Malayalam, Telugu, Tamil, Gujarati, Oriya, Kannada y tibetano
- Tifinagh, Yi, Cherokee, alfabeto silábico canadiense, Deseret, Shavian, Vai, Tagalog, Hanunoo, Buhid y Tagbanwa

Uso de Text Layout Framework en Flash Professional y Flex

Se pueden utilizar las clases TLF directamente para crear componentes personalizados en Flash. Asimismo, Flash Professional CS5 proporciona una nueva clase, fl.text.TLFTextField, que encapsula la funcionalidad de TLF. Utilice la clase TLFTextField para crear campos de texto en ActionScript que utilicen las funciones avanzadas de visualización de TLF. Cree un objeto TLFTextField la misma manera que crearía un campo de texto con la clase TextField. Seguidamente, utilice la propiedad `textFlow` para asignar formato avanzado de las clases TLF.

También se puede utilizar Flash Professional para crear la instancia de TLFTextField en el escenario utilizando la herramienta de texto. Posteriormente ActionScript se puede utilizar para controlar el formato y el diseño del contenido del campo de texto utilizando las clases TLF. Para obtener más información, consulte la clase TLFTextField en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Si está trabajando en Flex, utilice las clases TLF. Para obtener más información, consulte “Uso de Text Layout Framework” en la página 447.

Uso de Text Layout Framework

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Si está trabajando en Flex o está creando componentes de texto personalizados, utilice las clases TLF. TLF es una biblioteca de ActionScript 3.0 incluida por completo en la biblioteca textLayout.swc. La biblioteca TLF contiene unas cien interfaces y clases de ActionScript 3.0 organizadas en diez paquetes. Estos paquetes son subpaquetes del paquete flashx.textLayout.

Clases de Text Layout Framework

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Las clases TLF se pueden agrupar en tres categorías:

- Clases de formato y estructuras de datos
- Clases de representación
- Clases de interacción con el usuario

Clases de formato y estructuras de datos

Los siguientes paquetes contienen las estructuras de datos y las clases de formato para TLF:

- flashx.textLayout.elements
- flashx.textLayout.formats
- flashx.textLayout.conversion

La estructura de datos principal de TLF es la jerarquía de flujo de texto, que se define en el paquete de elementos. Dentro de esta estructura, puede asignar estilos y atributos a extensiones de texto con el paquete de formatos. También es posible controlar el modo en que el texto se exporta e importa en la estructura de datos con el paquete de conversión.

Clases de representación

Los siguientes paquetes contienen las clases de representación para TLF:

- flashx.textLayout.factory
- flashx.textLayout.container
- flashx.textLayout.compose

Las clases de estos paquetes facilitan la representación de texto para su visualización en Flash Player. El paquete factory proporciona una forma sencilla de mostrar texto estático. El paquete container incluye clases e interfaces que definen contenedores de visualización para texto dinámico. El paquete compose define técnicas para situar y visualizar texto dinámico en contenedores.

Clases de interacción con el usuario

Los siguientes paquetes contienen las clases de interacción del usuario para TLF:

- flashx.textLayout.edit
- flashx.textLayout.operations
- flashx.textLayout.events

Los paquetes edit y operations definen clases que se pueden utilizar para permitir la edición de texto almacenado en las estructuras de datos. El paquete de eventos contiene clases de gestión de eventos.

Pasos generales para la creación de texto con Text Layout Framework

Los siguientes pasos describen el proceso general para crear texto con Text Layout Format:

1. Importe el texto con formato en las estructuras de datos TLF. Para obtener más información, consulte "Estructuración del texto con TLF" en la página 452 y "Formato de texto con TLF" en la página 456.
2. Cree uno o varios contenedores de objetos de visualización vinculados para el texto. Para obtener más información, consulte "Administración de contenedores de texto con TLF" en la página 458.

3 Asocie el texto en las estructuras de datos con los contenedores y establezca las opciones de edición y desplazamiento. Para obtener más información, consulte "Activación de las operaciones para deshacer, editar y selección de texto con TLF" en la página 459.

4 Cree un controlador de eventos para volver a transmitir el texto como respuesta a los eventos de cambio de tamaño (resize) (u otros). Para obtener más información, consulte "Control de eventos con TLF" en la página 459.

Ejemplo de Text Layout Framework: News Layout

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

En el siguiente ejemplo se muestra el uso de TLF para diseñar una sencilla página de periódico. La página incluye un titular grande, un subtí-tulo y una sección de cuerpo de varias columnas:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.geom.Rectangle;

    import flashx.textLayout.compose.StandardFlowComposer;
    import flashx.textLayout.container.ContainerController;
    import flashx.textLayout.container.ScrollPolicy;
    import flashx.textLayout.conversion.TextConverter;
    import flashx.textLayout.elements.TextFlow;
    import flashx.textLayout.formats.TextLayoutFormat;

    public class TLFNewsLayout extends Sprite
    {
        private var hTextFlow:TextFlow;
        private var headContainer:Sprite;
        private var headlineController:ContainerController;
        private var hContainerFormat:TextLayoutFormat;

        private var bTextFlow:TextFlow;
        private var bodyTextContainer:Sprite;
        private var bodyController:ContainerController;
        private var bodyTextContainerFormat:TextLayoutFormat;

        private const headlineMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008'><flow:p textAlign='center'><flow:span
fontFamily='Helvetica' fontSize='18'>TLF News Layout Example</flow:span><flow:br/><flow:span
fontFamily='Helvetica' fontSize='14'>This example formats text like a newspaper page with a
headline, a subtitle, and multiple columns</flow:span></flow:p></flow:TextFlow>";

        private const bodyMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008' fontSize='12' textIndent='10' marginBottom='15'
paddingTop='4' paddingLeft='4'><flow:p marginBottom='inherit'><flow:span>There are many
</flow:span><flow:span fontStyle='italic'>such</flow:span><flow:span> lime-kilns in that tract
of country, for the purpose of burning the white marble which composes a large part of the
substance of the hills. Some of them, built years ago, and long deserted, with weeds growing in
the vacant round of the interior, which is open to the sky, and grass and wild-flowers rooting
themselves into the chinks of the stones, look already like relics of antiquity, and may yet be
overspread with the lichens of centuries to come. Others, where the lime-burner still feeds his
```

```
daily and nightlong fire, afford points of interest to the wanderer among the hills, who seats
himself on a log of wood or a fragment of marble, to hold a chat with the solitary man. It is a
lonesome, and, when the character is inclined to thought, may be an intensely thoughtful
occupation; as it proved in the case of Ethan Brand, who had mused to such strange purpose, in
days gone by, while the fire in this very kiln was burning.</flow:span></flow:p><flow:p
marginBottom='inherit'><flow:span>The man who now watched the fire was of a different order, and
troubled himself with no thoughts save the very few that were requisite to his business. At
frequent intervals, he flung back the clashing weight of the iron door, and, turning his face
from the insufferable glare, thrust in huge logs of oak, or stirred the immense brands with a
long pole. Within the furnace were seen the curling and riotous flames, and the burning marble,
almost molten with the intensity of heat; while without, the reflection of the fire quivered on
the dark intricacy of the surrounding forest, and showed in the foreground a bright and ruddy
little picture of the hut, the spring beside its door, the athletic and coal-begrimed figure of
the lime-burner, and the half-frightened child, shrinking into the protection of his father's
shadow. And when again the iron door was closed, then reappeared the tender light of the half-
full moon, which vainly strove to trace out the indistinct shapes of the neighboring mountains;
and, in the upper sky, there was a flitting congregation of clouds, still faintly tinged with the
rosy sunset, though thus far down into the valley the sunshine had vanished long and long
ago.</flow:span></flow:p></flow:TextFlow>";

        public function TLFNewsLayout()
        {
            //wait for stage to exist
            addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        }

        private function onAddedToStage(evtObj:Event):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            // Headline text flow and flow composer
            hTextFlow = TextConverter.importToFlow(headlineMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // initialize the headline container and controller objects
            headContainer = new Sprite();
            headlineController = new ContainerController(headContainer);
            headlineController.verticalScrollPolicy = ScrollPolicy.OFF;
            hContainerFormat = new TextLayoutFormat();
            hContainerFormat.paddingTop = 4;
            hContainerFormat.paddingRight = 4;
            hContainerFormat.paddingBottom = 4;
            hContainerFormat.paddingLeft = 4;

            headlineController.format = hContainerFormat;
            hTextFlow.flowComposer.addController(headlineController);
            addChild(headContainer);
            stage.addEventListener(flash.events.Event.RESIZE, resizeHandler);

            // Body text TextFlow and flow composer
            bTextFlow = TextConverter.importToFlow(bodyMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // The body text container is below, and has three columns
            bodyTextContainer = new Sprite();
```

```
            bodyController = new ContainerController(bodyTextContainer);
            bodyTextContainerFormat = new TextLayoutFormat();
            bodyTextContainerFormat.columnCount = 3;
            bodyTextContainerFormat.columnGap = 30;

            bodyController.format = bodyTextContainerFormat;
            bTextFlow.flowComposer.addController(bodyController);
            addChild(bodyTextContainer);
            resizeHandler(null);
        }

        private function resizeHandler(event:Event):void
        {
            const verticalGap:Number = 25;
            const stagePadding:Number = 16;
            var stageWidth:Number = stage.stageWidth - stagePadding;
            var stageHeight:Number = stage.stageHeight - stagePadding;
            var headlineWidth:Number = stageWidth;
            var headlineContainerHeight:Number = stageHeight;

            // Initial compose to get height of headline after resize
            headlineController.setCompositionSize(headlineWidth,
headlineContainerHeight);
            hTextFlow.flowComposer.compose();
            var rect:Rectangle = headlineController.getContentBounds();
            headlineContainerHeight = rect.height;

            // Resize and place headline text container
            // Call setCompositionSize() again with updated headline height
            headlineController.setCompositionSize(headlineWidth, headlineContainerHeight );
            headlineController.container.x = stagePadding / 2;
            headlineController.container.y = stagePadding / 2;
            hTextFlow.flowComposer.updateAllControllers();

            // Resize and place body text container
            var bodyContainerHeight:Number = (stageHeight - verticalGap -
headlineContainerHeight);
            bodyController.format = bodyTextContainerFormat;
            bodyController.setCompositionSize(stageWidth, bodyContainerHeight );
            bodyController.container.x = (stagePadding/2);
            bodyController.container.y = (stagePadding/2) + headlineContainerHeight +
verticalGap;
            bTextFlow.flowComposer.updateAllControllers();
        }
    }
}
```

La clase TLFNewsLayout utiliza dos contenedores de texto. Un contenedor muestra un titular y subtítulo y el otro muestra el texto de cuerpo de tres columnas. Para que resulte más fácil, el texto se codifica en el ejemplo como texto de marcado TLF. La variable `headlineMarkup` contiene el titular y el subtítulo y la variable `bodyMarkup` incluye el texto principal. Para obtener más información sobre TLF Markup, consulte "Estructuración del texto con TLF" en la página 452.

Tras la inicialización, la función `onAddedToStage()` importa el texto del titular en un objeto TextFlow, que es la estructura de datos principal de TLF:

```
hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);
```

A continuación, se crea un objeto Sprite para el contenedor y un controlador se crea y se asocia con el contenedor:

```
headContainer = new Sprite();
headlineController = new ContainerController(headContainer);
```

El controlador se inicializa para establecer el formato, el desplazamiento y otras opciones. El controlador contiene geometría que define los límites del contenedor al que se transmite el texto. Un objeto TextLayoutFormat contiene la opciones de formato:

```
hContainerFormat = new TextLayoutFormat();
```

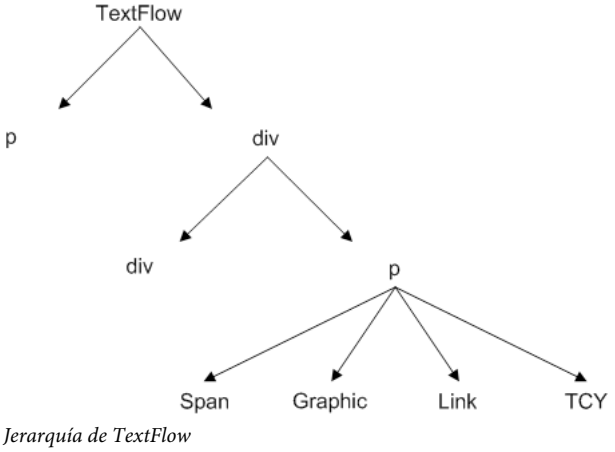
El controlador se asigna al compositor de flujo y la función añade el contenedor a la lista de visualización. La visualización y composición real de los contenedores se difiere al método `resizeHandler()`. La misma secuencia de pasos se lleva a cabo para inicializar el objeto TextFlow del cuerpo.

El método `resizeHandler()` mide el espacio disponible para procesar los contenedores y calculan su tamaño en consecuencia. Una llamada inicial al método `compose()` permite el cálculo de la altura adecuada del contenedor del titular. El método `resizeHandler()` puede situar y mostrar el contenedor del titular con el método `updateAllControllers()`. Finalmente, el método `resizeHandler()` utiliza el amaño del contenedor del titular para determinar la ubicación del contenedor del texto principal.

Estructuración del texto con TLF

TLF utiliza un árbol jerárquico para representar texto. Cada nodo del árbol es una instancia de la clase definida en el paquete de elementos. Por ejemplo, el nodo raíz del árbol siempre es una instancia de la clase TextFlow. La clase TextFlow representa una historia completa del texto. Una historia es una colección de texto y otros elementos que se tratan como una unidad o flujo. Una sola historia puede necesitar varias columnas o contenedores de texto para mostrar.

Aparte del nodo raíz, los elementos restantes se basan en parte en elementos XHTML. El siguiente diagrama muestra la jerarquía del marco:



Jerarquía de TextFlow

Marcado de Text Layout Framework

EL conocimiento de la estructura de TLF también resulta útil a la hora de tratar con el marcado de Text Layout Framework. El marcado de TLF es una representación XML del texto que forma parte de TLF. Aunque la arquitectura también admite otros formatos XML, el marcado es único, ya que se basa específicamente en la estructura de la jerarquía de TextFlow. Si opta por exportar XML desde TextFlow utilizando este formato de marcado, XML se exportará con esta jerarquía intacta.

TLF Markup proporciona la representación de texto de más alta fidelidad en una jerarquía de TextFlow. El lenguaje de marcado proporciona etiquetas para cada uno de los elementos básicos de la jerarquía de TextFlow y también ofrece atributos para todas las propiedades de formato disponibles en la clase TextLayoutFormat.

La siguiente tabla contiene las etiquetas que se pueden utilizar en marcado TLF.

Elemento	**Descripción**	**Elementos secundarios**	**Clase**
textflow	Elemento raíz del marcado.	div, p	TextFlow
div	División en un elemento TextFlow. Puede contener un grupo de párrafos.	div, list, p	DivElement
p	Párrafo.	a, tcy, span, img, tab, br, g	ParagraphElement
a	Vínculo.	tcy, span, img, tab, br, g	LinkElement
tcy	Extensión de texto horizontal (utilizada en un elemento TextFlow vertical).	a, span, img, tab, br, g	TCYElement
span	Extensión de texto en un párrafo.		SpanElement
img	Imagen de un párrafo.		InlineGraphicElement
tab	Carácter de tabulación		TabElement
BR	Carácter de salto. Utilizado para finalizar una línea de un párrafo; el texto continuará en la siguiente línea, pero permanece en el mismo párrafo.		BreakElement
linkNormalFormat	Define los atributos de formato utilizados por los vínculos de estado normal.	TextLayoutFormat	TextLayoutFormat
linkActiveFormat	Define los atributos de formato utilizados para los vínculos de estado activo, cuando el ratón está presionado sobre un vínculo.	TextLayoutFormat	TextLayoutFormat
linkHoverFormat	Define los atributos de formato utilizados para los vínculos al pasar por encima con el ratón, cuando este se encuentra dentro de los límites (desplazamiento sobre) un vínculo.	TextLayoutFormat	TextLayoutFormat
li	Elemento de lista. Debe estar dentro de un elemento de lista.	div, li, list, p	ListItemElement
list	Una lista. Las listas se pueden anidar o situar de forma adyacente entre sí. Los distintos esquemas de numeración o etiquetado se pueden aplicar a los elementos de lista.	div, li, list, p	ListElement
v	Elemento de grupo. Se utiliza para agrupar elementos en un párrafo. Permite anidar elementos bajo el nivel de párrafo.	a, tcy, span, img, tab, br, g	SubParagraphGroupElement

Más temas de ayuda

Marcado de listas de TLF 2.0

Objetos SubParagraphGroupElements y typeName de TLF 2.0

Uso de listas numeradas y con viñetas

Las clases ListElement y ListItemElement pueden utilizarse para añadir listas con viñetas a los controles de texto. Las listas con viñetas se pueden anidar y personalizar para que utilicen viñetas distintas (o marcadores) y numeración automática, así como numeración con estilo.

Para crear listas en los flujos de texto, utilice la etiqueta `<list>`. Posteriormente se utilizan las etiquetas `<li>` en la etiqueta `<list>` para cada elemento de la lista. Es posible personalizar la apariencia de las viñetas mediante la clase ListMarkerFormat.

En el siguiente ejemplo se crean listas sencillas:

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Las listas se pueden anidar en otras listas, tal y como se muestra en el ejemplo siguiente:

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:list paddingRight="24" paddingLeft="24">
        <flow:li>Item 1a</flow:li>
        <flow:li>Item 1b</flow:li>
        <flow:li>Item 1c</flow:li>
    </flow:list>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Para personalizar el tipo de marcador en la lista, utilice la propiedad `listStyleType` de ListElement. Esta propiedad puede ser cualquier valor definido por la clase ListStyleType (por ejemplo, `check`, `circle`, `decimal` y `box`). En el siguiente ejemplo se crean listas con distintos tipos de marcador y un incremento del contador personalizado:

```
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperAlpha">
<flow:li>upperAlpha item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list
paddingRight="24" paddingLeft="24" listStyleType="lowerAlpha"> <flow:li>lowerAlpha
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24"
paddingLeft="24" listStyleType="upperRoman"> <flow:li>upperRoman item</flow:li>
<flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerRoman"> <flow:listMarkerFormat> <!-- Increments the list by 2s rather than
1s. --> <flow:ListMarkerFormat counterIncrement="ordered 2"/> </flow:listMarkerFormat>
<flow:li>lowerRoman item</flow:li> <flow:li>another</flow:li> </flow:list>
```

La clase ListMarkerFormat se emplea para definir el contador. Además de definir el incremento de un contador, también es posible personalizar el contador, restableciéndolo con la propiedad `counterReset`.

Se puede personalizar más el aspecto de los marcadores de las listas con el uso de las propiedades `beforeContent` y `afterContent` de ListMarkerFormat. Estas propiedades se aplican al contenido que aparece antes y después del contenido del marcador.

En el siguiente ejemplo se añade la cadena “XX” antes del marcador y “YY” después del mismo:

```
<flow:list listStyleType="upperRoman" paddingLeft="36" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="XX"
            afterContent="YY"
            counterIncrement="ordered -1"/>
        </flow:listMarkerFormat>
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

La propia propiedad `content` puede definir más las personalizaciones del formato del marcador En el siguiente ejemplo se muestra un marcador numeral Roman ordenado y en mayúsculas:

```
<flow:list listStyleType="disc"  paddingLeft="96" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="Section "
            content="counters(ordered,&quot;*&quot;,upperRoman)"
            afterContent=": "/>
    </flow:listMarkerFormat>
    <flow:li>Item 1</li>
    <flow:li>Item 2</li>
    <flow:li>Item 3</li>
</flow:list>
```

Tal y como se muestra en el ejemplo anterior, la propiedad `content` también puede insertar un sufijo: una cadena que aparece tras el marcador, pero antes de `afterContent`. Para insertar esta cadena al proporcionar contenido XML en el flujo, ajuste la cadena en las entradas HTML `&quote;` en lugar de comillas (`"<string>"`).

Más temas de ayuda

Marcado de listas de TLF 2.0

Uso del margen en TLF

Todos los objetos FlowElement admiten propiedades de margen que se utilizan para controlar la posición del área de contenido de cada elemento y el espacio entre las áreas de contenido.

La anchura total de un elemento es la suma de la anchura de su contenido, más las propiedades `paddingLeft` y `paddingRight`. La altura total de un elemento es la suma de la altura de su contenido, más las propiedades `paddingTop` y `paddingBottom`.

El margen es el espacio entre el borde y el contenido. Las propiedades de margen son `paddingBottom`, `paddingTop`, `paddingLeft` y `paddingRight`. Se puede aplicar margen al objeto TextFlow y a los siguientes elementos secundarios:

- div
- img
- li
- list
- p

Las propiedades de margen no se pueden aplicar a los elementos de delimitación.

En el siguiente ejemplo se establecen las propiedades de margen en TextFlow:

```
<flow:TextFlow version="2.0.0" xmlns:flow="http://ns.adobe.com/textLayout/2008" fontSize="14"
textIndent="15" paddingTop="4" paddingLeft="4" fontFamily="Times New Roman">
```

Los valores válidos para las propiedades de margen son un número (en píxeles), “auto” o “inherit”. El valor predeterminado es “auto”, lo que significa que se calcula automáticamente y se establece en 0, para todos los elementos excepto ListElement. Para ListElements, “auto” es 0 excepto en la parte de inicio de la lista donde se utiliza el valor de la propiedad `listAutoPadding`. El valor predeterminado de `listAutoPadding` es 40, lo que proporciona a las listas una sangría predeterminada.

Las propiedades de margen no se heredan de forma predeterminada. Los valores “auto” e “inherit” son constantes definidas por la clase FormatValue.

Las propiedades de margen pueden ser valores negativos.

Más temas de ayuda

Cambios de margen en TLF 2.0 (en inglés)

Formato de texto con TLF

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El paquete flashx.textLayout.formats contiene interfaces y clases que permiten asignar formatos a cualquier elemento FlowElement en el árbol de la jerarquía del flujo de texto. Existen dos formas de aplicar el formato de contenedor. Se puede asignar un formato específico de forma individual o asignar un grupo de formatos simultáneamente con un objeto de formato especial.

La interfaz ITextLayoutForma contiene todos los formatos que se pueden aplicar a FlowElement. Algunos formatos se aplican a todo el párrafo de texto, pero lógicamente no se aplican a caracteres individuales. Por ejemplo, formatos como la justificación y las tabulaciones se aplican a párrafos completos, pero no se aplican a caracteres individuales.

Asignación de formatos a un objeto FlowElement con propiedades

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Es posible establecer formatos en cualquier elemento FlowElement mediante asignación de propiedades. La clase FlowElement implementa la interfaz ITextLayoutFormat, por lo que cualquier subclase de la clase FlowElement también debe implementar esta interfaz.

Por ejemplo, el siguiente código muestra cómo asignar formatos individuales a una instancia de ParagraphElement:

```
var p:ParagraphElement = new ParagraphElement();
p.fontSize = 18;
p.fontFamily = "Arial";
```

Asignación de formatos a un objeto FlowElement con la clase TextLayoutFormat

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Puede aplicar formatos a un objeto FlowElement con la clase TextLayoutFormat. Puede utilizar esta clase para crear un objeto de formato especial que contenga todos los valores de formato que desee. Puede asignar ese objeto a la propiedad `format` de cualquier objeto TextFlow, DivElement o ContainerController. TextLayoutFormat y FlowElement implementan la interfaz ITextLayoutFormat. Esta disposición garantiza que ambas clases contengan las mismas propiedades de formato.

Para obtener más información, consulte la clase TextLayoutFormat en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Herencia de formato

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Los formatos se heredan mediante la jerarquía de flujo de texto. Si se asigna una instancia de TextLayoutFormat a una instancia de FlowElement con elementos secundarios, la arquitectura inicia un proceso denominado *cascada*. Durante una cascada, la arquitectura examina de forma sucesiva cada nodo de la jerarquía que hereda de FlowElement. Posteriormente determina si asignar los valores heredados a cada propiedad de formato. Durante la cascada se aplican las siguientes reglas:

1. Los valores de propiedad se heredan únicamente de un ascendiente inmediato (denominado a veces elemento principal).
2. Los valores de propiedad solo se heredan si una propiedad aún no tiene un valor (es decir, el valor es `undefined`)..
3. Algunos atributos no heredan valores si son undefined, a no ser que el valor del atributo se establezca específicamente en “inherit” o como constante `flashx.textLayout.formats.FormatValue.INHERIT`.

Por ejemplo, si el valor `fontSize` es establece para un contenedor a nivel de TextFlow, la configuración aplica todos los contenedores en TextFlow. Es decir, los valores caen en cascada hacia abajo en la jerarquía del flujo de texto. No obstante, puede sustituir el valor de un contenedor determinado asignando un nuevo valor directamente al contenedor. Como contraejemplo, si se establece el valor `backgroundColor` en el nivel de TextFlow, los elementos secundarios de TextFlow no heredan ese valor. La propiedad `backgroundColor` no hereda de su elemento principal durante una cascada. Para omitir este comportamiento, establezca la propiedad `marginLeft` de cada elemento secundario en `flashx.textLayout.formats.FormatValue.INHERIT`..

Para obtener más información, consulte la clase TextLayoutFormat en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Importación y exportación de texto con TLF

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La clase TextConverter del paquete flashx.textLayout.conversion.* permite importar texto y exportar texto desde TLF. Utilice esta clase si va a cargar texto en tiempo de ejecución en lugar de compilar el texto en el archivo SWF. Esta clase también se puede utilizar para exportar texto almacenado en una instancia de TextFlow en un objeto String o XML.

Tanto la importación como la exportación son procedimientos sencillos. Se llama al método `export()` o `importToFlow()`; ambos forman parte de la clase TextFilter. Los dos métodos son estáticos, lo que significa que se llaman en la clase TextFilter en lugar de en una instancia de la clase TextFilter.

Las clases del paquete flashx.textLayout.conversion proporciona una flexibilidad considerable en la que el usuario opta por almacenar el texto. Por ejemplo, si el texto se almacena en una base de datos, se puede importar en la arquitectura para su visualización. Posteriormente se pueden utilizar las clases del paquete flashx.textLayout.edit para cambiar el texto y exportar de nuevo el texto modificado a la base de datos.

Para obtener más información, consulte la case flashx.textLayout.conversion en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Administración de contenedores de texto con TLF

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Una vez que el texto se almacena en las estructuras de datos de TLF, Flash Player puede mostrarlo. El texto que se almacena en la jerarquía de flujo se debe convertir a un formato que pueda mostrar Flash Player. TLF ofrece dos formas de crear objetos de visualización desde un flujo. El primer enfoque más sencillo resulta adecuado para mostrar texto estático. El segundo enfoque más complejo permite crear texto dinámico que se puede seleccionar y editar. En ambos casos, en última instancia el texto se convierte en instancias de la clase TextLine, que forma parte del nuevo paquete flash.text.engine.* de Flash Player 10.

Creación de texto estático

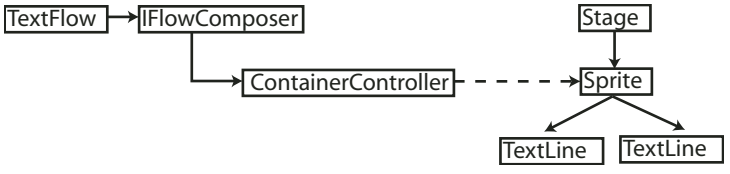
En el enfoque sencillo se utiliza la clase TextFlowTextLineFactory, que puede encontrarse en el paquete flashx.textLayout.factory. Más allá de su simplicidad, la ventaja de este enfoque radica en que presenta una menor superficie de memoria que el esquema de FlowComposer. Este enfoque se recomienda para texto estático que el usuario no necesita editar, seleccionar ni desplazar.

Para obtener más información, consulte la clase TextFlowTextLineFactory en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Creación de contenedores y texto dinámico

Utilice un compositor de flujo si desea tener más control sobre la visualización del texto que el que proporciona TextFlowTextLineFactory. Por ejemplo, con un compositor de flujo, los usuarios pueden seleccionar y editar el texto. Para obtener más información, consulte "Activación de las operaciones para deshacer, editar y selección de texto con TLF" en la página 459.

Un compositor de flujo es una instancia de la clase StandardFlowComposer en el paquete flashx.textLayout.compose. Un compositor de flujo administra la conversión de TextFlow en instancias de TextLine y también la ubicación de las instancias de TextLine en uno o varios contenedores.



El objeto IFlowComposer dispone de varios ContainerControllers o de ninguno de ellos.

Todas las instancias de TextFlow disponen de un objeto correspondiente que implementa la interfaz IFlowComposer. A este objeto IFlowComposer se puede acceder mediante la propiedad `TextFlow.flowComposer`. Se puede llamar a métodos definidos mediante la interfaz IFlowComposer a través de esta propiedad. Estos métodos permiten asociar el texto con uno o varios contenedores y preparan el texto para su visualización en un contenedor.

Un contenedor es simplemente una instancia de la clase Sprite, que es una subclase de la clase DisplayObjectContainer. Ambas clases forman parte de la API de la lista de visualización de Flash Player. Un contenedor es una forma más avanzada del rectángulo de delimitación utilizado con la clase TextLineFactory. Al igual que sucede con el rectángulo de delimitación, un contenedor limita el área donde aparecerán las instancias de TextLine. A diferencia de un rectángulo de delimitación, un contenedor dispone de un objeto "controller" correspondiente. El controlador administra el desplazamiento, composición, vinculación, formato y control de eventos para un contenedor o conjunto de contenedores. Cada contenedor cuenta con un objeto controlador correspondiente que es una instancia de la clase ContainerController del paquete flashx.textLayout.container.

Para mostrar texto, cree un objeto controlador para administrar y asociarlo con el compositor de flujo. Una vez asociado el contenedor, debe componer el texto antes de que pueda visualizarse. Por lo tanto, los contenedores disponen de dos estados: composición y visualización. La composición del texto es el proceso de conversión del texto desde la jerarquía de flujo de texto en instancias de TextLine y el cálculo de la forma en que esas instancias se ajustan al contenedor. La visualización es el proceso de actualización de la lista de visualización de Flash Player.

Para obtener más información, consulte IFlowComposer, StandardFlowComposer y ContainerController en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Activación de las operaciones para deshacer, editar y selección de texto con TLF

Flash Player 9.0 y posterior, Adobe AIR 1.0 y posterior

La capacidad de seleccionar o editar texto se controla en el nivel de TextFlow. Todas las instancias de la clase TextFlow disponen de un administrador de interacción asociado. Se puede acceder al administrador de interacción del objeto TextFlow mediante la propiedad `TextFlow.interactionManager` del objeto. Para activar la selección, asigne una instancia de la clase SelectionManager a la propiedad `interactionManager`. Para activar tanto la selección como la edición, asigne una instancia de la clase EditManager en lugar de una instancia de la clase SelectionManager. Para habilitar las operaciones de deshacer, cree una instancia de la clase UndoManager e inclúyala como argumento al llamar al constructor para EditManager. La clase UndoManager permite mantener un historial de las actividades de edición más recientes del usuario y permite al usuario deshacer y rehacer ediciones específicas. Estas tres clases forman parte del paquete de edición.

Para obtener más información, consulte SelectionManager, EditManager y UndoManager en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Control de eventos con TLF

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Los objetos TextFlow distribuyen eventos en diversas circunstancias, entre las que se incluyen:

- Cuando el texto o el diseño cambian.

- Antes de que comience una operación y una vez finalizada una operación.
- Si cambia el estado de un objeto FlowElement.
- Cuando se completa una operación de composición.

Para obtener más información, consulte la clase flashx.textLayout.events en la Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Más temas de ayuda

Eventos FlowElement y LinkElement y EventMirrors de TLF

Posicionamiento de imágenes en el texto

Para posicionar InlineGraphicElement en el texto, se utilizan las siguientes propiedades:

- Propiedad `float` de la clase InlineGraphicElement.
- Propiedad `clearFloats` de FlowElement.

La propiedad `float` controla la situación del gráfico y el texto de alrededor. La propiedad `clearFloats` controla la situación de los elementos del párrafo relativos a `float`.

Para controlar la ubicación de una imagen en un elemento de texto, se utiliza la propiedad `float`. En el siguiente ejemplo se añade una imagen a un párrafo y se alinea a la izquierda de modo que el texto se ajuste a la derecha:

```
<flow:p paragraphSpaceAfter="15" >Images in a flow are a good thing. For example, here is a
float. It should show on the left: <flow:img float="left" height="50" width="19"
source="../assets/bulldog.jpg"></flow:img> Don't you agree? Another sentence here. Another
sentence here. Another sentence here. Another sentence here. Another sentence here. Another
sentence here. Another sentence here. Another sentence here.</flow:p>
```

Los valores válidos de la propiedad `float` son “left”, “right”, “start”, “end” y “none”. La clase Float define estas constantes. El valor predeterminado es “none”.

La propiedad `clearFloats` resulta útil cuando se desea ajustar la posición de inicio de los siguientes párrafos que normalmente se ajustan alrededor de la imagen. Por ejemplo, supongamos que se dispone de una imagen que es más grande que el primer párrafo. Para garantizar que el segundo párrafo comience *tras* la imagen, establezca la propiedad `clearFloats`.

En el siguiente ejemplo se utiliza una imagen que es más alta que el texto en el primer párrafo. Para que el segundo párrafo comience tras la imagen en el bloque de texto, en este ejemplo se establece la propiedad `clearFloats` en el segundo párrafo hasta “end”.

```
<flow:p paragraphSpaceAfter="15" >Here is another float, it should show up on the right:
<flow:img float="right" height="50" elementHeight="200" width="19"
source="../assets/bulldog.jpg"></flow:img>We'll add another paragraph that should clear past
it.</flow:p><flow:p clearFloats="end" >This should appear after the previous float on the
right.</flow:p>
```

Los valores válidos de la propiedad `clearFloats` son “left”, “right”, “end”, “start”, “none”, y “both”. La clase ClearFloats define estas constantes. También se puede definir la propiedad `clearFloats` como “inherit”, que es una constante definida por la clase FormatValue. El valor predeterminado es “none”.

Más temas de ayuda

Cuadros flotantes de TLF (en inglés)

Capítulo 24: Trabajo con sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript se ha diseñado para crear aplicaciones interactivas y envolventes. Un elemento a menudo olvidado de las aplicaciones envolventes es el sonido. Se pueden añadir efectos de sonido a un videojuego, comentarios de audio a una interfaz de usuario o incluso crear un programa que analice archivos MP3 cargados por Internet, con el sonido como un componente importante de la aplicación.

Es posible cargar archivos de audio externos y trabajar con el audio incorporado en un archivo SWF. Asimismo, se muestra cómo controlar el audio, crear representaciones visuales de la información de sonido y capturar el sonido desde un micrófono del usuario.

Más temas de ayuda

Paquete flash.media

flash.events.SampleDataEvent

Fundamentos de la utilización de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los equipos pueden capturar y codificar audio digital (representación mediante ordenador de información de sonido) y almacenarlo y recuperarlo para su reproducción con altavoces. El sonido se puede reproducir utilizando Adobe® Flash® Player o Adobe® AIR™ y ActionScript.

Los datos de sonido convertidos a formato digital tienen varias características, como el volumen del sonido y si este es estéreo o mono. Al reproducir un sonido en ActionScript, también se pueden ajustar estas características (por ejemplo, aumentar su volumen o hacer que parezca proceder de una dirección determinada).

Para poder controlar un sonido en ActionScript es necesario tener la información de sonido cargada en Flash Player o AIR. Hay cuatro cinco de obtener datos de audio en Flash Player o AIR y trabajar con ellos mediante ActionScript.

- Cargar un archivo de sonido externo como, por ejemplo, un archivo mp3 en el archivo SWF.
- Incorporar la información de sonido en el archivo SWF directamente cuando se esté creando.
- Capturar el audio desde un micrófono incorporado al equipo del usuario.
- Transmitir audio desde un servidor.
- Generar y reproducir audio dinámicamente.

Al cargar datos de sonido desde un archivo de sonido externo, se puede iniciar la reproducción del principio del archivo de sonido mientras se cargan los restantes datos.

Aunque hay varios formatos de archivo de sonido que se utilizan para codificar audio digital, ActionScript 3.0, Flash Player y AIR admiten archivos de sonido almacenados en formato MP3. No pueden cargar ni reproducir directamente archivos de sonido con otros formatos, como WAV o AIFF.

Al trabajar con sonido en ActionScript, es probable que se utilicen varias clases del paquete flash.media. La clase Sound se utiliza para acceder a la información de audio mediante la carga de un archivo de sonido o la asignación de una función a un evento que muestrea los datos de sonido y posteriormente se inicia la reproducción. Una vez iniciada la reproducción de un sonido, Flash Player y AIR proporcionan acceso a un objeto SoundChannel. Puesto que un archivo de audio que se ha cargado puede ser uno de varios sonidos que se reproducen en el ordenador de un usuario, cada sonido que se reproduce utiliza su propio objeto SoundChannel; la salida combinada de todos los objetos SoundChannel mezclados es lo que se reproduce realmente a través de los altavoces del ordenador. La instancia de SoundChannel se utiliza para controlar las propiedades del sonido y detener su reproducción. Por último, si se desea controlar el audio combinado, la clase SoundMixer permite controlar la salida mezclada.

Asimismo, se pueden utilizar otras muchas clases para realizar tareas más específicas cuando se trabaja con sonido en ActionScript; para más información sobre todas las clases relacionadas con el sonido, consulte "Aspectos básicos de la arquitectura de sonido" en la página 462.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes utilizados:

Amplitud Distancia de un punto de la forma de onda del sonido desde la línea cero o de equilibrio.

Velocidad de bits Cantidad de datos que se codifican o se transmiten en cada segundo de un archivo de sonido. En los archivos MP3, la velocidad suele expresarse en miles de bits por segundo (kbps). Una velocidad superior suele implicar una onda de sonido de mayor calidad.

Almacenamiento en búfer Recepción y almacenamiento de los datos de sonido antes de que se reproduzcan.

mp3 MPEG-1 Audio Layer 3, o MP 3, es un formato de compresión de sonido conocido.

Desplazamiento lateral Posición de una señal de audio entre los canales izquierdo y derecho de un campo de sonido estéreo.

Pico Punto más alto de una forma de onda.

Velocidad de muestreo Define el número de muestras por segundo que se toman de una señal de audio analógica para crear una señal digital. La velocidad de muestra del audio de CD estándar es de 44,1 kHz o 44.100 muestras por segundo.

Transmisión Proceso de reproducción del principio de un archivo de sonido o de vídeo mientras se descargan los datos restantes desde un servidor.

Volumen Intensidad de un sonido.

Forma de onda Forma de un gráfico de las distintas amplitudes de una señal de sonido a lo largo del tiempo.

Aspectos básicos de la arquitectura de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las aplicaciones pueden cargar datos de sonido de cinco orígenes principales:

- Archivos de sonido externos cargados en tiempo de ejecución
- Recursos de sonido incorporados en el archivo SWF de la aplicación
- Datos de sonido de un micrófono conectado al sistema del usuario
- Datos de sonido transmitidos desde un servidor multimedia remoto, como Flash Media Server

- Datos de sonido generados dinámicamente mediante el uso del controlador de eventos `sampleData`.

Los datos de sonido se pueden cargar completamente antes de su reproducción, o bien se pueden transmitirse, es decir, se pueden reproducir mientras se están cargando.

Las clases de sonido de ActionScript 3.0 admiten archivos de sonido que se almacenan en formato mp3. No pueden cargar ni reproducir directamente archivos de sonido con otros formatos, como WAV o AIFF. No obstante, cuando se comienza con Flash Player 9.0.115.0, los archivos de audio AAC se pueden cargar y reproducir utilizando la clase NetStream. Se trata de la misma técnica que se utiliza para cargar y reproducir contenido de vídeo. Para obtener más información sobre esta técnica, consulte "Trabajo con vídeo" en la página 494.

Adobe Flash Professional permite importar archivos de sonido WAV o AIFF e incorporarlos en los archivos SWF de la aplicación con formato MP3. La herramienta de edición de Flash también permite comprimir archivos de sonido incorporados para reducir su tamaño de archivo, aunque esta reducción de tamaño se compensa con una mejor calidad de sonido. Para más información, consulte "Importación de sonidos" en *Uso de Flash*.

La arquitectura de sonido de ActionScript 3.0 utiliza las siguientes clases del paquete flash.media.

Clase	**Descripción**
flash.media.Sound	La clase Sound controla la carga del sonido, administra las propiedades de sonido básicas e inicia la reproducción de sonido.
flash.media.SoundChannel	Cuando una aplicación reproduce un objeto Sound, se crea un nuevo objeto SoundChannel para controlar la reproducción. El objeto SoundChannel controla el volumen de los canales de reproducción izquierdo y derecho del sonido. Cada sonido que se reproduce tiene su propio objeto SoundChannel.
flash.media.SoundLoaderContext	La clase SoundLoaderContext especifica cuántos segundos de búfer se utilizarán al cargar un sonido, y si Flash Player o AIR buscan un archivo de política en el servidor al cargar un archivo. Un objeto SoundLoaderContext se utiliza como parámetro del método `Sound.load()`.
flash.media.SoundMixer	La clase SoundMixer controla la reproducción y las propiedades de seguridad que pertenecen a todos los sonidos de una aplicación. De hecho, se mezclan varios canales de sonido mediante un objeto SoundMixer común, de manera que los valores de propiedad de dicho objeto afectarán a todos los objetos SoundChannel que se reproducen actualmente.
flash.media.SoundTransform	La clase SoundTransform contiene valores que controlan el desplazamiento y el volumen de sonido. Un objeto SoundTransform puede aplicarse a un objeto SoundChannel, al objeto SoundMixer global o a un objeto Microphone, entre otros.
flash.media.ID3Info	Un objeto ID3Info contiene propiedades que representan información de metadatos ID3, almacenada a menudo en archivos de sonido MP3.
flash.media.Microphone	La clase Microphone representa un micrófono u otro dispositivo de entrada de sonido conectado al ordenador del usuario. La entrada de audio de un micrófono puede enrutarse a altavoces locales o enviarse a un servidor remoto. El objeto Microphone controla la ganancia, la velocidad de muestra y otras características de su propio flujo de sonido.
flash.media.AudioPlaybackMode	La clase AudioPlaybackMode define constantes para la propiedad `audioPlaybackMode` de la clase SoundMixer.

Cada sonido que se carga y se reproduce necesita su propia instancia de las clases Sound y SoundChannel. La clase SoundMixer global mezcla la salida de varias instancias de SoundChannel durante la reproducción.

Las clases Sound, SoundChannel y SoundMixer no se utilizan para datos de sonido obtenidos de un micrófono o de un servidor de flujo de medios como Flash Media Server.

Carga de archivos de sonido externos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cada instancia de la clase Sound existe para cargar y activar la reproducción de un recurso de sonido específico. Una aplicación no puede reutilizar un objeto Sound para cargar más de un sonido. Para cargar un nuevo recurso de sonido, debe crear otro objeto Sound.

Si se carga un archivo de sonido pequeño, como un sonido de clic que se asociará a un botón, la aplicación puede crear un nuevo objeto Sound y cargar automáticamente el archivo de sonido, tal como se muestra a continuación:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

El constructor `Sound()` acepta un objeto URLRequest como primer parámetro. Cuando se proporciona un valor del parámetro URLRequest, el nuevo objeto Sound empieza a cargar automáticamente el recurso de sonido especificado.

En todos los casos, excepto en los más sencillos, la aplicación debe prestar atención al progreso de carga del sonido y detectar los posibles errores. Por ejemplo, si el sonido del clic tiene un tamaño bastante grande, es posible que no esté completamente cargado cuando el usuario haga clic en el botón que activa dicho sonido. Si se intenta reproducir un sonido no cargado, puede producirse un error en tiempo de ejecución. Resulta más seguro esperar a que se cargue totalmente el sonido antes de permitir que los usuarios realicen acciones que puedan iniciar la reproducción de sonidos.

El objeto Sound distribuye varios eventos distintos durante el proceso de carga del sonido. La aplicación puede detectar estos eventos para hacer un seguimiento del progreso de carga y asegurarse de que el sonido se carga por completo antes de reproducirse. En la tabla siguiente se enumeran los eventos que puede distribuir un objeto Sound.

Evento	Descripción
open (`Event.OPEN`)	Se distribuye antes de que se inicie la operación de carga del sonido.
progress (`ProgressEvent.PROGRESS`)	Se distribuye periódicamente durante el proceso de carga del sonido cuando se reciben datos del archivo o del flujo.
id3 (`Event.ID3`)	Se distribuye cuando hay datos ID3 disponibles para un sonido MP3.
complete (`Event.COMPLETE`)	Se distribuye cuando se han cargado todos los datos del recurso de sonido.
ioError (`IOErrorEvent.IO_ERROR`)	Se distribuye cuando no se puede encontrar un archivo de sonido, o bien cuando el proceso de carga se interrumpe antes de que se puedan recibir todos los datos de sonido.

El código siguiente ilustra la manera de reproducir un sonido una vez cargado:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

En primer lugar, el código de ejemplo crea un nuevo objeto Sound sin asignarle un valor inicial para el parámetro URLRequest. A continuación, detecta el evento `Event.COMPLETE` del objeto Sound, que provoca la ejecución del método `onSoundLoaded()` cuando se cargan todos los datos de sonido. Después, llama al método `Sound.load()` con un nuevo valor URLRequest para el archivo de sonido.

El método `onSoundLoaded()` se ejecuta cuando se completa la carga de sonido. La propiedad `target` del objeto Event es una referencia al objeto Sound. La llamada al método `play()` del objeto Sound inicia la reproducción de sonido.

Supervisión del proceso de carga del sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los archivos de sonido pueden tener un tamaño muy grande y tardar mucho tiempo en cargarse. Aunque Flash Player y AIR permiten que la aplicación reproduzca sonidos incluso antes de que se carguen completamente, es posible que se desee proporcionar una indicación al usuario de la cantidad de datos de sonido que se han cargado, así como de la cantidad de sonido que ya se ha reproducido.

La clase Sound distribuye dos eventos que hacen relativamente fácil la visualización del proceso de carga de un sonido: `ProgressEvent.PROGRESS` y `Event.COMPLETE`. En el ejemplo siguiente se muestra cómo utilizar estos eventos para mostrar la información de progreso del sonido que se carga:

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

En este código se crea primero un objeto Sound y luego se añaden detectores a dicho objeto para los eventos `ProgressEvent.PROGRESS` y `Event.COMPLETE`. Una vez que el método `Sound.load()` se ha llamado y que el archivo de sonido recibe los primeros datos, se produce un evento `ProgressEvent.PROGRESS`, que activa el método `onSoundLoadProgress()`.

El porcentaje de datos de sonido cargados equivale al valor de la propiedad `bytesLoaded` del objeto ProgressEvent dividido entre el valor de la propiedad `bytesTotal`. Las mismas propiedades `bytesLoaded` y `bytesTotal` también están disponibles en el objeto Sound. El ejemplo anterior muestra mensajes relativos al progreso de carga del sonido, aunque se pueden utilizar fácilmente los valores `bytesLoaded` y `bytesTotal` para actualizar los componentes de barra de progreso, como los que se incluyen en la arquitectura Adobe Flex o en la herramienta de edición Adobe Flash.

En este ejemplo también se muestra la forma en que una aplicación puede reconocer un error y responder a él durante la carga de archivos de sonido. Por ejemplo, si no se puede encontrar un archivo de sonido con un nombre determinado, el objeto Sound distribuye un evento `Event.IO_ERROR`. En el código anterior, se ejecuta el método `onIOError()`, que muestra un breve mensaje de error cuando se produce un error.

Trabajo con sonidos incorporados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La utilización de sonidos incorporados, en lugar de cargar sonidos de un archivo externo, resulta útil para sonidos de pequeño tamaño que se emplean como indicadores en la interfaz de usuario de la aplicación (por ejemplo, los sonidos que se reproducen cuando se hace clic en los botones).

Si se incorpora un archivo de sonido en la aplicación, el tamaño del archivo SWF resultante aumenta con el tamaño del archivo de sonido. Es decir, la incorporación de archivos de sonido grandes en la aplicación puede hacer que el tamaño del archivo SWF sea excesivo.

El método preciso de incorporar un archivo de sonido en el archivo SWF de la aplicación varía de un entorno de desarrollo a otro.

Uso de un archivo de sonido incorporado en Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La herramienta de edición de Flash permite importar sonidos en varios formatos de sonido y almacenarlos como símbolos en la Biblioteca. Después se pueden asignar a fotogramas en la línea de tiempo o a los fotogramas de un estado de botón, utilizarlos con Comportamientos o utilizarlos directamente en código ActionScript. En esta sección se describe la manera de utilizar sonidos incorporados en código ActionScript con la herramienta de edición de Flash. Para más información sobre las otras maneras de utilizar sonidos incorporados en Flash, consulte "Importación de sonidos" en *Uso de Flash*.

Para incorporar un archivo de sonido utilizando la herramienta de edición de Flash:

1. Seleccione Archivo > Importar > Importar a biblioteca y, a continuación, elija un archivo de sonido para importarlo.
2. Haga clic con el botón derecho en el nombre del archivo importado en el panel Biblioteca y seleccione Propiedades. Active la casilla de verificación Exportar para ActionScript.
3. En el campo Clase, escriba el nombre que debe utilizarse al hacer referencia a este sonido incorporado en ActionScript. De manera predeterminada, utilizará el nombre del archivo de sonido de este campo. Si el nombre de archivo incluye un punto, como en "DrumSound.mp3", debe cambiarlo por otro como "DrumSound"; ActionScript no permite utilizar un punto en un nombre de clase. El campo Clase base debe mostrar flash.media.Sound.

4 Haga clic en OK (Aceptar). Es posible que vea un cuadro de mensaje que indique que no se encontró una definición para esta clase en la ruta de clases. Haga clic en Aceptar y continúe. Si se introduce una clase cuyo nombre no coincide con el de ninguna de las clases de la ruta de clases de la aplicación, se genera automáticamente una nueva clase que hereda de la clase flash.media.Sound.

5 Para utilizar el sonido incorporado hay que hacer referencia al nombre de clase del sonido en ActionScript. Por ejemplo, el código siguiente empieza creando una nueva instancia de la clase DrumSound generada automáticamente:

```
var drum:DrumSound = new DrumSound();
var channel:SoundChannel = drum.play();
```

DrumSound es una subclase de la clase flash.media.Sound, por lo que hereda los métodos y las propiedades de la clase Sound, incluido el método `play()` antes mostrado.

Uso de un archivo de sonido incorporado en Flex

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Hay muchas maneras de incorporar activos de sonido en una aplicación Flex, como:

- Utilizar la etiqueta de metadatos `[Embed]` en un script
- Usar la directiva `@Embed` en MXML para asignar un activo incorporado como una propiedad de un componente, como Button o SoundEffect.
- Utilizar la directiva `@Embed` dentro de un archivo CSS

Esta sección explica la primera opción: cómo incorporar código ActionScript en una aplicación Flex usando la etiqueta de metadatos `[Embed]`.

Para incorporar un activo en código ActionScript, utilice la etiqueta de metadatos `[Embed]`.

Coloque el archivo de sonido en la carpeta principal de código fuente o en otra carpeta situada en la ruta de compilación del proyecto. Cuando el compilador encuentra una etiqueta de metadatos Embed, crea automáticamente la clase de activo incorporado. Se puede acceder a la clase mediante una variable del tipo de datos Class declarada inmediatamente después de la etiqueta de metadatos `[Embed]`.

El código siguiente incorpora un sonido llamado smallSound.mp3 y utiliza una variable denominada `soundClass` para almacenar una referencia a la clase del activo incorporado asociado a ese sonido. A continuación, el código crea una instancia de la clase del activo incorporado, la convierte en una instancia de la clase Sound y llama al método `play()` de dicha instancia:

```
package
{
    import flash.display.Sprite;
    import flash.media.Sound;
    import flash.media.SoundChannel;

    public class EmbeddedSoundExample extends Sprite
    {
        [Embed(source="smallSound.mp3")]
        public var soundClass:Class;

        public function EmbeddedSoundExample()
        {
            var smallSound:Sound = new soundClass() as Sound;
            smallSound.play();
        }
    }
}
```

Para utilizar el sonido incorporado para establecer una propiedad de un componente Flex, debería convertirse como en instancia de la clase mx.core.SoundAsset en lugar de en una instancia de la clase Sound. Para ver un ejemplo parecido que utilice la clase SoundAsset, consulte Clases de activos incorporados.

Trabajo con archivos de flujo de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se reproduce un archivo de sonido o de vídeo mientras se cargan sus datos, se dice que se está *transmitiendo* el archivo. Los archivos de sonido externos que se cargan desde un servidor remoto suelen transmitirse; de este modo, no es necesario que el usuario espere hasta que se carguen todos los datos para poder escuchar el sonido.

La propiedad `SoundMixer.bufferTime` representa el número de milisegundos de datos de sonido que deben recopilar Flash Player o AIR antes de permitir la reproducción de sonido. Es decir, si la propiedad `bufferTime` se establece en 5000, Flash Player o AIR cargan al menos 5000 milisegundos (el equivalente en datos) del archivo de sonido antes de empezar a reproducir el sonido. El valor predeterminado de `SoundMixer.bufferTime` es 1000.

La aplicación puede sustituir el valor global de `SoundMixer.bufferTime` para un sonido determinado especificando explícitamente un nuevo valor de `bufferTime` al cargar el sonido. Para sustituir el tiempo de búfer predeterminado, primero hay que crear una nueva instancia de la clase SoundLoaderContext, establecer el valor de su propiedad `bufferTime` y pasarla como un parámetro al método `Sound.load()`, tal como se muestra a continuación:

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

Mientras sigue la reproducción, Flash Player y AIR intentan mantener el tamaño del búfer de sonido en un valor igual o mayor. Si los datos de sonido se cargan más rápido que la velocidad de reproducción, esta continuará sin interrupciones. Sin embargo, si la velocidad de carga de los datos disminuye debido a limitaciones de red, la cabeza lectora podría alcanzar el final del búfer de sonido. Si esto sucede, se suspende la reproducción, aunque se reanudará automáticamente en cuanto se hayan cargado más datos de sonido.

Para determinar si se ha suspendido la reproducción porque Flash o AIR están esperando a que se carguen más datos, se debe utilizar la propiedad `Sound.isBuffering`.

Trabajo con sonido generado dinámicamente

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Nota: *la posibilidad de generar audio dinámicamente está presente a partir de Flash Player 10 y Adobe AIR 1.5.*

En lugar de cargar o transmitir un sonido existente, se pueden generar dinámicamente los datos de audio. Puede generar datos de audio cuando se asigna un detector de eventos para el evento `sampleData` de un objeto Sound. (El evento `sampleData` se define en la clase SampleDataEvent en el paquete flash.events.) En este entorno, el objeto Sound no carga datos de sonido desde un archivo. Actúa como socket para datos de sonido transmitidos en flujo mediante la función asignada al evento.

Cuando se añade un detector de eventos `sampleData` a un objeto Sound, el objeto solicita datos periódicamente para agregarlos al búfer de sonido. Este búfer contiene datos para el objeto Sound que se va a reproducir. Cuando se llama al método `play()` del objeto Sound, se distribuye el evento `sampleData` al solicitar nuevos datos de sonido. (El valor es true solo cuando el objeto Sound no ha cargado datos mp3 desde un archivo.)

El objeto SampleDataEvent incluye una propiedad `data`. En el detector de eventos, los objetos ByteArray se escriben en este objeto`data`. Los conjuntos de bytes en los que se escribe este objeto se añaden a los datos de sonido almacenados en búfer que reproduce el objeto Sound. El conjunto de bytes en el búfer es un flujo de valores de coma flotante de -1 a 1. Cada valor de coma flotante representa al amplitud de un canal (izquierdo o derecho) de una muestra de sonido. El sonido se muestrea a 44.100 muestras por segundo. Cada muestra contiene un canal izquierdo y derecho, intercalada como datos de coma flotante en el conjunto de bytes.

En la función de controlador, se utiliza el método `ByteArray.writeFloat()` para escribir en la propiedad `data` del evento `sampleData`. Por ejemplo, el siguiente código genera una onda de seno:

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

Cuando se llama a `Sound.play()`, la aplicación inicia la llamada al controlador de eventos, solicitando los datos de la muestra de sonido. La aplicación continúa enviando eventos a medida que se reproduce el sonido hasta que se detiene el suministro o se llama a `SoundChannel.stop()`.

La latencia del evento varía en función de la plataforma y puede cambiar en futuras versiones de Flash Player y AIR. No dependa de una latencia concreta: calcúlela. Para calcular la latencia, utilice la siguiente fórmula:

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Proporcione de 2048 hasta 8192 muestras a la propiedad `data` del objeto SampleDataEvent (para cada llamada al controlador de eventos). Para obtener el mejor rendimiento, proporcione tantas muestras como sea posible (hasta 8192). Cuanto menor sea el número de muestras, más probable será la aparición de interferencias en la reproducción. Este comportamiento puede variar en función de la plataforma y según las situaciones; por ejemplo, al cambiar de tamaño la ventana del navegador. El código que funciona en una plataforma cuando solo se proporcionan 2048 muestras puede que no funcione tan bien cuando se ejecuta en una plataforma distinta. Si necesita la menor latencia posible, considere la posibilidad de que sea el usuario quien seleccione la cantidad de datos.

Si se proporcionan menos de 2048 muestras (por llamada al detector de eventos `sampleData`), la aplicación se detiene tras reproducir las muestras restantes. El objeto SoundChannel distribuye entonces un evento SoundComplete.

Modificación de sonido desde datos mp3

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

El método `Sound.extract()` se utiliza para extraer datos de un objeto Sound. Puede utilizar (y modificar) los datos para escribir en el flujo dinámico de otro objeto Sound para la reproducción. Por ejemplo, el siguiente código utiliza los bytes de archivo MP3 cargado y los transmite mediante una función de filtro, `upOctave()`:

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

Limitaciones en los sonidos generados

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Cuando se utiliza un detector de eventos `sampleData` con un objeto Sound, los únicos métodos de Sound que permanecen activados son`Sound.extract()` y `Sound.play()`. Llamar a otros métodos o propiedades genera una excepción. Todos los métodos y propiedades del objeto SoundChannel siguen activados.

Reproducción de sonidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para reproducir un sonido cargado solo hay que llamar al método `Sound.play()` de un objeto Sound:

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Al reproducir sonidos con ActionScript 3.0, se pueden realizar las operaciones siguientes:

- Reproducir un sonido desde una posición de inicio específica
- Pausar un sonido y reanudar la reproducción desde la misma posición posteriormente
- Saber exactamente cuándo termina de reproducirse un sonido
- Hacer un seguimiento del progreso de la reproducción de un sonido
- Cambiar el volumen o el desplazamiento cuando se reproduce un sonido

Para realizar estas operaciones durante la reproducción deben utilizarse las clases SoundChannel, SoundMixer y SoundTransform.

La clase SoundChannel controla la reproducción de un solo sonido. La propiedad `SoundChannel.position` puede considerarse como una cabeza lectora, que indica el punto actual en los datos de sonido que se están reproduciendo.

Cuando una aplicación llama al método `Sound.play()`, se crea una nueva instancia de la clase SoundChannel para controlar la reproducción.

La aplicación puede reproducir un sonido desde una posición de inicio específica, pasando dicha posición en milisegundos como el parámetro `startTime` del método `Sound.play()`. También puede especificar un número fijo de veces que se repetirá el sonido en una sucesión rápida pasando un valor numérico en el parámetro `loops` del método `Sound.play()`.

Cuando se llama al método `Sound.play()` con los parámetros `startTime` y `loops`, el sonido se reproduce de forma repetida desde el mismo punto de inicio cada vez, tal como se muestra en el código siguiente:

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

En este ejemplo, el sonido se reproduce desde un punto un segundo después del inicio del sonido, tres veces seguidas.

Pausa y reanudación de un sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si la aplicación reproduce sonidos largos, como canciones o emisiones podcast, es recomendable dejar que los usuarios pausen y reanuden la reproducción de dichos sonidos. Durante la reproducción en ActionScript un sonido no se puede pausar; solo se puede detener. Sin embargo, un sonido puede reproducirse desde cualquier punto. Se puede registrar la posición del sonido en el momento en que se detuvo y volver a reproducir el sonido desde dicha posición posteriormente.

Por ejemplo, supongamos que el código carga y reproduce un archivo de sonido como este:

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Mientras se reproduce el sonido, la propiedad `SoundChannel.position` indica el punto del archivo de sonido que se está reproduciendo en ese momento. La aplicación puede almacenar el valor de posición antes de detener la reproducción del sonido, como se indica a continuación:

```
var pausePosition:int = channel.position;
channel.stop();
```

Para reanudar la reproducción del sonido desde el mismo punto en que se detuvo, hay que pasar el valor de la posición almacenado anteriormente.

```
channel = snd.play(pausePosition);
```

Control de la reproducción

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Es posible que la aplicación desee saber cuándo se deja de reproducir un sonido para poder iniciar la reproducción de otro o para limpiar algunos recursos utilizados durante la reproducción anterior. La clase SoundChannel distribuye un evento `Event.SOUND_COMPLETE` cuando finaliza la reproducción de un sonido. La aplicación puede detectar este evento y adoptar las medidas oportunas como se muestra a continuación:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

La clase SoundChannel no distribuye eventos de progreso durante la reproducción. Para informar sobre el progreso de la reproducción, la aplicación puede configurar su propio mecanismo de sincronización y hacer un seguimiento de la posición de la cabeza lectora del sonido.

Para calcular el porcentaje de sonido que se ha reproducido, se puede dividir el valor de la propiedad `SoundChannel.position` entre la duración de los datos del sonido que se está reproduciendo:

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Sin embargo, este código solo indica porcentajes de reproducción precisos si los datos de sonido se han cargado completamente antes del inicio de la reproducción. La propiedad `Sound.length` muestra el tamaño de los datos de sonido que se cargan actualmente, no el tamaño definitivo de todo el archivo de sonido. Para hacer un seguimiento del progreso de reproducción de un flujo de sonido, la aplicación debe estimar el tamaño final de todo el archivo de sonido y utilizar dicho valor en sus cálculos. Se puede estimar la duración final de los datos de sonido mediante las propiedades `bytesLoaded` y `bytesTotal` del objeto Sound, de la manera siguiente:

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

El código siguiente carga un archivo de sonido más grande y utiliza el evento `Event.ENTER_FRAME` como mecanismo de sincronización para mostrar el progreso de la reproducción. Notifica periódicamente el porcentaje de reproducción, que se calcula como el valor de la posición actual dividido entre la duración total de los datos de sonido:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Una vez iniciada la carga de los datos de sonido, este código llama al método `snd.play()` y almacena el objeto SoundChannel resultante en la variable `channel`. A continuación, añade un detector de eventos a la aplicación principal para el evento `Event.ENTER_FRAME` y otro detector de eventos al objeto SoundChannel para el evento `Event.SOUND_COMPLETE` que se produce cuando finaliza la reproducción.

Cada vez que la aplicación alcanza un nuevo fotograma en su animación, se llama al método `onEnterFrame()`. El método `onEnterFrame()` estima la duración total del archivo de sonido según la cantidad de datos que ya se han cargado, y luego calcula y muestra el porcentaje de reproducción actual.

Cuando se ha reproducido todo el sonido, se ejecuta el método `onPlaybackComplete()` y se elimina el detector de eventos para el evento `Event.ENTER_FRAME`, de manera que no intente mostrar actualizaciones de progreso tras el fin de la reproducción.

El evento `Event.ENTER_FRAME` puede distribuirse muchas veces por segundo. En algunos casos no será necesario mostrar el progreso de la reproducción con tanta frecuencia. De ser así, la aplicación puede configurar su propio mecanismo de sincronización con la clase flash.util.Timer; véase "Trabajo con fechas y horas" en la página 1.

Detener flujos de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existe una anomalía en el proceso de reproducción para los sonidos que se están transmitiendo, es decir, para los sonidos que todavía se están cargando mientras se reproducen. Cuando la aplicación llama al método `SoundChannel.stop()` en una instancia de SoundChannel que reproduce un flujo de sonido, la reproducción se detiene en un fotograma y en el siguiente se reinicia desde el principio del sonido. Esto sucede porque el proceso de carga del sonido todavía está en curso. Para detener la carga y la reproducción de un flujo de sonido, llame al método `Sound.close()`.

Consideraciones de seguridad al cargar y reproducir sonidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La capacidad de la aplicación para acceder a datos de sonido puede limitarse según el modelo de seguridad de Flash Player o AIR. Cada sonido está sujeto a las restricciones de dos entornos limitados de seguridad diferentes, el del contenido en sí (el "entorno limitado de contenido") y el de la aplicación o el objeto que carga y reproduce el sonido (el "entorno limitado de propietario"). Para el contenido de la aplicación AIR en el entorno limitado de seguridad de la aplicación, todos los sonidos, incluidos los que se cargan desde otros dominios, están accesibles para el contenido en el entorno limitado de seguridad de la aplicación. Sin embargo, el contenido de otros entornos limitados de seguridad sigue las mismas reglas que el contenido ejecutado en Flash Player. Para más información sobre el modelo de seguridad de Flash Player en general y la definición de los entornos limitados, consulte "Seguridad" en la página 1085.

El entorno limitado de contenido controla si se pueden extraer datos detallados del sonido con la propiedad `id3` o el método `SoundMixer.computeSpectrum()`. No limita la carga o la reproducción del sonido en sí.

El dominio de origen del archivo de sonido define las limitaciones de seguridad del entorno limitado de contenido. Normalmente, si un archivo de sonido se encuentra en el mismo dominio o la misma carpeta que el archivo SWF de la aplicación o el objeto que lo carga, estos tendrán acceso total al archivo de sonido. Si el sonido procede de un dominio diferente al de la aplicación, aún es posible extraerlo al entorno limitado de contenido mediante un archivo de política.

La aplicación puede pasar un objeto SoundLoaderContext con una propiedad `checkPolicyFile` como parámetro al método `Sound.load()`. Si se establece la propiedad `checkPolicyFile` en `true`, se indica a Flash Player o AIR que busque un archivo de política en el servidor desde el que se carga el sonido. Si ya existe un archivo de política, y proporciona acceso al dominio del archivo SWF que se carga, el archivo SWF puede cargar el archivo de sonido, acceder a la propiedad `id3` del objeto Sound y llamar al método `SoundMixer.computeSpectrum()` para sonidos cargados.

El entorno limitado de propietario controla la reproducción local de los sonidos. La aplicación o el objeto que empieza a reproducir un sonido definen el entorno limitado de propietario.

El método `SoundMixer.stopAll()` detiene los sonidos de todos los objetos SoundChannel que se reproducen en ese momento, siempre y cuando cumplan los criterios siguientes:

- Los objetos del mismo entorno limitado de propietario han iniciado los sonidos.
- Los sonidos proceden de un origen con un archivo de política que proporciona acceso al dominio de la aplicación o del objeto que llama al método `SoundMixer.stopAll()`.

Sin embargo, en una aplicación de AIR, el contenido del entorno limitado de seguridad de la aplicación (contenido instalado con la aplicación AIR) no está limitado por estas restricciones de seguridad.

Para saber si el método `SoundMixer.stopAll()` detendrá realmente todos los sonidos en reproducción, la aplicación puede llamar al método `SoundMixer.areSoundsInaccessible()`. Si el método devuelve un valor `true`, algunos de los sonidos que se reproducen se encuentran fuera del control del entorno limitado de propietario actual y no se detendrán con el método `SoundMixer.stopAll()`.

El método `SoundMixer.stopAll()` también evita que la cabeza lectora continúe para todos los sonidos cargados desde archivos externos. Sin embargo, los sonidos que se incorporan en archivos FLA y se asocian a los fotogramas de la línea de tiempo con la herramienta de edición de Flash podrían volver a reproducirse si la animación se desplaza a un nuevo fotograma.

Control de desplazamiento y volumen de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El objeto SoundChannel controla los canales estéreo izquierdo y derecho de un sonido. Si el sonido MP3 es monoaural, los canales estéreo izquierdo y derecho del objeto SoundChannel incluirán formas de onda idénticas.

Se puede conocer la amplitud de cada canal estéreo del objeto que se reproduce mediante las propiedades `leftPeak` y `rightPeak` del objeto SoundChannel. Estas propiedades muestran la amplitud máxima de la forma de onda del sonido; no representan el volumen real de la reproducción. El volumen real de la reproducción es una función de la amplitud de la onda de sonido y de los valores de volumen establecidos en el objeto SoundChannel y la clase SoundMixer.

La propiedad pan (desplazamiento) de un objeto SoundChannel puede utilizarse para especificar un nivel de volumen diferente para cada uno de los canales izquierdo y derecho durante la reproducción. Esta propiedad puede tener un valor que oscila entre -1 y 1, donde -1 significa que el canal izquierdo se reproduce al máximo volumen mientras que el canal derecho está en silencio y 1 significa que el canal derecho se reproduce al máximo volumen mientras que el canal izquierdo está en silencio. Los valores numéricos entre -1 y 1 establecen valores proporcionales para los valores de canal izquierdo y derecho, y un valor 0 implica que los dos canales se reproducen a un volumen medio y equilibrado.

En el ejemplo de código siguiente se crea un objeto SoundTransform con un valor de volumen de 0,6 y un valor de desplazamiento de -1 (volumen máximo en el canal izquierdo y sin volumen en el canal derecho). Pasa el objeto SoundTransform como parámetro al método `play()`, que aplica dicho objeto al nuevo objeto SoundChannel que se crea para controlar la reproducción.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

Se puede modificar el volumen y el desplazamiento mientras se reproduce un sonido estableciendo las propiedades `pan` o `volume` de un objeto SoundTransform y aplicando dicho objeto como propiedad `soundTransform` de un objeto SoundChannel.

También se pueden establecer simultáneamente valores globales de volumen y desplazamiento con la propiedad `soundTransform` de la clase SoundMixer, como se muestra en el ejemplo siguiente:

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Además, se puede utilizar un objeto SoundTransform para establecer los valores de volumen y desplazamiento de un objeto Microphone (véase "Captura de entradas de sonido" en la página 481) y de los objetos Sprite y SimpleButton.

En el ejemplo siguiente se alterna el desplazamiento del sonido del canal izquierdo al derecho, y a la inversa, mientras se reproduce el sonido.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Este código se inicia cargando un archivo de sonido y creando un nuevo objeto SoundTransform con el volumen establecido en 1 (volumen máximo) y el desplazamiento establecido en 0 (equilibrio entre los canales izquierdo y derecho). A continuación, llama al método `snd.play()` y pasa el objeto SoundTransform como parámetro.

Mientras se reproduce el sonido, el método `onEnterFrame()` se ejecuta repetidamente. El método `onEnterFrame()` utiliza la función `Math.sin()` para generar un valor entre -1 y 1, rango que corresponde a los valores aceptables de la propiedad `SoundTransform.pan`. La propiedad `pan` del objeto SoundTransform se establece en el nuevo valor y, posteriormente, se establece la propiedad `soundTransform` del canal para utilizar el objeto SoundTransform alterado.

Para ejecutar este ejemplo, reemplace el nombre de archivo bigSound.mp3 con el nombre de un archivo MP3 local. A continuación, ejecute el ejemplo. Escuchará el aumento del volumen del canal izquierdo mientras disminuye el volumen del canal derecho, y viceversa.

En este ejemplo podría lograrse el mismo efecto estableciendo la propiedad `soundTransform` de la clase SoundMixer. Sin embargo, se vería afectado el desplazamiento de todos los sonidos en reproducción, no solo el sonido que reproduce este objeto SoundChannel.

Trabajo con metadatos de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los archivos de sonido que utilizan el formato MP3 pueden contener datos adicionales sobre el sonido en forma de etiquetas ID3.

No todos los archivos MP3 contienen metadatos ID3. Cuando un objeto Sound carga un archivo de sonido MP3, distribuye un evento `Event.ID3` si el archivo de sonido incluye metadatos ID3. Para evitar errores en tiempo de ejecución, la aplicación debe esperar a recibir el evento `Event.ID3` antes de acceder a la propiedad `Sound.id3` de un sonido cargado.

En el código siguiente se muestra la manera de detectar que se han cargado los metadatos ID3 para un archivo de sonido.

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

Este código empieza por crear un objeto Sound e indicarle que detecte el evento `Event.ID3`. Cuando se cargan los metadatos ID3 del archivo de sonido, se llama al método `onID3InfoReceived()`. El destino del objeto Event que se pasa al método `onID3InfoReceived()` es el objeto Sound original, de modo que el método obtiene la propiedad `id3` del objeto Sound y luego recorre todas sus propiedades con nombre para trazar sus valores.

Acceso a datos de sonido sin formato

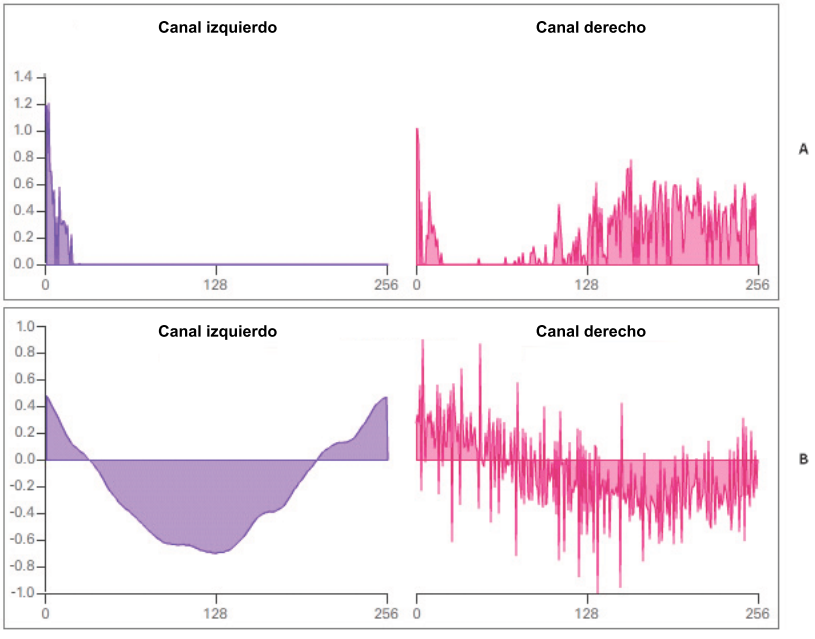
Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `SoundMixer.computeSpectrum()` permite que una aplicación lea los datos de sonido sin formato para la forma de onda que se reproduce en ese momento. Si se reproduce más de un objeto SoundChannel, el método `SoundMixer.computeSpectrum()` muestra los datos de sonido combinados de cada objeto SoundChannel mezclado.

Los datos de sonido se devuelven como un objeto ByteArray con 512 bytes de datos, cada uno de los cuales contiene un valor de coma flotante que oscila entre -1 y 1. Estos valores representan la amplitud de los puntos de la forma de onda del sonido en reproducción. Los valores se transmiten en dos grupos de 256; el primer grupo para el canal estéreo izquierdo y el segundo para el canal estéreo derecho.

El método `SoundMixer.computeSpectrum()` devolverá datos del espectro de frecuencias en lugar de datos de forma de onda si el parámetro `FFTMode` se establece en `true`. El espectro de frecuencias muestra la amplitud clasificada por la frecuencia de sonido, de menor a mayor. El algoritmo FFT (Fast Fourier Transform) se utiliza para convertir los datos de forma de onda en datos del espectro de frecuencias. Los valores resultantes del espectro de frecuencias oscilan entre 0 y 1,414 aproximadamente (la raíz cuadrada de 2).

En el siguiente diagrama se comparan los datos devueltos por el método `computeSpectrum()` cuando el parámetro `FFTMode` se establece en `true` y cuando se establece en `false`. El sonido cuyos datos se han utilizado para este diagrama contiene un sonido alto de contrabajo en el canal izquierdo y un sonido de batería en el canal derecho.



Valores devueltos por el método SoundMixer.computeSpectrum()
A. *fftMode=true* ***B.*** *fftMode=false*

El método `computeSpectrum()` también puede devolver datos que se han vuelto a muestrear a una velocidad inferior. Normalmente, esto genera datos de forma de onda o de frecuencia más suaves a expensas del detalle. El parámetro `stretchFactor` controla la velocidad a la que se muestrea el método `computeSpectrum()`. Cuando el parámetro `stretchFactor` se establece en 0 (valor predeterminado), los datos de sonido se muestrean a una velocidad de 44,1 kHz. La velocidad se reduce a la mitad en los valores sucesivos del parámetro stretchFactor, de manera que un valor de 1 especifica una velocidad de 22,05 kHz, un valor de 2 especifica una velocidad de 11,025 kHz, etc. El método `computeSpectrum()` aún devuelve 256 bytes en cada canal estéreo cuando se utiliza un valor `stretchFactor` más alto.

El método `SoundMixer.computeSpectrum()` tiene algunas limitaciones:

- Puesto que los datos de sonido de un micrófono o de los flujos RTMP no pasan por el objeto SoundMixer global, el método `SoundMixer.computeSpectrum()` no devolverá datos de dichas fuentes.

- Si uno o varios sonidos que se reproducen proceden de orígenes situados fuera del entorno limitado de contenido actual, las restricciones de seguridad harán que el método `SoundMixer.computeSpectrum()` genere un error. Para obtener más detalles sobre las limitaciones de seguridad del método `SoundMixer.computeSpectrum()`, consulte "Consideraciones de seguridad al cargar y reproducir sonidos" en la página 474 y "Acceso a medios cargados como datos" en la página 1111.

Sin embargo, en una aplicación de AIR, el contenido del entorno limitado de seguridad de la aplicación (contenido instalado con la aplicación AIR) no está limitado por estas restricciones de seguridad.

Creación de un visualizador de sonido sencillo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

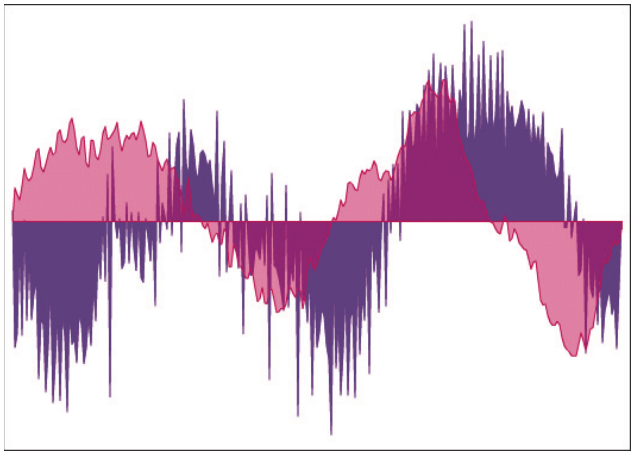
En el ejemplo siguiente se utiliza el método `SoundMixer.computeSpectrum()` para mostrar un gráfico de la forma de onda que se anima con cada fotograma:

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
```

```
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

En primer lugar, este ejemplo carga y reproduce un archivo de sonido, y luego detecta el evento `Event.ENTER_FRAME` que activará el método `onEnterFrame()` mientras se reproduce el sonido. El método `onEnterFrame()` se inicia llamando al método `SoundMixer.computeSpectrum()`, que almacena los datos de forma de onda del sonido en el objeto ByteArray `bytes`.

La forma de onda del sonido se representa con la API de dibujo vectorial. El bucle `for` recorre los primeros 256 valores de datos; representa el canal estéreo izquierdo y dibuja una línea desde un punto al siguiente con el método `Graphics.lineTo()`. Un segundo bucle `for` recorre el siguiente conjunto de 256 valores y los representa en orden inverso esta vez, de derecha a izquierda. Las representaciones de la forma de onda resultante pueden generar un interesante efecto de reflejo de imagen, tal como se muestra en la imagen siguiente.



Captura de entradas de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Microphone permite a la aplicación conectar un micrófono u otro dispositivo de entrada de sonido en el sistema del usuario, y difundir el audio de la entrada a los altavoces, o bien enviar los datos de audio a un servidor remoto, como Flash Media Server. Es posible acceder a los datos de audio sin procesar desde el micrófono y grabar o procesarlos; también se puede enviar el audio directamente a los altavoces del sistema o enviar datos de audio comprimidos a un servidor remoto. Se puede emplear el códec Speex o Nellymoser para los datos enviados a un servidor remoto. (El códec Speex se admite a partir de Flash Player 10 y Adobe AIR 1.5.)

Más temas de ayuda

Michael Chaize: AIR, Android y el micrófono

Christophe Coenraets: Notas de voz para Android

Acceso a un micrófono

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Microphone no tiene un método constructor. En su lugar, se utiliza el método `Microphone.getMicrophone()` estático para obtener una nueva instancia de Microphone, tal como se muestra a continuación.

```
var mic:Microphone = Microphone.getMicrophone();
```

Si se llama al método `Microphone.getMicrophone()` sin un parámetro, se devuelve el primer dispositivo de entrada de sonido detectado en el sistema del usuario.

Un sistema puede tener más de un dispositivo de entrada de sonido conectado. La aplicación puede utilizar la propiedad `Microphone.names` para obtener un conjunto de los nombres de todos los dispositivos de entrada de sonido disponibles. A continuación, puede llamar al método `Microphone.getMicrophone()` con un parámetro `index` que coincide con el valor de índice del nombre de un dispositivo del conjunto.

Es posible que el sistema no tenga conectado un micrófono u otro dispositivo de entrada de sonido. Se puede utilizar la propiedad `Microphone.names` o el método `Microphone.getMicrophone()` para comprobar si el usuario tiene un dispositivo de entrada de sonido instalado. Si el usuario no tiene un dispositivo de entrada de sonido instalado, la longitud del conjunto `names` es cero y el método `getMicrophone()` devuelve un valor `null`.

Cuando la aplicación llama al método `Microphone.getMicrophone()`, Flash Player muestra el cuadro de diálogo Configuración de Flash Player, que pregunta al usuario si desea que Flash Player acceda a la cámara y al micrófono del sistema. Una vez que el usuario hace clic en el botón Allow (Permitir) o en el botón Deny (Denegar) de este cuadro de diálogo, se distribuye un objeto StatusEvent. La propiedad `code` de la instancia de StatusEvent indica si se ha permitido o denegado el acceso al micrófono, tal como se muestra en este ejemplo:

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

La propiedad `StatusEvent.code` contiene "Microphone.Unmuted" si se ha permitido el acceso, o bien "Microphone.Muted" si se ha denegado.

la propiedad `Microphone.muted` se establece en `true` o `false` cuando el usuario permite o deniega el acceso al micrófono, respectivamente. Sin embargo, la propiedad `muted` no se establece en la instancia de Microphone hasta que se haya distribuido el objeto StatusEvent, de manera que la aplicación también debe esperar a que el evento `StatusEvent.STATUS` se distribuya antes de comprobar la propiedad `Microphone.muted`.

Para que Flash Player pueda mostrar el cuadro de diálogo de configuración, la aplicación se debe ser lo suficientemente amplia para mostrarlo (215 por 138 píxeles como mínimo). De lo contrario, el acceso se deniega automáticamente.

El contenido que se ejecuta en el entorno limitado de la aplicación de AIR no necesita el permiso de otro usuario para acceder al micrófono. De este modo, los eventos status para silenciar y quitar el silencio del micrófono nunca se distribuyen. El contenido que se ejecuta en AIR fuera del entorno limitado de la aplicación requiere permiso del usuario, por lo que estos eventos de estado se pueden distribuir.

Enrutamiento de audio de micrófono a altavoces locales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La entrada de audio de un micrófono puede enrutarse a los altavoces del sistema locales llamando al método `Microphone.setLoopback()` con un valor de parámetro `true`.

Cuando el sonido de un micrófono local se enruta a los altavoces locales, existe el riesgo de crear un bucle de acoplamiento de audio, que puede provocar sonidos estridentes e incluso dañar el hardware de sonido. Si se llama al método `Microphone.setUseEchoSuppression()` con un valor de parámetro `true`, se reduce (sin eliminarse por completo) el riesgo de que se produzca dicho acoplamiento de audio. Adobe recomienda llamar siempre a `Microphone.setUseEchoSuppression(true)` antes de llamar a `Microphone.setLoopback(true)`, a menos que se sepa con certeza que el usuario reproduce el sonido con auriculares u otro dispositivo distinto de los altavoces.

En el código siguiente se muestra cómo enrutar el audio de un micrófono local a los altavoces del sistema locales:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Alteración del audio de micrófono

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación puede alterar los datos de audio que proceden de un micrófono de dos maneras. Primero, puede cambiar la ganancia del sonido de la entrada, que multiplica de forma efectiva los valores de entrada por una cantidad especificada para crear un sonido más alto o más bajo. La propiedad `Microphone.gain` acepta valores numéricos entre 0 y 100 (ambos incluidos). Un valor de 50 funciona como un multiplicador de 1 y especifica un volumen normal. Un valor de 0 funciona como un multiplicador de 0 y silencia de forma efectiva el audio de la entrada. Los valores superiores a 50 especifican un volumen por encima del normal.

La aplicación también puede cambiar la frecuencia de muestreo del audio de la entrada. Las frecuencias de muestreo más altas aumentan la calidad del sonido, pero crean flujos de datos más densos que utilizan más recursos para la transmisión y el almacenamiento. La propiedad `Microphone.rate` representa la frecuencia de muestreo de audio calculada en kilohercios (kHz). La frecuencia de muestreo predeterminada es de 8 kHz. La propiedad `Microphone.rate` puede establecerse en un valor mayor que 8 kHz si el micrófono admite una frecuencia superior. Por ejemplo, si se establece la propiedad `Microphone.rate` en un valor de 11, se establece la velocidad de muestreo en 11 kHz; si se establece en 22, se establece la velocidad de muestreo en 22 kHz, y así sucesivamente. Las frecuencias de muestreo disponibles dependen del códec seleccionado. Cuando se usa el códec Nellymoser, puede especificar 5, 8, 11, 16, 22 y 44 kHz como frecuencia de muestreo. Si se utiliza el códec Speex (disponible a partir de Flash Player 10 y Adobe AIR 1.5), solo es posible utilizar frecuencias de 16 kHz.

Detección de actividad del micrófono

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para conservar los recursos de procesamiento y ancho de banda, Flash Player intenta detectar el momento en que el micrófono no transmite ningún sonido. Cuando el nivel de actividad del micrófono se mantiene por debajo del umbral de nivel de silencio durante un periodo de tiempo, Flash Player deja de transmitir la entrada de audio y distribuye un objeto ActivityEvent en su lugar. Si se utiliza el códec Speex (disponible en Flash Player 10 y en Adobe AIR 1.5 o versiones posteriores), se debe establecer el nivel de silencio en 0 para garantizar que la aplicación transmita datos de audio de forma continuada. La detección de actividad de voz de Speex reduce automáticamente el ancho de banda.

Nota: *un objeto Microphone solo distribuye eventos Activity cuando la aplicación controla el micrófono. De este modo, si no se llama a `setLoopBack( true )`, añada un detector de eventos de datos de muestra, o bien, añada el micrófono a un objeto NetStream y no se distribuirán eventos de actividad.*

La clase Microphone tiene tres propiedades para supervisar y controlar la detección de actividad:

- La propiedad de solo lectura `activityLevel` indica la cantidad de sonido que detecta el micrófono, en una escala de 0 a 100.
- La propiedad `silenceLevel` especifica la cantidad de sonido necesaria para activar el micrófono y distribuir un evento `ActivityEvent.ACTIVITY`. La propiedad `silenceLevel` también utiliza una escala de 0 a 100, y el valor predeterminado es 10.
- La propiedad `silenceTimeout` describe el número de milisegundos durante los cuales el nivel de actividad debe permanecer por debajo del nivel de silencio, hasta que se distribuye un evento `ActivityEvent.ACTIVITY` para indicar que el micrófono ya está en silencio. El valor predeterminado de `silenceTimeout` es 2000.

Las propiedades `Microphone.silenceLevel` y `Microphone.silenceTimeout` son de solo lectura, pero se puede utilizar el método `Microphone.setSilenceLevel()` para cambiar sus valores.

En algunos casos, el proceso de activación del micrófono cuando se detecta una nueva actividad puede provocar una breve demora. Esas demoras de activación pueden eliminarse manteniendo el micrófono activo todo el tiempo. La aplicación puede llamar al método `Microphone.setSilenceLevel()` con el parámetro `silenceLevel` establecido en cero para indicar a Flash Player que mantenga el micrófono activo y siga recopilando datos de audio, incluso cuando no se detecten sonidos. Y a la inversa, si se establece el parámetro `silenceLevel` en 100, se evita que el micrófono esté siempre activado.

El ejemplo siguiente muestra información sobre el micrófono e informa de los eventos de actividad y de estado que distribuye un objeto Microphone:

```
import flash.events.ActivityEvent;
import flash.events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

Cuando ejecute el ejemplo anterior, hable al micrófono del sistema o haga ruido, y verá que aparecen las sentencias trace resultantes en una consola o una ventana de depuración.

Intercambio de audio con un servidor multimedia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al utilizar ActionScript con un servidor de flujo de medios como Flash Media Server estarán disponibles otras funciones de audio.

En concreto, la aplicación puede asociar un objeto Microphone a un objeto NetStream y transmitir datos directamente desde el micrófono del usuario al servidor. Los datos de audio también se pueden transmitir desde el servidor a una aplicación creada con Flash o Flex y reproducirse como parte de un objeto MovieClip, o bien mediante un objeto Video.

El códec Speex está disponible a partir de Flash Player 10 y Adobe AIR 1.5. Para definir el códec utilizado para audio comprimido enviado al servidor de medios, se debe establecer la propiedad `codec` del objeto Microphone. Esta propiedad puede tener dos valores, que se enumeran en la clase SoundCodec. Con la definición de la propiedad codec como `SoundCodec.SPEEX`, se selecciona el códec Speex para comprimir audio. Con el establecimiento de la propiedad en `SoundCodec.NELLYMOSER` (valor predeterminado), se selecciona el códec Nellymoser para comprimir audio.

Para obtener más información, consulte la documentación de Flash Media Server en línea en www.adobe.com/go/learn_fms_docs_es.

Captura de datos de sonido del micrófono

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

En Flash Player 10.1. y AIR 2, o posterior, es posible capturar datos a partir de los datos del micrófono como conjunto de bytes de valores de coma flotante. Cada valor representa un ejemplo de datos de audio monofónicos.

Para obtener los datos del micrófono, establezca un detector de eventos para el evento `sampleData` del objeto Microphone. El objeto Microphone distribuye eventos `sampleData` de forma periódica conforme el búfer del micrófono se llena con muestras de sonido. El objeto SampleDataEvent dispone de la propiedad `data`, que es un conjunto de bytes de muestras de sonido. Las muestras se representan como valores de coma flotante y cada una de ellas representa una muestra de sonido monofónico.

El siguiente código captura los datos de sonido del micrófono en un objeto ByteArray denominado `soundBytes`:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
function micSampleDataHandler(event:SampleDataEvent):void {
    while(event.data.bytesAvailable) {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
```

Es posible reutilizar los bytes de la muestra como audio de reproducción para un objeto Sound. Si es así, la propiedad `rate` del objeto Microphone se debe establecer en 44, que es la frecuencia de muestreo que utilizan los objetos Sound. (También se pueden convertir las muestras del micrófono capturadas con una frecuencia más baja a una frecuencia de 44 kHz necesaria para el objeto Sound.) Asimismo, se debe tener en cuenta que el objeto Microphone captura muestras monofónicas, mientras que el objeto Sound emplea sonido estéreo; por lo tanto, se debe escribir cada uno de los bytes capturados por el objeto Microphone en el objeto Sound dos veces. En el siguiente ejemplo se capturan 4 segundos de datos del micrófono y se reproducen utilizando un objeto Sound:

```
const DELAY_LENGTH:int = 4000;
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.gain = 100;
mic.rate = 44;
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);

var timer:Timer = new Timer(DELAY_LENGTH);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function micSampleDataHandler(event:SampleDataEvent):void
{
    while(event.data.bytesAvailable)
    {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
var sound:Sound = new Sound();
var channel:SoundChannel;
function timerHandler(event:TimerEvent):void
{
    mic.removeEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
    timer.stop();
    soundBytes.position = 0;
    sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playbackSampleHandler);
    channel.addEventListener( Event.SOUND_COMPLETE, playbackComplete );
    channel = sound.play();
}

function playbackSampleHandler(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192 && soundBytes.bytesAvailable > 0; i++)
    {
        trace(sample);
        var sample:Number = soundBytes.readFloat();
        event.data.writeFloat(sample);
        event.data.writeFloat(sample);
    }
}

function playbackComplete( event:Event ):void
{
    trace( "Playback finished.");
}
```

Para obtener más información sobre la reproducción de sonidos a partir de los datos de muestra de sonido, consulte “Trabajo con sonido generado dinámicamente” en la página 469.

Ejemplo de sonido: Podcast Player

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una emisión podcast es un archivo de sonido que se distribuye por Internet, bajo demanda o por suscripción. Las emisiones podcast suelen publicarse como parte de una serie (denominada "canal podcast"). Puesto que los episodios de emisiones podcast pueden durar entre un minuto y muchas horas, normalmente se transmiten mientras se reproducen. Los episodios de emisiones podcast, también denominados "elementos", se suelen transmitir en formato MP3. Las emisiones podcast de vídeo gozan de una gran popularidad, pero esta aplicación de ejemplo solo reproduce emisiones podcast de audio que utilizan archivos MP3.

Este ejemplo no constituye un agregador de emisiones podcast completo. Por ejemplo, no gestiona suscripciones a emisiones podcast específicas ni recuerda las emisiones podcast que ha escuchado el usuario la próxima vez que se ejecuta la aplicación. Podría servir de punto de partida para un agregador de emisiones podcast más completo.

El ejemplo de Podcast Player ilustra las siguientes técnicas de programación con ActionScript:

- Leer un fuente RSS externa y analizar su contenido XML
- Crear una clase SoundFacade para simplificar la carga y la reproducción de archivos de sonido
- Mostrar el progreso de la reproducción de sonido
- Pausar y reanudar la reproducción de sonido

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Podcast Player se encuentran en la carpeta Samples/PodcastPlayer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
PodcastPlayer.mxml o PodcastPlayer.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	Clase de documento que contiene la lógica de la interfaz de usuario para el reproductor de podcast (solo Flash).
SoundPlayer.mxml	Componente MXML que muestra botones de reproducción y barras de progreso, y controla la reproducción del sonido, solo para Flex.
main.css	Estilos de la interfaz de usuario de la aplicación (solo Flex).
images/	Iconos de aplicación de estilos a los botones (solo Flex).
comp/example/programmingas3/podcastplayer/SoundPlayer.as	Clase del símbolo de clip de película SoundPlayer que contiene la lógica de la interfaz de usuario para el reproductor de sonidos (solo Flash).
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	Procesador de celdas personalizado para visualizar un botón de reproducción en una celda de cuadrícula de datos (solo Flash).
com/example/programmingas3/podcastplayer/RSSBase.as	Una clase base que proporciona propiedades y métodos comunes para las clases RSSChannel y RSSItem.

Archivo	Descripción
com/example/programmingas3/podcastplayer/RSSChannel.as	Una clase ActionScript que contiene datos sobre un canal RSS.
com/example/programmingas3/podcastplayer/RSSItem.as	Una clase ActionScript que contiene datos sobre un elemento RSS.
com/example/programmingas3/podcastplayer/SoundFacade.as	La clase ActionScript principal de la aplicación. Reúne los métodos y los eventos de las clases Sound y SoundChannel, y añade las funciones de pausa y reanudación de reproducción.
com/example/programmingas3/podcastplayer/URLService.as	Una clase ActionScript que recupera datos de un URL remoto.
playerconfig.xml	Un archivo XML que contiene una lista de las fuentes RSS que representan canales podcast.
comp/example/programmingas3/utils/DateUtil.as	Clase que se utiliza para aplicar formato de fechas fácilmente (solo Flash).

Lectura de datos RSS para un canal podcast

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación Podcast Player empieza leyendo información sobre varios canales podcast y sus episodios:

1. En primer lugar, la aplicación lee un archivo de configuración XML que contiene una lista de canales podcast y muestra dicha lista al usuario.

2. Cuando el usuario selecciona uno de los canales podcast, lee la fuente RSS del canal y muestra una lista de los episodios del canal.

Este ejemplo utiliza la clase de utilidad URLLoader para recuperar datos basados en texto desde una ubicación remota o un archivo local. Podcast Player crea primero un objeto URLLoader para obtener una lista de fuentes RSS en formato XML del archivo playerconfig.xml. A continuación, cuando el usuario selecciona una fuente específica de la lista, se crea un nuevo objeto URLLoader para leer datos RSS del URL de dicha fuente.

Simplificación de la carga y la reproducción de sonido mediante la clase SoundFacade

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La arquitectura de sonido ActionScript 3.0 es eficaz, pero compleja. Las aplicaciones que solo necesitan funciones básicas de carga y reproducción de sonido pueden utilizar una clase que oculta parte de la complejidad; dicha clase proporciona un conjunto más sencillo de llamadas de método y eventos. En el mundo de los patrones de diseño de software, esta clase se denomina *facade*.

La clase SoundFacade presenta una interfaz única para realizar las tareas siguientes:

- Cargar archivos de sonido con un objeto Sound, un objeto SoundLoaderContext y la clase SoundMixer
- Reproducir archivos de sonido con los objetos Sound y SoundChannel
- Distribuir eventos de progreso de reproducción
- Pausar y reanudar la reproducción del sonido mediante los objetos Sound y SoundChannel

La clase SoundFacade intenta ofrecer la mayor parte de las funciones de las clase de sonido ActionScript, pero con una menor complejidad.

En el código siguiente se muestra la declaración de clase, las propiedades de clase y el método constructor `SoundFacade()`:

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;

    public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                                autoPlay:Boolean = true, streaming:Boolean = true,
                                bufferTime:int = -1):void
    {
        this.url = soundUrl;

        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

La clase SoundFacade amplía la clase EventDispatcher para que pueda distribuir sus propios eventos. El código de clase declara primero las propiedades de un objeto Sound y un objeto SoundChannel. La clase también almacena el valor del URL del archivo de sonido y una propiedad `bufferTime` que se utilizará al transmitir el sonido. Asimismo, acepta algunos valores de parámetro booleanos que afectan al comportamiento de carga y reproducción.

- El parámetro `autoLoad` indica al objeto que la carga de sonido debe empezar en cuanto se cree este objeto.
- El parámetro `autoPlay` indica que la reproducción de sonido debe empezar en cuanto se hayan cargado suficientes datos de sonido. Si se trata de un flujo de sonido, la reproducción empezará tan pronto como se hayan cargado suficientes datos, según especifica la propiedad `bufferTime`.
- El parámetro `streaming` indica que este archivo de sonido puede empezar a reproducirse antes de que haya finalizado la carga.

El parámetro `bufferTime` se establece de forma predeterminada en un valor de -1. Si el método constructor detecta un valor negativo en el parámetro `bufferTime`, establece la propiedad `bufferTime` en el valor de `SoundMixer.bufferTime`. Esto permite que la aplicación se establezca de forma predeterminada en el valor `SoundMixer.bufferTime` global, según se desee.

Si el parámetro `autoLoad` se establece en `true`, el método constructor llama inmediatamente al siguiente método `load()` para iniciar la carga del archivo de sonido:

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

El método `load()` crea un nuevo objeto Sound y añade detectores para todos los eventos de sonido importantes. A continuación, indica al objeto Sound que cargue el archivo de sonido, mediante un objeto SoundLoaderContext para pasar el valor `bufferTime`.

Puesto que se puede cambiar la propiedad `url`, se puede usar una instancia SoundFacade para reproducir distintos archivos de sonido en secuencia: solo hay que cambiar la propiedad `url` y llamar al método `load()` para que se cargue el nuevo archivo de sonido.

Los tres siguientes métodos de detección de eventos muestran la forma en que el objeto SoundFacade hace un seguimiento del progreso de carga y decide el momento en que se inicia la reproducción del sonido:

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

El método `onLoadOpen()` se ejecuta cuando se inicia la carga de sonido. Si se puede reproducir el sonido en el modo de transmisión, el método `onLoadComplete()` establece el indicador `isReadyToPlay` en `true` inmediatamente. El indicador `isReadyToPlay` determina si la aplicación puede empezar a reproducir el sonido, tal vez como respuesta a una acción de usuario, como hacer clic en un botón Reproducir. La clase SoundChannel administra el búfer de los datos de sonido, de modo que no es necesario comprobar explícitamente si se han cargado suficientes datos antes de llamar al método `play()`.

El método `onLoadProgress()` se ejecuta periódicamente durante el proceso de carga. Distribuye un clon de su objeto ProgressEvent que utilizará el código que usa este objeto SoundFacade.

Cuando los datos de sonido se han cargado completamente, se ejecuta el método `onLoadComplete()` y se llama al método `play()` para sonidos que no sean de una transmisión de flujo si es necesario. El método `play()` se muestra a continuación.

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

El método `play()` llama al método `Sound.play()` si el sonido está listo para reproducirse. El objeto SoundChannel resultante se almacena en la propiedad `sc`. El método `play()` crea un objeto Timer que se utilizará para distribuir eventos de progreso de reproducción a intervalos regulares.

Visualización del progreso de reproducción

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Crear un objeto Timer para controlar la reproducción es una operación compleja que solo se debería tener que programar una sola vez. La encapsulación de esta lógica Timer en una clase reutilizable (como la clase SoundFacade) permite que las aplicaciones detecten las mismas clases de eventos de progreso cuando se carga y se reproduce un sonido.

El objeto Timer que crea el método `SoundFacade.play()` distribuye una instancia de TimerEvent cada segundo. El siguiente método `onPlayTimer()` se ejecuta cuando llega una nueva clase TimerEvent.

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

El método `onPlayTimer()` implementa la técnica de estimación de tamaño descrita en la sección“Control de la reproducción” en la página 472. Luego crea una nueva instancia de ProgressEvent con un tipo de evento de `SoundFacade.PLAY_PROGRESS`, con la propiedad `bytesLoaded` establecida en la posición actual del objeto SoundChannel y la propiedad `bytesTotal` establecida en la duración estimada de los datos de sonido.

Pausa y reanudación de la reproducción

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `SoundFacade.play()` mostrado anteriormente acepta un parámetro `pos` correspondiente a un punto inicial de los datos de sonido. Si el valor `pos` es cero, el sonido empieza a reproducirse desde el principio.

El método `SoundFacade.stop()` también acepta un parámetro `pos`, tal como se muestra aquí:

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

Cuando se llama al método `SoundFacade.stop()`, este establece la propiedad `pausePosition` para que la aplicación sepa dónde colocar la cabeza lectora si el usuario desea reanudar la reproducción del mismo sonido.

Los métodos `SoundFacade.pause()` y `SoundFacade.resume()` que se muestran a continuación invocan los métodos `SoundFacade.stop()` y `SoundFacade.play()` respectivamente y pasan un valor de parámetro `pos` cada vez.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

El método `pause()` pasa el valor `SoundChannel.position` al método `play()`, que almacena dicho valor en la propiedad `pausePosition`. El método `resume()` empieza a reproducir de nuevo el mismo sonido con el valor `pausePosition` como punto inicial.

Ampliación del ejemplo de Podcast Player

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En este ejemplo se presenta un Podcast Player básico que aborda la utilización de la clase SoundFacade reutilizable. También se podrían añadir otras funciones para ampliar la utilidad de esta aplicación, entre las que se incluyen:

- Almacenar la lista de fuentes e información de utilización sobre cada episodio de una instancia de SharedObject que se pueda utilizar la próxima vez que el usuario ejecute la aplicación.
- Permitir que el usuario añada su propia fuente RSS a la lista de canales podcast.
- Recordar la posición de la cabeza lectora cuando el usuario detenga o abandone un episodio; de este modo, se puede reiniciar desde dicho punto la próxima vez que el usuario ejecute la aplicación.
- Descargar archivos MP3 de episodios para escuchar cuando el usuario no esté conectado a Internet.
- añadir funciones de suscripción que busquen nuevos episodios periódicamente en un canal podcast y actualicen la lista de episodios de forma automática.
- Añadir funciones de búsqueda y exploración de emisiones podcast con la API de un servicio de alojamiento de podcast, como Odeo.com.

Capítulo 25: Trabajo con vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El vídeo de Flash es una de las tecnologías destacadas de Internet. Sin embargo, la presentación tradicional de vídeo (en una pantalla rectangular con una barra de progreso y algunos botones de control debajo) solo constituye uno de los posibles usos del vídeo. Mediante ActionScript, se obtiene un acceso preciso a la carga, la presentación y la reproducción de vídeo, así como control sobre dichas operaciones.

Fundamentos de la utilización de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una de las funciones importantes de Adobe® Flash® Player and Adobe® AIR™ radica en la capacidad de mostrar y manipular información de vídeo con ActionScript del mismo modo que se pueden manipular otros contenidos visuales, como imágenes, animaciones, texto, etc. Cuando se crea un archivo de Flash Video (FLV) en Adobe Flash CS4 Professional, existe la opción de seleccionar un aspecto que incluya controles de reproducción comunes. No obstante, no hay motivo por el que se deba limitar a las opciones disponibles. Con ActionScript, se obtiene un control preciso de la carga, visualización y la reproducción de vídeo, lo que significa que es posible crear un aspecto de vídeo personalizado o utilizar el vídeo de forma menos tradicional, si se prefiere. La utilización de vídeo en ActionScript implica trabajar con una combinación de varias clases:

- clase Video: el cuadro de contenido de vídeo clásico en el escenario es una instancia de la clase Video. La clase Video es un objeto de visualización, de manera que se puede manipular con las mismas técnicas que se aplican a otros objetos similares, como el ajuste de la posición, la aplicación de transformaciones, la aplicación de filtros y modos de mezcla, etc.
- StageVideo: la clase Video suele utilizar la representación y descodificación con software. Si la aceleración de hardware con GPU está disponible en un dispositivo, la aplicación puede aprovechar la presentación con aceleración por hardware cambiando a la clase StageVideo. La API de StageVideo incluye un conjunto de eventos que indican al código cuándo alternar entre los objetos StageVideo y Video. El vídeo del escenario impone algunas restricciones menores en la reproducción de vídeo. Si la aplicación acepta estas limitaciones, implemente la API de StageVideo. Consulte “Directrices y limitaciones” en la página 536.
- Clase NetStream: cuando se carga un archivo de vídeo para que se controle mediante ActionScript, una instancia de NetStream representa el origen del contenido de vídeo; en este caso, una trasmisión de datos de vídeo. La utilización de la instancia de NetStream también implica utilizar un objeto NetConnection, que es la conexión al archivo de vídeo, al igual que el túnel con el que se alimentan los datos de vídeo.
- Clase Camera: cuando se trabaja con datos de vídeo desde una cámara conectada al equipo del usuario, una instancia de Camera representa el origen del contenido de vídeo; la cámara del usuario y los datos de vídeo que hace disponibles. Como novedad en Flash Player 11.4 y AIR 3.4, puede utilizar una cámara para flujos de StageVideo.

Cuando se carga vídeo externo, se puede cargar el archivo desde un servidor web estándar para obtener una descarga progresiva, o bien, se puede trabajar con un flujo de vídeo transmitido por un servidor especializado como Flash® Media Server de Adobe.

Conceptos y términos importantes

Punto de referencia Marcador que se puede colocar en un instante de tiempo específico en un archivo de vídeo; por ejemplo, funciona como un marcador para buscar ese instante o proporcionar datos adicionales asociados a dicho momento.

Codificación Proceso de convertir datos de vídeo en un formato a otro formato; por ejemplo, convertir un vídeo de origen de alta resolución a un formato adecuado para la transmisión por Internet.

Fotograma Segmento individual de información de vídeo; cada fotograma es como una imagen estática que representa una instantánea de un momento en el tiempo. Si se reproducen fotogramas de forma secuencial a alta velocidad, se crea la ilusión del movimiento.

Fotograma clave Fotograma de vídeo que contiene información completa del fotograma. El resto de fotogramas que siguen a un fotograma clave solo contienen información sobre cómo difieren del fotograma clave, en lugar de incluir información relativa al fotograma completo.

Metadatos Información sobre un archivo de vídeo que se puede incorporar en el archivo de vídeo y recuperarse cuando se haya cargado el vídeo.

Descarga progresiva Cuando se transmite un archivo de vídeo desde un servidor web estándar, los datos de vídeo se cargan mediante la descarga progresiva; es decir, la información de vídeo se carga de forma secuencial. Esto presenta la ventaja de que el vídeo puede empezar a reproducirse antes de que se haya descargado todo el archivo; sin embargo, impide poder saltar a una parte del vídeo que no se haya cargado aún.

Transmisión Como alternativa a la descarga progresiva, se puede utilizar un servidor de vídeo especial para emitir vídeo por Internet mediante la técnica conocida como transmisión de flujo (a veces denominada "transmisión de flujo verdadera"). Con la transmisión de flujo, el ordenador del espectador nunca descarga el vídeo completo a la vez. Para acelerar los tiempos de descarga, el ordenador solo necesita, en cualquier momento, una parte de la información de vídeo total. Puesto que un servidor especial controla la transmisión del contenido de vídeo, se puede acceder en todo momento a cualquier parte del vídeo, en lugar de tener que esperar a que se descargue antes de acceder a él.

Aspectos básicos de los formatos de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además del formato de vídeo Adobe FLV, Flash Player y Adobe AIR admiten audio y vídeo codificados en H.264 y HE-AAC desde formatos de archivo estándar MPEG-4. Estos formatos transmiten vídeo de alta calidad a velocidades de bits más bajas. Los desarrolladores pueden aprovechar herramientas estándar de la industria, incluyendo Adobe Premiere Pro y Adobe After Effects, para crear y proporcionar contenido de vídeo convincente.

Texto	Formato	Contenedor
Vídeo	H.264	MPEG-4: MP4, M4V, F4V, 3GPP
Vídeo	Sorenson Spark	Archivo FLV
Vídeo	ON2 VP6	Archivo FLV
Audio	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4:MP4, M4V, F4V, 3GPP
Audio	MP3	MP3
Audio	Nellymoser	Archivo FLV
Audio	Speex	Archivo FLV

Más temas de ayuda

Flash Media Server: Supported codecs (Flash Media Server: códecs compatibles; en inglés)

Adobe HTTP Dynamic Streaming (en inglés)

Codificación de vídeo para dispositivos móviles

AIR en Android puede descodificar una amplia gama de vídeos H.264. Sin embargo, solo un pequeño subconjunto de vídeos H.264 es adecuado para una reproducción correcta en dispositivos móviles. Esto se debe a que diversos teléfonos móviles tienen limitaciones de capacidad de procesamiento. Adobe Flash Player para móvil puede descodificar vídeos H.264 utilizando la aceleración de hardware incorporada. Esta descodificación garantiza una mejor calidad con un menor consumo de energía.

El estándar H.264 admite varias técnicas de codificación. Solo los dispositivos de alto nivel reproducen vídeos correctamente con niveles y perfiles complejos. Sin embargo, una gran parte de dispositivos puede reproducir vídeo codificado en perfil de línea de base. En los dispositivos móviles, la aceleración de hardware está disponible para un subconjunto de estas técnicas. Los parámetros del nivel y el perfil definen este subconjunto de técnicas de codificación y la configuración utilizada por el codificador. Para los desarrolladores, traduce en codificación el vídeo en la resolución seleccionada que se reproduce correctamente en la mayoría de dispositivos.

Aunque las resoluciones que se benefician de la aceleración de hardware varían de dispositivo a dispositivo, la mayoría admite las siguientes resoluciones estándar.

Relación de aspecto	Resoluciones recomendadas		
4:3	640 × 480	512 × 384	480 × 360
16:9	640 × 360	512 x 288	480 × 272

Nota: *Flash Player admite todos niveles y perfiles del estándar H.264. Al ajustarse a estas recomendaciones, se asegura la aceleración de hardware y se obtiene una mejor experiencia de usuario en la mayoría de dispositivos. Estas recomendaciones no son obligatorias.*

Para obtener información detallada y la configuración de codificación en Adobe Media Encoder CS5, consulte Recommendations for encoding H.264 video for Flash Player 10.1 on mobile devices (http://www.adobe.com/devnet/devices/articles/mobile_video_encoding.html) (Recomendaciones para la codificación de vídeo H.264 para Flash Player 10.1 en dispositivos móviles; en inglés).

Nota: *en iOS, solo se puede reproducir vídeo codificado con los códecs Sorenson Spark y On2 VP6 mediante la clase Video. Es posible reproducir vídeo con codificación H.264 en el reproductor de vídeo del dispositivo si se abre la URL con el vídeo mediante la función* `flash.net.navigateToURL()`*. También se puede reproducir vídeo H.264 usando la etiqueta <video> en una página HTML visualizada en un objeto StageWebView.*

Compatibilidad de Flash Player y AIR con archivos de vídeo codificado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player 7 admite archivos FLV que están codificados con códec de vídeo Sorenson™ Spark™. Flash Player 8 admite archivos FLV codificados con el codificador Sorenson Spark u On2 VP6 en Flash Professional 8. El códec de vídeo On2 VP6 admite un canal alfa.

Flash Player 9.0.115.0 y versiones posteriores admiten archivos que proceden del formato contenedor MPEG-4 estándar. Estos archivos incluyen F4V, MP4, M4A, MOV, MP4V, 3GP y 3G2, si contienen vídeo H.264 o audio codificado HE-AAC v2, o ambos. H.264 proporciona mayor calidad de vídeo a velocidades de bits inferiores en comparación con el mismo perfil de codificación en Sorenson o On2. HE-AAC v2 es una extensión de AAC, un formato de audio estándar definido en el estándar de vídeo MPEG-4. HE-AAC v2 utiliza técnicas de réplica de banda espectral (SBR) y estéreo paramétrico (PS) para aumentar la eficacia de la codificación a velocidades de bits bajas.

En la siguiente tabla se incluyen los códecs compatibles. También se muestra el formato de archivo SWF correspondiente y las versiones de Flash Player y AIR necesarias para reproducirlos:

Códec	**Versión del formato de archivo SWF (primera versión de publicación admitida)**	**Flash Player y AIR (primera versión necesaria para reproducción)**
Sorenson Spark	6	Flash Player 6, Flash Lite 3
On2 VP6	6	Flash Player 8, Flash Lite 3. Únicamente Flash Player 8 y versiones posteriores permiten publicar y reproducir vídeo On2 VP6.
H.264 (MPEG-4 Parte 10)	9	Flash Player 9 Update 3, AIR 1.0
ADPCM	6	Flash Player 6, Flash Lite 3
MP3	6	Flash Player 6, Flash Lite 3
AAC (MPEG-4 Parte 3)	9	Flash Player 9 Update 3, AIR 1.0
Speex (audio)	10	Flash Player 10, AIR 1.5
Nellymoser	6	Flash Player 6

Aspectos básicos de los formatos de archivo de vídeo F4V y FLV de Adobe

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Adobe proporciona los formatos de archivo de vídeo F4V y FLV para trasmitir contenido a Flash Player y AIR. Para obtener una descripción completa de estos formatos de archivo de vídeo, consulte www.adobe.com/go/video_file_format_es.

Formato de archivo de vídeo F4V

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Comenzando con Flash Player actualización 3 (9.0.115.0) y AIR 1.0, Flash Player y AIR admiten el formato de vídeo F4V de Adobe, basado en el formato ISO MP4; los subconjuntos del formato admiten diferentes funciones. Flash Player espera que un archivo F4V válido comience con uno de los siguientes cuadros de nivel superior:

- ftyp

  El cuadro ftyp identifica las funciones que debe admitir un programa para reproducir un formato de archivo concreto.

- moov

  El cuadro moov es realmente el encabezado de un archivo F4V. Contiene uno o varios de los demás cuadros que a su vez incluyen otros cuadros que definen la estructura de los datos F4V. Un archivo F4V debe incluir únicamente un cuadro moov.

- mdat

  Un cuadro mdat contiene la carga útil de datos para el archivo F4V. Un archivo FV contiene únicamente un cuadro mdat. Un cuadro moov también puede estar presente en el archivo, ya que el cuadro mdat no se puede interpretar por sí mismo.

Los archivos F4V admiten enteros multibyte con un orden de bytes bigEndian, en el que el byte más significativo se produce primero, en la dirección más baja.

Formato de archivo de vídeo FLV

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El formato del archivo FLV de Adobe contiene datos de audio y vídeo codificados para la publicación con Flash Player. Puede utilizar un codificador, como Adobe Media Encoder orSorenson™ Squeeze, para convertir un archivo de vídeo de QuickTime o Windows Media en un archivo FLV.

Nota: *se pueden crear archivos FLV importando vídeo en Flash y exportándolo como archivo FLV. Se puede utilizar el complemento de exportación de FLV para exportar archivos FLV de aplicaciones de edición de vídeo compatibles. Para cargar archivos FLV desde un servidor web, es necesario registrar la extensión de nombre de archivo y el tipo MIME con el servidor web. Consulte la documentación de su servidor web. El tipo MIME de los archivos FLV es* `video/x-flv`*. Para más información, consulte "Configuración de archivos FLV para alojar en el servidor" en la página 527.*

Para obtener más información sobre archivos FLV, consulte "Temas avanzados para archivos FLV" en la página 526.

Vídeo externo frente a incorporado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La utilización de archivos de vídeo externos ofrece algunas posibilidades que no están disponibles al utilizar vídeo importado:

- Se pueden utilizar clips de vídeo más largos en su aplicación sin que ello ralentice la reproducción. Los archivos de vídeo externos utilizan memoria en caché, lo que significa que los archivos de gran tamaño se almacenan en pequeñas partes y se accede a los mismos de forma dinámica. Por este motivo, los archivos FLV y F4V externos requieren menos memoria que los archivos de vídeo incrustados.
- Un archivo de vídeo externo puede tener una velocidad de fotogramas distinta a la del archivo SWF en el que se reproduce. Por ejemplo, la velocidad de fotogramas del archivo SWF se puede establecer en 30 fotogramas por segundo (fps) y la del vídeo en 21 fps. Esta opción permite un mejor control del vídeo que el vídeo incorporado, para garantizar una reproducción del vídeo sin problemas. Asimismo, permite reproducir archivos de vídeo a distintas velocidades de fotogramas sin necesidad de modificar el contenido del archivo SWF existente.
- Con los archivos de vídeo externos, la reproducción de contenido SWF no se interrumpe mientras que el archivo de vídeo se está cargando. A veces, los archivos de vídeo importados pueden interrumpir la reproducción de un documento para realizar ciertas funciones, como acceder a una unidad de CD-ROM. Los archivos de vídeo pueden realizar funciones independientemente del contenido SWF, sin que la reproducción se vea interrumpida.
- La subtitulación de contenido de vídeo es más fácil con los archivos FLV externos, ya que es posible acceder a los metadatos del vídeo mediante controladores de eventos.

Aspectos básicos de la clase Video

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Video permite mostrar un flujo de vídeo en vivo en una aplicación sin incorporarlo al archivo SWF. Se puede capturar y reproducir vídeo en vivo mediante el método `Camera.getCamera()`. También se puede utilizar la clase Video para reproducir archivos de vídeo en HTTP o desde el sistema de archivos local. Hay cuatro formas diferentes de utilizar la clase Video en los proyectos:

- Cargar un archivo de vídeo dinámicamente con las clases NetConnection y NetStream y mostrar el vídeo en un objeto Video.
- Capturar las entradas de la cámara del usuario. Para obtener más información, consulte "Trabajo con cámaras" en la página 543.
- Utilizar el componente FLVPlayback.
- Utilizar el control VideoDisplay.

Nota: *las instancias de un objeto Video en el escenario son instancias de la clase Video.*

Aunque la clase Video se encuentra en el paquete flash.media, hereda de la clase flash.display.DisplayObject; por lo tanto, toda la funcionalidad del objeto de visualización (como las transformaciones de matriz y los filtros) también se aplican a las instancias de Video.

Para obtener más información, consulte "Manipulación de objetos de visualización" en la página 178, "Trabajo con la geometría" en la página 217 y "Aplicación de filtros a objetos de visualización" en la página 279.

Carga de archivos de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La carga de vídeos con las clases NetStream y NetConnection es un proceso de varios pasos. Como práctica recomendada, los pasos de añadir el objeto Video a la lista de visualización, asociar el objeto NetStream a la instancia de Video y llamar al método `play()` del objeto NetStream deben realizarse en el orden especificado:

1. Cree un objeto NetConnection. Si se va a conectar a un archivo de vídeo local o a uno que no utilice un servidor, como Flash Media Server 2 de Adobe, transmita el valor `null` al método `connect()` para reproducir los archivos de vídeo desde una dirección HTTP o una unidad local. Si se conecta a un servidor, defina este parámetro con el URI de la aplicación que contiene el archivo de vídeo en el servidor.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. Cree un nuevo objeto Video que muestre el vídeo y añádalo a la lista de visualización del escenario, tal y como se puede ver en este fragmento de código:

```
var vid:Video = new Video();
addChild(vid);
```

3. Cree un objeto NetStream transfiriendo el objeto NetConnection como un argumento al constructor. El siguiente fragmento de código conecta un objeto NetStream con la instancia de NetConnection y define los controladores de evento para el flujo:

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);

function netStatusHandler(event:NetStatusEvent):void
{
    // handle netStatus events, described later
}

function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

4 Asocie el objeto NetStream al objeto Video mediante el método `attachNetStream()` del objeto Video, tal y como puede verse en el siguiente fragmento de código:

```
vid.attachNetStream(ns);
```

5 Llame al método `play()` del objeto NetStream con la URL del archivo de vídeo como un argumento para iniciar la reproducción del vídeo. El siguiente fragmento de código carga y reproduce un archivo de vídeo llamado "video.mp4" en el mismo directorio que el archivo SWF:

```
ns.play("video.mp4");
```

Más temas de ayuda

Flex: Spark VideoPlayer control (en inglés)

spark.components.VideoDisplay

Control de la reproducción de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase NetStream ofrece cuatro métodos principales para controlar la reproducción de vídeo:

`pause()`: detiene la reproducción de un flujo de vídeo. Si el vídeo ya está en pausa, la llamada a este método no tendrá ningún efecto.

`resume()`: reanuda la reproducción de un flujo de ví-deo que se ha detenido. Si el vídeo ya se está reproduciendo, la llamada a este método no tendrá ningún efecto.

`seek()`: busca el fotograma clave más cercano a la ubicación especificada (un desplazamiento, en segundos, desde el comienzo del flujo).

`togglePause()`: realiza una pausa o reanuda la reproducción de un flujo.

Nota: *el método `stop()` no está disponible. Para detener un flujo, se debe pausar la reproducción y buscar el principio del flujo de vídeo.*

Nota: *el método `play()` no reanuda la reproducción; se utiliza para cargar archivos de vídeo.*

En el ejemplo siguiente se demuestra cómo controlar un vídeo mediante diferentes botones. Para ejecutar el siguiente ejemplo, cree un nuevo documento y añada cuatro instancias de botón al espacio de trabajo (`pauseBtn`, `playBtn`, `stopBtn` y `togglePauseBtn`):

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

Si se hace clic en la instancia de botón `pauseBtn`, el archivo de vídeo quedará en pausa. Si el vídeo ya está en pausa, hacer clic en este botón no tendrá ningún efecto. Si se hace clic en el botón `playBtn`, se reanuda la reproducción de vídeo si esta estaba en pausa anteriormente; de lo contrario, el botón no tendrá ningún efecto si el vídeo ya estaba en reproducción.

Detección del final de un flujo de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para poder detectar el principio y el final de un flujo de vídeo, se debe añadir un detector de eventos a la instancia de NetStream para el evento `netStatus`. En el código siguiente se demuestra cómo detectar varios códigos mediante la reproducción del vídeo:

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

El código anterior genera el resultado siguiente:

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

Los dos códigos que se desean detectar específicamente son "NetStream.Play.Start" y "NetStream.Play.Stop", que señalan el principio y el final de la reproducción del vídeo. El siguiente fragmento utiliza una sentencia switch para filtrar estos dos códigos y rastrear un mensaje:

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

Si se detecta el evento `netStatus` (`NetStatusEvent.NET_STATUS`), se puede crear un reproductor de vídeo que cargue el vídeo siguiente de una lista de reproducción una vez que haya terminado de reproducirse el vídeo actual.

Reproducción de vídeo en modo de pantalla completa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player y AIR permiten crear una aplicación de pantalla completa para la reproducción de vídeo y admiten la escala de vídeo a pantalla completa.

Para el contenido de AIR que se ejecuta en modo de pantalla completa en el escritorio, el protector de pantalla del sistema y las opciones de ahorro de energía se desactivan durante la reproducción hasta que la entrada de vídeo se detenga o el usuario salga del modo de pantalla completa.

Para obtener información detallada sobre el uso del modo de pantalla completa, consulte “Trabajo con el modo de pantalla completa” en la página 171.

Activación del modo de pantalla completa para Flash Player en un navegador

Antes de que pueda implementar el modo de pantalla completa para Flash Player en un navegador, actívelo mediante la plantilla de publicación para su aplicación. Las plantillas que permiten la pantalla completa incluyen etiquetas `<object>` y `<embed>` que contienen un parámetro `allowFullScreen`. El siguiente ejemplo muestra el parámetro `allowFullScreen` en una etiqueta `<embed>`.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

En Flash, seleccione Archivo -> Configuración de publicación y en el cuadro de diálogo Configuración de publicación, en la ficha HTML, seleccione la plantilla Solo Flash - Permitir pantalla completa.

En Flex, asegúrese de que la plantilla HTML incluya las etiquetas `<object>` y `<embed>` que son compatibles con la pantalla completa.

Inicio del modo de pantalla completa

Para el contenido de Flash Player que se ejecuta en un navegador, el modo de pantalla completa se inicia para vídeo como respuesta a un clic de ratón o a una pulsación de tecla. Por ejemplo, el modo de pantalla completa se puede iniciar cuando el usuario hace clic en un botón con la etiqueta de modo de pantalla completa o selecciona un comando Pantalla completa en un menú contextual. Para responder al usuario, añada un detector de eventos al objeto en el que sucede la operación. El siguiente código añade un detector de eventos a un botón en el que el usuario hace clic para introducir el modo de pantalla completa:

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

El código inicia el modo de pantalla completa estableciendo la propiedad `Stage.displayState` en `StageDisplayState.FULL_SCREEN`. Este código escala todo el escenario a modo de pantalla completa con el ajuste de escala de vídeo en proporción al espacio que ocupa en el escenario.

La propiedad `fullScreenSourceRect` permite especificar un área concreta del escenario para realizar un ajuste de escala a pantalla completa. En primer lugar, defina el rectángulo que desee escalar a pantalla completa. Después asígnelo a la propiedad `Stage.fullScreenSourceRect`. Esta versión de la función `fullScreenButtonHandler()` agrega dos líneas adicionales de código que escalan solo el vídeo a pantalla completa.

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Aunque en este ejemplo se invoca un controlador de eventos como respuesta a un clic de ratón, la técnica de pasar al modo de pantalla completa es la misma tanto para Flash Player como para AIR. Defina el rectángulo que desee escalar y, a continuación, establezca la propiedad `Stage.displayState`. Para obtener más información, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

El siguiente ejemplo completo agrega código que crea la conexión y el objeto NetStream para el vídeo y comienza a reproducirlo.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
```

```
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width,
video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

La función `onMetaData()` es una función callback para administrar metadatos de vídeo, si existen. Una función callback es una función que llama el tiempo de ejecución como respuesta a algún tipo de instancia o evento. En este ejemplo, la función `onMetaData()` es un código auxiliar que cumple con el requisito de proporcionar la función. Para obtener más información, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 507.

Cancelación del modo de pantalla completa

Un usuario puede cancelar el modo de pantalla completa introduciendo uno de los métodos abreviados de teclado como, por ejemplo, la tecla Esc. Para cancelar el modo de pantalla completa en ActionScript, establezca la propiedad `Stage.diplayState` en `StageDisplayState.NORMAL`. El código del siguiente ejemplo cancela el modo de pantalla completa cuando se produce el evento `netStatus` de NetStream.Play.Stop.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

Aceleración de hardware a pantalla completa

Cuando se vuelve a escalar un área rectangular del escenario en modo de pantalla completa, Flash Player o AIR utilizan la aceleración de hardware, si está disponible y habilitada. El tiempo de ejecución utiliza el adaptador de vídeo del equipo para agilizar el ajuste de escala del vídeo, o una parte del escenario, al tamaño de pantalla competa. En este caso, las aplicaciones de Flash Player a menudo lo aprovechan cambiando a la clase StageVideo desde la clase Video (o la clase Camera; Flash Player 11.4/AIR 3.4 y posterior).

Para obtener más información sobre la aceleración de hardware a pantalla completa, consulte "Trabajo con el modo de pantalla completa" en la página 171. Para obtener más información sobre StageVideo, consulte "Uso de la clase StageVideo para la presentación con aceleración por hardware" en la página 534.

Transmisión de archivos de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para transmitir archivos de Flash Media Server, se pueden utilizar las clases NetConnection y NetStream para conectarse a una instancia de servidor remoto y reproducir un flujo especificado. Para especificar un servidor RTMP (Real-Time Messaging Protocol), la URL de RTMP deseada como, por ejemplo, "rtmp://localhost/appName/appInstance", se transmite al método `NetConnection.connect()` en lugar de transmitir un valor `null`. Para reproducir un flujo grabado o en directo determinado desde el servidor Flash Media Server especificado, se transmite un nombre identificativo para los datos en directo publicados por `NetStream.publish()`, o bien, un nombre de archivo grabado para reproducción al método `NetStream.play()`.

Envío de vídeo a un servidor

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se desean crear aplicaciones más complejas con objetos Video o Camera, Flash Media Server ofrece una combinación de funciones de flujo de medios y un entorno de desarrollo adecuado para crear y publicar aplicaciones multimedia para un público numeroso. Esta combinación permite que los desarrolladores creen aplicaciones como el vídeo bajo demanda, las difusiones de eventos web en vivo, las transmisiones de MP3, así como los blogs de vídeo, la mensajería de vídeo y los entornos de chat multimedia. Para obtener más información, consulte la documentación de Flash Media Server en línea en www.adobe.com/go/learn_fms_docs_es.

Aspectos básicos de los puntos de referencia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede incorporar puntos de referencia en un archivo de vídeo F4V o FLV de Adobe durante la codificación. Los puntos de referencia siempre se han incorporado a las películas para proporcionar al proyectista una señal visual que indicara que la cinta estaba llegando a su fin. En los formatos de vídeo F4V y FLV de Adobe, un punto de referencia permite activar una o varias acciones en la aplicación en el momento en que se producen en el flujo de vídeo.

Se pueden utilizar diferentes tipos de puntos de referencia con Flash Video. ActionScript permite interactuar con puntos de referencia que se incorporen en un archivo de vídeo al crearlo.

- Puntos de referencia de navegación: estos puntos se incorporan al flujo de vídeo y al paquete de metadatos al codificar el archivo de vídeo. Los puntos de referencia de navegación se utilizan para permitir a los usuarios buscar una parte especificada de un archivo.
- Puntos de referencia de evento: los puntos de referencia de evento se incorporan al flujo de vídeo y al paquete de metadatos FLV al codificar el archivo de vídeo. Se puede escribir código para controlar eventos que se activan en puntos especificados durante la reproducción de vídeo.

- Puntos de referencia de ActionScript: se trata de puntos referencia que solo están disponibles en el componente FLVPlayback de Flash. Los puntos de referencia de ActionScript son puntos externos que se crean y se accede a ellos con el código de ActionScript. Se puede escribir código para activar estos puntos de referencia en relación con la reproducción del vídeo. Estos puntos de referencia son menos precisos que los incorporados (hasta una décima de segundo), ya que el reproductor de vídeo realiza un seguimiento de los mismos de forma independiente. Si se va a crear una aplicación en la que se desea que los usuarios naveguen a un punto de referencia, se deben crear e incorporar puntos de referencia al codificar el archivo en lugar de utilizar puntos de referencia de ActionScript. Se recomienda incorporar los puntos de referencia en el archivo FLV, ya que resultan más precisos.

Los puntos de referencia de navegación crean un fotograma clave en una ubicación especificada, por lo que se puede utilizar código para desplazar la cabeza lectora del reproductor de vídeo a dicha ubicación. Se pueden establecer puntos determinados en un archivo de vídeo donde se desee que busquen los usuarios. Por ejemplo, si el vídeo incluyera varios capítulos o segmentos, se podría controlar mediante la incorporación de puntos de referencia de navegación en el archivo de vídeo.

Para más información sobre la codificación de archivos de vídeo de Adobe con puntos de referencia, consulte "Incorporación de puntos de referencia" *en Uso de Flash*.

Se puede acceder a parámetros de punto de referencia al escribir código ActionScript. Los parámetros de punto de referencia constituyen una parte del objeto de evento recibido mediante el controlador callback.

Para activar determinadas acciones en el código cuando un archivo FLV alcanza un punto de referencia específico, se utiliza el controlador de eventos `NetStream.onCuePoint`.

Para sincronizar una acción para un punto de referencia en un archivo de vídeo F4V, debe recuperar los datos del punto desde las funciones callback `onMetaData()` o `onXMPData()` y activar el punto de referencia utilizando la clase Timer en ActionScript 3.0. Para obtener más información sobre los puntos de referencia F4V, consulte "Uso de onXMPData()" en la página 519.

Para obtener más información sobre la administración de puntos de referencia y metadatos, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 507.

Escritura de métodos callback para metadatos y puntos de referencia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede activar acciones en su aplicación cuando el reproductor reciba metadatos específicos o cuando se alcancen puntos de referencia concretos. Cuando se produzcan estos eventos, debe utilizar métodos callback específicos como controladores de eventos. La clase NetStream especifica los siguientes eventos de metadatos que pueden suceder durante la reproducción: `onCuePoint` (solo archivos FLV), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` y `onXMPData`.

Se deben escribir métodos callback para estos controladores; de lo contrario, el motor de ejecución de Flash podría generar errores. Por ejemplo, en el código siguiente se reproduce un archivo FLV denominado video.flv en la misma carpeta donde se encuentra el archivo SWF:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

El código anterior carga un archivo de vídeo local llamado video.flv y detecta la distribución de `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`). Este evento se distribuye cuando se genera una excepción desde el código asincrónico nativo. En este caso, se distribuye cuando un el archivo de vídeo contiene metadatos o información de punto de referencia y no se han definido los detectores apropiados. El código anterior controla el evento `asyncError` y omite el error si no interesan los metadatos o la información de punto de referencia del archivo de vídeo. Si tuviera un archivo FLV con metadatos y varios puntos de referencia, la función `trace()` mostraría los siguientes mensajes de error:

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

Los errores se producen porque el objeto NetStream no ha podido encontrar un método callback `onMetaData` o `onCuePoint`. Hay varias maneras de definir estos métodos callback en las aplicaciones.

Más temas de ayuda

Flash Media Server: Handling metadata in streams (Flash Media Server: Administración de metadatos en flujos; en inglés)

Definición de la propiedad client del objeto NetStream en Object

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al establecer la propiedad `client` en Object o una subclase de NetStream, se pueden redirigir los métodos callback `onMetaData` y `onCuePoint`, o bien, omitirlos completamente. En el ejemplo siguiente se demuestra cómo se puede utilizar una clase Object vacía para omitir los métodos callback sin detectar el evento `asyncError`.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Si se desea detectar los métodos callback `onMetaData` o `onCuePoint`, es necesario definir métodos para controlarlos, tal y como se muestra en el siguiente fragmento de código:

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

El código anterior detecta el método callback `onMetaData` y llama al método `metaDataHandler()`, que rastrea una cadena. Si el motor de ejecución de Flash ha detectado un punto de referencia, no se generarán errores aunque se defina el método callback `onCuePoint`.

Creación de una clase personalizada y definición de métodos para controlar los métodos callback

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el código siguiente se establece la propiedad `client` del objeto NetStream en una clase personalizada, CustomClient, que define controladores para los métodos callback:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La clase CustomClient se presenta del siguiente modo:

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

La clase CustomClient define un controlador para el controlador callback `onMetaData`. Si se ha detectado un punto de referencia y se ha llamado al controlador callback `onCuePoint`, se distribuirá un evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) indicando que flash.net.NetStream no ha podido invocar la función callback onCuePoint. Para evitar este error, se debe definir un método callback `onCuePoint` en la clase CustomClient, o bien definir un controlador de eventos para el evento `asyncError`.

Ampliación de la clase NetStream y adición de métodos para controlar los métodos callback

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El código siguiente crea una instancia de la clase CustomNetStream, que se define en un listado de código posterior:

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

En la lista de códigos siguiente se define la clase CustomNetStream que amplía la clase NetStream y controla la creación del objeto NetConnection necesario y los métodos de controlador callback `onMetaData` y `onCuePoint`:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Si se desea cambiar el nombre de los métodos `onMetaData()` y `onCuePoint()` en la clase CustomNetStream, se podría utilizar el código siguiente:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Ampliación y dinamización de la clase NetStream

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede ampliar la clase NetStream y hacer que la subclase sea dinámica, de manera que los controladores callback `onCuePoint` y `onMetaData` puedan añadirse dinámicamente. Esto se ilustra en la lista siguiente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La clase DynamicCustomNetStream es de esta forma:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

Incluso sin controladores para los controladores callback `onMetaData` y `onCuePoint`, no se generan errores, ya que la clase DynamicCustomNetStream es dinámica. Si se desea definir métodos para los controladores callback `onMetaData` y `onCuePoint`, se puede utilizar el código siguiente:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

Establecimiento de la propiedad client del objeto NetStream en this

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si la propiedad `client` se establece en `this`, la aplicación busca en el ámbito actual los métodos `onMetaData()` y `onCuePoint()`. Esto se puede observar en el ejemplo siguiente:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Si se llama a los controladores callback `onMetaData` o `onCuePoint` y no hay métodos para controlar la función callback, no se generan errores. Para controlar estos controladores callback, hay que crear métodos `onMetaData()` y `onCuePoint()` en el código, tal como se muestra en el siguiente fragmento de código:

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

Uso de puntos de referencia y metadatos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos callback NetStream se utilizan para capturar y procesar eventos de metadatos y puntos de referencia conforme se reproduce el vídeo.

Uso de puntos de referencia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En la siguiente tabla se describen los métodos callback que se pueden utilizar para capturar puntos de referencia F4V y FLV en Flash Player y AIR.

Tiempo de ejecución	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

En el siguiente ejemplo se utiliza un sencillo bucle `for..in` para repetir todas las propiedades del parámetro `infoObject` que recibe la función `onCuePoint()`. Llama a la función trace() para que aparezca un mensaje cuando reciba los datos de punto de referencia:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

Aparece el siguiente resultado:

```
parameters:
name: point1
time: 0.418
type: navigation
```

Este código utiliza una de las distintas técnicas para configurar el objeto en el que se ejecuta el método callback. También puede utilizar otras técnicas; para obtener más información, consulte "Escritura de métodos callback para metadatos y puntos de referencia" en la página 507.

Uso de metadatos de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las funciones `OnMetaData()` y `OnXMPData()` se pueden emplear para acceder a la información de metadatos en su archivo de vídeo, incluyendo puntos de referencia.

Uso de OnMetaData()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los metadatos incluyen información sobre el archivo de vídeo, como la duración, anchura, altura y velocidad de fotogramas. La información de metadatos que se añade al archivo de vídeo depende del software que se utilice para codificar el archivo de vídeo.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

El código anterior genera resultados como los siguientes:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

Si el vídeo no tiene sonido, la información de metadatos relativa al audio (como `audiodatarate`) devuelve `undefined`, ya que no se ha añadido información de audio a los metadatos durante la codificación.

En el código anterior, no se muestra la información de punto de referencia. Para ver los metadatos del punto de referencia, se puede utilizar la siguiente función que muestra de forma sucesiva los elementos en una clase Object:

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

Mediante el fragmento de código anterior para rastrear el parámetro `infoObject` del método `onMetaData()`, se crea la salida siguiente:

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

El siguiente ejemplo muestra los metadatos para un vídeo MP4. Se da por hecho que existe un objeto llamado `metaDataOut`, en el que se escriben los metadatos.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

La función `onMetaData()` generó los siguientes resultados para este vídeo:

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

Uso del objeto information

La siguiente tabla muestra los posibles valores para los metadatos de vídeo que se transmiten a la función callback `onMetaData()` en el objeto que reciben:

Parámetro	Descripción
aacaot	Tipo de objeto de audio AAC; se admiten 0, 1 o 2.
avclevel	Número de nivel AVC IDC como, por ejemplo, 10, 11, 20, 21, etc.
avcprofile	Número de perfil AVC como, por ejemplo, 55, 77, 100, etc.
audiocodecid,	Cadena que indica el códec de audio (técnica de codificación/descodificación) utilizado; por ejemplo, ".Mp3" o "mp4a"
audiodatarate	Número que indica la velocidad a la que se ha codificado el audio, expresada en kilobytes por segundo.
audiodelay	Número que indica el tiempo del archivo FLV correspondiente al "tiempo 0" del archivo FLV original. Es necesario demorar ligeramente el contenido del vídeo para sincronizarlo correctamente con el audio.
canSeekToEnd	Un valor booleano que es `true` si el archivo FLV se codifica con un fotograma clave en el último fotograma, lo que permite buscar hasta el final de un clip de película de descarga progresiva. Es `false` si el archivo FLV no se codifica con un fotograma clave en el último fotograma.
cuePoints	Conjunto de objetos, uno en cada punto de referencia incorporado en el archivo FLV. El valor es undefined si el archivo FLV no contiene ningún punto de referencia. Cada objeto tiene las siguientes propiedades: • `type`: cadena que especifica el tipo de punto de referencia como "navigation" o "event". • `name`: cadena que indica el nombre del punto de referencia. • `time`: número que indica el tiempo del punto de referencia en segundos, con una precisión de tres decimales (milisegundos). • `parameters`: objeto opcional que tiene pares nombre-valor designados por el usuario durante la creación de los puntos de referencia.
duration	Número que especifica la duración del archivo de vídeo, en segundos.
framerate	Número que especifica la velocidad de fotogramas del archivo FLV.
height	Número que especifica la altura del archivo FLV, en píxeles.
seekpoints	Conjunto que incluye los fotogramas clave disponibles como marcas horarias en milisegundos. Opcional.
etiquetas	Conjunto de pares clave-valor que representa la información en el átomo "ilst", que es el equivalente de las etiquetas ID3 para archivos MP4. iTunes utiliza estas etiquetas. Se puede utilizar para mostrar ilustraciones, si está disponible.
trackinfo	Objeto que proporciona información sobre todas las pistas del archivo MP4, incluyendo su ID de descripción de ejemplo.
videocodecid,	Cadena que es la versión de códec que se utilizó para codificar el vídeo; por ejemplo, "avc1" o "VP6F".

Parámetro	Descripción
videodatarate	Número que especifica la velocidad de datos de vídeo del archivo FLV.
videoframerate	Velocidad de fotogramas del vídeo MP4.
width	Número que especifica la anchura del archivo FLV, en píxeles.

La tabla siguiente muestra los posibles valores del parámetro `videocodecid`:

videocodecid,	Nombre de códec
2	Sorenson H.263
3	Screen video (solo en SWF versión 7 y posterior)
4	VP6 (solo en SWF versión 8 y posterior)
5	Vídeo VP6 con canal alfa (solo en SWF versión 8 y posterior)

La tabla siguiente muestra los posibles valores del parámetro `audiocodecid`:

audiocodecid,	Nombre de códec
0	uncompressed
1	ADPCM
2	MP3
4	Nellymoser @ 16 kHz mono
5	Nellymoser, 8kHz mono
6	Nellymoser
10	AAC
11	Speex

Uso de onXMPData()

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

La función callback `onXMPData()` recibe información específica de Adobe Extensible Metadata Platform (XMP) incorporada en el archivo de vídeo Adobe F4V o FLV. Los metadatos XMP incluyen puntos de referencia, así como otros metadatos de vídeo. La compatibilidad con metadatos XMP se incorpora a partir de Flash Player 10 y Adobe AIR 1.5 y se admite en versiones posteriores.

En el siguiente ejemplo se procesan datos del punto de referencia en los metadatos XMP:

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
```

```
                var strTrackName:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                var strFrameRateXML:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                nTracksFrameRate = Number(strFrameRate);

                strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

Para un breve archivo de vídeo denominado startrekintro.f4v, este ejemplo produce las siguientes líneas de trazado. Las líneas muestran los datos del punto de referencia para los puntos de referencia de evento y navegación en los metadatos XMP:

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

Nota: *en los datos XMP, el tiempo se almacena como ticks DVA en lugar de en segundos. Para calcular la hora del punto de referencia, divida la hora de inicio entre la velocidad de fotogramas. Por ejemplo, la hora de inicio de 7695905817600 dividida entre una velocidad de fotogramas de 254016000000 es igual a 30:30.*

Para ver los metadatos XMP completos sin formato, lo que incluye la velocidad de fotogramas, elimine los identificadores de comentarios (//'s) que preceden a la segunda y tercera sentencia `trace()` al principio de la función `onXMPData()`.

Para obtener más información sobre XMP, consulte:

- partners.adobe.com/public/developer/xmp/topic.html
- www.adobe.com/devnet/xmp/

Uso de metadatos de imagen

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El evento `onImageData` envía los datos de imagen como un conjunto de bytes a través de un canal de datos AMF0. Los datos se pueden presentar en formatos JPEG, PNG o GIF. Defina un método callback `onImageData()` para procesar esta información, del mismo modo que definiría métodos callback para `onCuePoint` y `onMetaData`. El siguiente ejemplo muestra y accede a los datos de imagen utilizando el método callback `onImageData()`:

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

Uso de metadatos de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El evento `onTextData` envía datos de texto mediante un canal de datos AMF0. Los datos de texto presentan un formato UTF-8 y contienen información adicional sobre el formato, en función de la especificación de texto temporizado 3GP. Esta especificación define un formato de subtítulo estandarizado. Defina un método callback `onTextData()` para procesar esta información, del mismo modo que definiría métodos callback para `onCuePoint` o `onMetaData`. En el siguiente ejemplo, el método `onTextData()` muestra el número de ID de pista y el texto de pista correspondiente.

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

Control de la actividad de NetStream

Flash Player 10.3 y posterior, Adobe AIR 2.7 y posterior

Es posible controlar la actividad de NetStream para poder recopilar la información necesaria para admitir el informe y el análisis del uso de medios. Las funciones de control analizadas en esta sección permiten crear bibliotecas de medición de medios que recopilan datos sin un acoplamiento cerrado en el reproductor de vídeo concreto que muestra los medios. Esto permite que los desarrolladores del cliente seleccionen sus reproductores de vídeo favoritos al utilizar su biblioteca. Utilice la clase NetMonitor para controlar la creación y la actividad de los objetos NetStream en una aplicación. La clase NetMonitor proporciona una lista del objeto NetStreams activo existente en cualquier momento concreto y también distribuye un evento siempre que se crea un objeto NetStream.

El objeto NetStream distribuye los eventos incluidos en la siguiente tabla, dependiendo del tipo de medios que se reproduzcan:

Evento	Descarga progresiva	Flujo RTMP	Flujo HTTP
NetStream.Play.Start	Sí	Sí	No
NetStream.Play.Stop	Sí	Sí	No
NetStream.Play.Complete	Sí	Sí	No
NetStream.SeekStart.Notify	Sí	Sí	Sí
NetStream.Seek.Notify	Sí	Sí	Sí
NetStream.Unpause.Notify	Sí	Sí	Sí
NetStream.Unpause.Notify	Sí	Sí	Sí
NetStream.Play.Transition	No se aplica	Sí	No se aplica
NetStream.Play.TransitionComplete	No se aplica	Sí	No se aplica
NetStream.Buffer.Full	Sí	Sí	Sí
NetStream.Buffer.Flush	Sí	Sí	Sí
NetStream.Buffer.Empty	Sí	Sí	Sí

El objeto NetStreamInfo asociado a una instancia de NetStream también almacena los últimos metadatos y los objetos de datos XMP que se encontraron en los medios.

Cuando los medios se reproducen mediante flujo HTTP, NetStream.Play.Start, NetStream.Play.Stop y NetStream.Play.Complete no se distribuyen, ya que la aplicación cuenta con un control total del flujo de medios. Un reproductor de vídeo debe sintetizar y distribuir estos eventos para los flujos HTTP.

Del mismo modo, NetStream.Play.Transition y NetStream.Play.TransitionComplete no se distribuyen para ninguna descarga progresiva ni medio HTTP. El cambio de velocidad de bits dinámica es una función de RTMP. Si un reproductor de vídeo que utiliza un flujo HTTP admite una función similar, el reproductor puede sintetizar y distribuir eventos de transición.

Más temas de ayuda

Adobe Developer Connection: Measuring video consumption in Flash (Medición del consumo de vídeo en Flash; en inglés)

Control de eventos NetStream

Dos tipos de eventos proporcionan datos de uso útil: `netStatus` y `mediaTypeData`. Asimismo, se puede emplear un temporizador para registrar periódicamente la posición de la cabeza lectora de NetStream.

Los eventos `netStatus` proporcionan información que se puede utilizar para determinar qué cantidad de flujo ha visualizado un usuario. Los eventos de transición de flujo de búfer y RTMFP también generan un evento `netStatus`.

Los eventos `mediaTypeData` proporcionan información de metadatos y datos XMP. El evento Netstream.Play.Complete se distribuye como evento `mediaTypeData`. Otros datos incorporados en el flujo también se encuentran disponibles mediante los eventos `mediaTypeData`, incluyendo puntos de referencia, texto e imágenes.

El siguiente ejemplo muestra cómo crear una clase que controle los eventos de datos y estados desde cualquier objeto NetStreams activo en una aplicación. Generalmente, una clase de este tipo cargaría los datos en los que estuviera interesada realizando el análisis en un servidor para recopilación.

```
package com.adobe.example
{
    import flash.events.NetDataEvent;
    import flash.events.NetMonitorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetMonitor;
    import flash.net.NetStream;

    public class NetStreamEventMonitor
    {
        private var netmon:NetMonitor;
        private var heartbeat:Timer = new Timer( 5000 );

        public function NetStreamEventMonitor()
        {
            //Create NetMonitor object
            netmon = new NetMonitor();
            netmon.addEventListener( NetMonitorEvent.NET_STREAM_CREATE, newNetStream );

            //Start the heartbeat timer
            heartbeat.addEventListener( TimerEvent.TIMER, onHeartbeat );
            heartbeat.start();
        }

        //On new NetStream
        private function newNetStream( event:NetMonitorEvent ):void
        {
            trace( "New Netstream object");
            var stream:NetStream = event.netStream;
            stream.addEventListener(NetDataEvent.MEDIA_TYPE_DATA, onStreamData);
            stream.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
        }

        //On data events from a NetStream object
        private function onStreamData( event:NetDataEvent ):void
        {

            var netStream:NetStream = event.target as NetStream;
            trace( "Data event from " + netStream.info.uri + " at " + event.timestamp );
            switch( event.info.handler )
            {
                case "onMetaData":
                    //handle metadata;
                    break;
                case "onXMPData":
                    //handle XMP;
                    break;
                case "onPlayStatus":
                    //handle NetStream.Play.Complete
                case "onImageData":
                    //handle image
                    break;
                case "onTextData":
                    //handle text
                    break;
                default:
                    //handle other events
```

```
            }
        }

        //On status events from a NetStream object
        private function onStatus( event:NetStatusEvent ):void
        {
            trace( "Status event from " + event.target.info.uri + " at " + event.target.time );
            //handle status events
        }
        //On heartbeat timer
        private function onHeartbeat( event:TimerEvent ):void
        {
            var streams:Vector.<NetStream> = netmon.listStreams();
            for( var i:int = 0; i < streams.length; i++ )
            {
                trace( "Heartbeat on " + streams[i].info.uri + " at " + streams[i].time );
                //handle heartbeat event
            }
        }

    }
}
```

Detección del dominio del reproductor

La URL y el dominio de la página web en la que el usuario está visualizando el contenido de medios no siempre están inmediatamente disponibles. Si el sitio web de alojamiento lo permite, se puede usar la clase ExternalInterface para obtener la URL exacta. Sin embargo, algunos sitios web que permiten reproductores de terceros no autorizan el uso de ExternalInterface. En estos casos, se puede obtener el dominio de la página web actual a partir de la propiedad `pageDomain` de la clase Security. La URL completa no se divulga por motivos de privacidad y seguridad.

El dominio de la página se encuentra disponible mediante la propiedad estática `pageDomain` de la clase Security:

```
var domain:String = Security.pageDomain;
```

Temas avanzados para archivos FLV

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los siguientes temas abordan algunos problemas especiales del trabajo con archivos FLV.

Configuración de archivos FLV para alojar en el servidor

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al trabajar con archivos FLV, es posible que sea necesario configurar el servidor para que funcione con el formato de archivo FLV. MIME (Multipurpose Internet Mail Extensions) es una especificación de datos estandarizada que permite enviar archivos que no son ASCII a través de conexiones de Internet. Los navegadores web y los clientes de correo electrónico se configuran para interpretar numerosos tipos MIME de modo que puedan enviar y recibir vídeo, audio, gráficos y texto con formato. Para cargar archivos FLV de un servidor web, es posible que se necesite registrar la extensión de archivo y el tipo MIME en el servidor web, por lo que se debe comprobar la documentación del servidor web. El tipo MIME de los archivos FLV es `video/x-flv`. La información completa del tipo de archivo FLV es la siguiente:

- Tipo Mime: vídeo/x-flv
- Extensión de archivo: .flv
- Parámetros necesarios: ninguno
- Parámetros opcionales: ninguno
- Consideraciones de codificación: los archivos FLV son binarios; algunas aplicaciones pueden necesitar establecer el subtipo application/octet-stream.
- Problemas de seguridad: ninguno
- Especificación publicada: www.adobe.com/go/video_file_format_es

Microsoft ha cambiado la forma en la que se controlan los flujos de medios en el servidor web Servicios de Microsoft Internet Information Server (IIS) 6.0 desde las versiones anteriores. Las versiones anteriores de IIS no necesitan modificaciones para reproducir Flash Video. En IIS 6.0, servidor web predeterminado que incluye Windows 2003, el servidor necesita un tipo MIME para reconocer que los archivos FLV son flujos de medios.

Cuando los archivos SWF que transmiten archivos FLV externos se sitúan en un servidor Microsoft Windows Server® 2003 y se visualizan en un navegador, el archivo SWF se reproduce correctamente, pero el vídeo FLV no se transmite. Este problema afecta a todos los archivos FLV que se encuentran en Windows Server 2003, incluidos los archivos creados con versiones anteriores de la herramienta de edición de Flash o con Macromedia Flash Video Kit para Dreamweaver MX 2004 de Adobe. Estos archivos funcionan correctamente en otros sistemas operativos.

Para obtener información sobre la configuración de Microsoft Windows 2003 y Microsoft IIS Server 6.0 para transmitir vídeo FLV, consulte www.adobe.com/go/tn_19439_es.

Uso de archivos FLV locales en Macintosh

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si se intenta reproducir un archivo FLV local desde una unidad que no sea del sistema en un equipo Apple® Macintosh® con una ruta que utilice una barra relativa (/), el vídeo no se reproducirá. Las unidades que no son del sistema incluyen, entre otras, unidades CD-ROM, discos duros con particiones, medios de almacenamiento extraíbles y dispositivos de almacenamiento conectados.

Nota: *el motivo de este error es una limitación del sistema operativo y no de Flash Player o AIR.*

Para que un archivo FLV se reproduzca en una unidad que no sea del sistema en Macintosh, hay que hacer referencia a la misma con una ruta absoluta mediante la notación basada en dos puntos (:) en lugar de la basada en barras (/). La lista siguiente muestra la diferencia de los dos tipos de notación:

- Notación basada en barras**:** myDrive/myFolder/myFLV.flv

- Notación basada en dos puntos**:** (Mac OS®) myDrive:myFolder:myFLV.flv

Ejemplo de vídeo: Video Jukebox

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el ejemplo siguiente se crea un jukebox de vídeo que carga dinámicamente una lista de vídeos que se reproducirán en orden secuencial. Con ello, se crea una aplicación que permite que el usuario examine una serie de tutoriales de vídeo, o tal vez especifica los anuncios que se deben reproducir antes de publicar el vídeo solicitado del usuario. Este ejemplo muestra las siguientes características de ActionScript 3.0:

- Actualizar una cabeza lectora según el progreso de reproducción de un archivo de vídeo
- Detectar y analizar los metadatos de un archivo de vídeo
- Controlar códigos específicos en un objeto NetStream
- Cargar, reproducir, pausar y detener un archivo FLV cargado dinámicamente
- Cambiar el tamaño de un objeto Video en la lista de visualización según los metadatos del objeto NetStream

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Video Jukebox se encuentran en la carpeta Samples/VideoJukebox. La aplicación consta de los siguientes archivos:

Archivo	Descripción
VideoJukebox.fla o VideoJukebox.mxml	Archivo principal de la aplicación Flex (MXML) o Flash (FLA).
VideoJukebox.as	Clase que proporciona la funcionalidad principal de la aplicación.
playlist.xml	Archivo que muestra los archivos de vídeo que se cargarán en el jukebox de vídeo.

Carga de un archivo de lista de reproducción de vídeo externo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El archivo playlist.xml externo especifica los vídeos que se cargarán y el orden en que se reproducirán. Para cargar el archivo XML, se debe utilizar un objeto URLLoader y un objeto URLRequest, tal como se muestra en el código siguiente:

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Este código se coloca en el constructor de la clase VideoJukebox para que el archivo se cargue antes de ejecutar otro código. En cuanto el archivo XML termine de cargarse, se llama al método `xmlCompleteHandler()`, que analiza el archivo externo en un objeto XML, tal como se muestra en el código siguiente:

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

El objeto XML `playlist` contiene el código XML sin procesar del archivo externo, mientras que videosXML es un objeto XMLList que solo incluye los nodos de vídeo. En el siguiente fragmento se muestra un archivo playlist.xml de ejemplo:

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

Por último, el método `xmlCompleteHandler()` llama al método `main()`, que configura las diferentes instancias de componente de la lista de reproducción, así como los objetos NetConnection y NetStream, que se utilizan para cargar los archivos FLV externos.

Creación de la interfaz de usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear la interfaz de usuario, es necesario arrastrar cinco instancias de Button a la lista de visualización y asignarles los nombres de instancia siguientes: `playButton`, `pauseButton`, `stopButton`, `backButton` y `forwardButton`.

Para cada una de estas instancias de Button, deberá asignar un controlador para el evento `click`, tal como se muestra en el fragmento siguiente:

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

El método `buttonClickHandler()` utiliza una sentencia switch para determinar la instancia de Button en la que se ha hecho clic, tal como se muestra en el código siguiente:

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

A continuación, añada una instancia de Slider a la lista de visualización y asígnele el nombre `volumeSlider`. En el código siguiente se establece la propiedad `liveDragging` de la instancia de Slider en `true` y se define un detector de eventos para el evento `change` de dicha instancia:

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

Añada una instancia de ProgressBar a la lista de visualización y asígnele el nombre `positionBar`. Establezca su propiedad `mode` en manual, tal como se muestra en el fragmento siguiente:

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Finalmente, añada una instancia de Label a la lista de visualización y asígnele el nombre `positionLabel`. La instancia de Timer establecerá el valor de la instancia Label.

Detección de los metadatos de un objeto Video

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando Flash Player encuentra metadatos para los vídeos cargados, se llama al controlador callback `onMetaData()` en la propiedad `client` del objeto NetStream. En el código siguiente se inicializa un objeto y se configura el controlador callback especificado:

```
client = new Object();
client.onMetaData = metadataHandler;
```

El método `metadataHandler()` copia sus datos en la propiedad meta, definida anteriormente en el código. Esto permite acceder a los metadatos del vídeo actual en cualquier momento y en toda la aplicación. A continuación, se ajusta el tamaño del objeto Video del escenario para que coincida con las dimensiones que devuelven los metadatos. Por último, se desplaza la instancia de ProgressBar positionBar, cuyo tamaño se ajusta según el tamaño del vídeo que se reproduce en ese momento. El código siguiente contiene todo el método `metadataHandler()`:

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

Carga dinámica de un vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para cargar dinámicamente cada uno de los vídeos, la aplicación utiliza un objeto NetConnection y un objeto NetStream. En el código siguiente se crea un objeto NetConnection y se pasa el valor `null` al método `connect()`. Si se especifica `null`, Flash Player se conecta a un vídeo del servidor local en lugar de conectarse a un servidor, como Flash Media Server.

En el código siguiente se crean las instancias de NetConnection y NetStream, se define un detector de eventos para el evento `netStatus` y se asigna el objeto `client` a la propiedad `client`.

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

Se llama al método `netStatusHandler()` cuando se cambia el estado del vídeo. Esto también se aplica cuando un vídeo inicia o detiene la reproducción, almacena en búfer, o bien si no se encuentra un flujo de vídeo. El código siguiente muestra el evento `netStatusHandler()`:

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

El código anterior evalúa la propiedad de código del objeto info y filtra si el código es "NetStream.Play.Start", "NetStream.Play.StreamNotFound" o "NetStream.Play.Stop". Los otros códigos se omitirán. Si el objeto NetStream inicia el código, se inicia la instancia de Timer que actualiza la cabeza lectora. Si el objeto NetStream no se puede encontrar o se detiene, la instancia de Timer se detiene y la aplicación intenta reproducir el siguiente vídeo de la lista de reproducción.

Cada vez que se ejecuta Timer, la instancia de ProgressBar `positionBar` actualiza su posición actual llamando al método `setProgress()` de la clase ProgressBar, y la instancia de Label `positionLabel` se actualiza con el tiempo transcurrido y el tiempo total del vídeo actual.

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

Control del volumen de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede controlar el volumen del vídeo cargado dinámicamente estableciendo la propiedad `soundTransform` del objeto NetStream. La aplicación de jukebox de vídeo permite modificar el nivel de volumen cambiando el valor `volumeSlider` de la instancia Slider. En el código siguiente se muestra cómo se puede cambiar el nivel de volumen asignando el valor del componente Slider a un objeto SoundTransform, que se establece en la propiedad `soundTransform` del objeto NetStream:

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

Control de la reproducción de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El resto de la aplicación controla la reproducción de vídeo cuando este alcanza el final del flujo de vídeo, o bien cuando el usuario salta al vídeo anterior o siguiente.

En el método siguiente se recupera el URL de vídeo del objeto XMLList para el índice seleccionado en ese momento:

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

El método `playVideo()` llama al método `play()` del objeto NetStream para cargar el vídeo seleccionado actualmente:

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

El método `playPreviousVideo()` reduce el índice de vídeo actual, llama al método `playVideo()` para cargar el nuevo archivo de vídeo y establece la instancia de ProgressBar en visible:

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

El método final, `playNextVideo()`, incrementa el índice de vídeo y llama al método `playVideo()`. Si el vídeo actual es el último de la lista de reproducción, se llama al método `clear()` del objeto Video y la propiedad `visible` de la instancia de ProgressBar se establece en `false`:

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

Uso de la clase StageVideo para la presentación con aceleración por hardware

Flash Player 10.2 y posterior

Flash Player optimiza el rendimiento del vídeo utilizando la aceleración de hardware para la descodificación H.264. El rendimiento se puede mejorar más con el uso de la API de StageVideo. El vídeo de escenario permite que la aplicación aproveche la presentación acelerada por hardware.

Entre los motores de ejecución que admiten la API de StageVideo se incluyen:

- Flash Player 10.2 y posterior

Nota: *en Flash Player 11.4/AIR 3.4 y versión posterior, puede utilizar la entrada de la cámara con StageVideo.*

Para obtener el código fuente descargable y detalles adicionales para la función de vídeo de escenario, consulte Getting Started with Stage Video (Introducción a Stage Video; en inglés).

Para obtener un tutorial de inicio rápido sobre StageVideo, consulte Working with Stage Video (Trabajo con Stage Video; en inglés).

Aceleración de hardware utilizando StageVideo

La presentación acelerada por hardware, que incluye escala de vídeo, conversión de color y fusión, mejora las ventajas de rendimiento de la descodificación acelerada de hardware. En los dispositivos que ofrecen aceleración (hardware) con GPU, se puede utilizar un objeto flash.media.StageVideo para procesar el vídeo directamente en el hardware del dispositivo. El procesamiento directo libera la CPU de realizar otras tareas mientras la GPU representa el vídeo. Por otra parte, la clase heredada Video suele utilizar la presentación de software. La presentación con software se produce en la CPU y consume una parte importante de recursos del sistema.

Actualmente, muy pocos dispositivos proporciona una aceleración con GPU completa. Sin embargo, Stage Video permite que la aplicación aproveche al máximo cualquier aceleración de hardware que esté disponible.

La clase StageVideo no hace obsoleta a la clase Video. Al trabajar de forma conjunta, estas dos clases ofrecen la experiencia de reproducción de vídeo óptima que permiten los recursos del dispositivo en cualquier momento. La aplicación aprovecha la aceleración de hardware detectando los eventos adecuados y alternando entre StageVideo y Video según sea necesario.

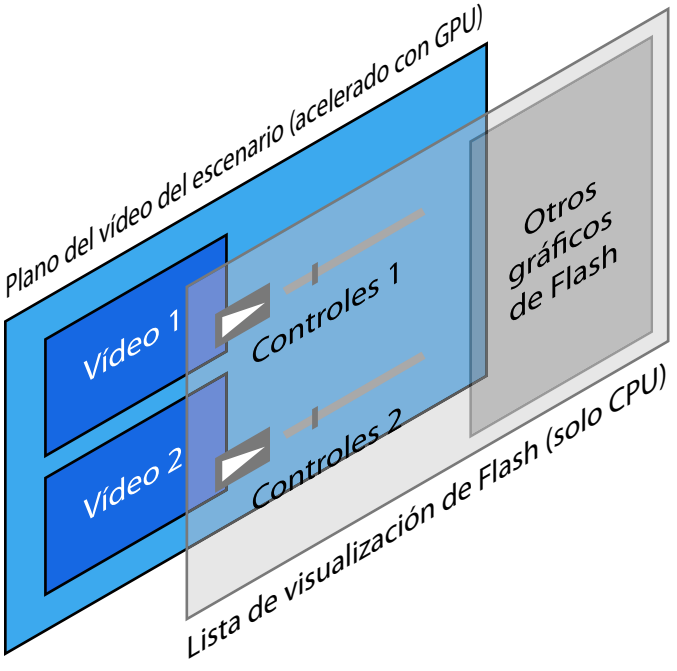
La clase StageVideo impone determinadas restricciones en el uso del vídeo. Antes de implementar StageVideo, consulte las directrices y asegúrese de que la aplicación las acepte. Si acepta las restricciones, utilice la clase StageVideo siempre que Flash Player detecte que la representación acelerada con hardware esté disponible. Consulte "Directrices y limitaciones" en la página 536.

Planos paralelos: lista de visualización de Flash y Stage video

Con el modelo de vídeo de escenario, Flash Player puede separar el vídeo de la lista de visualización. Flash Player divide la visualización de la composición entre dos planos ordenados por z:

Plano de Stage Video El plano de Stage Video se sitúa en el fondo. Solo muestra el vídeo con aceleración por hardware. Debido a este diseño. el plano no está disponible si la aceleración de hardware no se admite ni está disponible en el dispositivo. En ActionScript, los objetos StageVideo administran los vídeos reproducidos en el plano de Stage Video.

Plano de lista de visualización de Flash Las entidades de la lista de visualización de Flash se componen en el plano situado frente al plano de Stage Video. Las entidades de la lista de visualización incluyen cualquier elemento que represente el motor de ejecución, incluyendo controles de reproducción. Si la aceleración de hardware no está disponible, los vídeos se puede reproducir solo en este plano, utilizando el objeto de la clase Video. Stage Video siempre se reproduce tras los gráficos de la lista de visualización de Flash.



Planos de visualización de vídeo

El objeto StageVideo aparece en una región de la pantalla rectangular alineada con la ventana y no girada. No se pueden disponer en capas los objetos tras el plano de vídeo del escenario. Sin embargo, el plano de la lista de visualización de Flash se puede utilizar para poner en capa los demás gráficos en la parte superior del plano de vídeo del escenario. Stage Video se ejecuta al mismo tiempo que la lista de visualización de De este modo, es posible utilizar los dos mecanismos de forma conjunta para crear un efecto visual unificado que utilice dos planos discretos. Por ejemplo, se puede utilizar el plano frontal para los controles de reproducción que funcionan en el vídeo de escenario que se ejecuta en segundo plano.

Códec H.264 y Stage Video

En las aplicaciones de Flash Player, la implementación de la aceleración con hardware de vídeo implica dos pasos:

1 Codificación de vídeo como H.264

2 Implementación de la API de StageVideo

Para obtener mejores resultados, realice ambos pasos. El códec H.264 permite aprovechar al máximo la aceleración con hardware, desde la descodificación de vídeo hasta la presentación.

Stage video elimina la lectura de GPU a CPU. Es decir, la GPU ya no vuelve a enviar fotogramas descodificados a la CPU para la composición con los objetos de la lista de visualización. En cambio, la GPU muestra los fotogramas descodificados y procesados directamente en la pantalla, tras los objetos de la lista de visualización. Esta técnica reduce el uso de memoria y CPU y también proporciona una mejor fidelidad de píxel.

Más temas de ayuda

"Aspectos básicos de los formatos de vídeo" en la página 495

Directrices y limitaciones

Cuando el vídeo se ejecuta en modo de pantalla completa, stage video siempre está disponible si el dispositivo admite la aceleración con hardware. Sin embargo, Flash Player, también se ejecuta en un navegador. En el contexto de navegador, la configuración de `wmode` afecta a la disponibilidad de Stage Video. Intente utilizar `wmode="direct"` en todo momento si desea usar Stage Video. Stage Video no admite otra configuración de `wmode` cuando no está modo de pantalla completa. Esta restricción implica que, en tiempo de ejecución, Stage Video puede vacilar de forma imprevisible entre estar o no disponible. Por ejemplo, si el usuario sale del modo de pantalla completa durante la ejecución de Stage Video, el contexto del vídeo vuelve a cambiar al navegador. Si el parámetro `wmode` del navegador no es establece en `"direct"`, Stage Video puede dejar de estar disponible de repente. Flash Player comunica los cambios de contexto de reproducción a las aplicaciones mediante un conjunto de eventos. Si implementa la API de StageVideo, conserve un objeto Video como copia de seguridad cuando stage video deje de estar disponible.

Debido a su relación directa con el hardware, Stage Video limita algunas funciones de vídeo. Stage Video impone las siguientes limitaciones:

- Para cada archivo SWF, Flash Player limita el número de objetos StageVideo que pueden mostrar vídeos al mismo tiempo a cuatro. Sin embargo, el límite real puede ser más bajo, dependiendo de los recursos de hardware del dispositivo.
- La duración del vídeo no está sincronizada con el calendario del contenido de que muestra el motor de ejecución.
- El área de la pantalla de vídeo solo puede ser un rectángulo. No se puede utilizar áreas de visualización más avanzadas, como formas elípticas o irregulares.
- No se puede girar el vídeo.
- No se puede almacenar en caché de mapa de bits el vídeo ni utilizar el objeto BitmapData para acceder al mismo.
- No se pueden aplicar filtros al vídeo.
- No se pueden aplicar transformaciones de color al vídeo.
- No se puede aplicar un valor alfa para el vídeo.
- Los modos de fusión que se aplican a los objetos del plano de la lista de visualización no se aplican a stage video.
- Se puede colocar el vídeo únicamente en los límites de píxeles.

- Aunque la representación con GPU es la mejor disponible para el hardware del dispositivo determinado, no es 100% "idéntico de píxeles" en los dispositivos. Se producen ligeras variaciones debido a las diferencias de plataforma y controlador.
- Algunos dispositivos no admiten todos los espacios de color necesarios. Por ejemplo, algunos dispositivos no son compatibles con el estándar H.264, BT.709. En estos casos, se puede utilizar BT.601 para una visualización rápida.
- Stage Video no se puede emplear con la configuración de WMODE como, por ejemplo, normal, opaco o transparente. Stage Video solo admite `WMODE=direct` cuando no está en modo de pantalla completa. WMODE no surte ningún efecto en Safari 4 o posterior ni IE 9 o posterior.

En la mayoría de los casos, estas limitaciones no afectan a las aplicaciones de reproducción de vídeo. Si acepta estas limitaciones, utilice Stage Video siempre que sea posible.

Más temas de ayuda

"Trabajo con el modo de pantalla completa" en la página 171

Uso de las API de StageVideo

Stage Video es un mecanismo del motor de ejecución que mejora la reproducción de vídeo y el rendimiento del dispositivo. El motor de ejecución crea y mantiene este mecanismo; como desarrollador, su función consiste en configurar la aplicación para aprovechar este recurso.

Para utilizar Stage Video, se implementa una arquitectura de controladores de eventos que detectan el momento en que Stage Video está o no disponible. Cuando se recibe una notificación de que Stage Video está disponible, se recupera un objeto StageVideo de la propiedad `Stage.stageVideos`. El motor de ejecución llena este objeto Vector con uno o varios objetos StageVideo. Posteriormente se puede utilizar uno de los objetos StageVideo proporcionados, en lugar de un objeto Video, para reproducir el vídeo.

En Flash Player,cuando se reciba una notificación de que Stage Video ya no está disponible, cambie el flujo de vídeo al objeto Video.

Nota: *no se pueden crear objetos StageVideo.*

Propiedad Stage.stageVideos

La propiedad `Stage.stageVideos` es un objeto Vector que permite el acceso a las instancias de StageVideo. Este vector puede incluir hasta cuatro objetos StageVideo, dependiendo de los recursos del sistema y el hardware. Los dispositivos móviles se pueden limitar a uno o ninguno.

Cuando Stage Video no está disponible, este vector no contiene objetos. Para evitar errores en tiempo de ejecución, compruebe que solo accede a los miembros de este vector cuando el evento `StageVideoAvailability` más reciente indique que Stage Video está disponible.

Eventos StageVideo

La arquitectura de la API de StageVideo proporciona los siguientes eventos:

StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY Enviado cuando cambia la propiedad `Stage.stageVideos`. La propiedad `StageVideoAvailabilityEvent.availability` indica `AVAILABLE` (DISPONIBLE) o `UNAVAILABLE` (NO DISPONIBLE). Utilice este evento para determinar si la propiedad `stageVideos` contiene algún objeto StageVideo, en lugar de comprobar directamente la longitud del vector `Stage.stageVideos`.

StageVideoEvent.RENDER_STATE Enviado cuando un objeto NetStream o Camera se ha añadido a un objeto StageVideo y se está reproduciendo. Indica el tipo de descodificación que se está utilizando: hardware, software o no

disponible (no se muestra nada). El destino del evento contiene las propiedades `videoWidth` y `videoHeight` que son seguras para su uso en el cambio de tamaño de la ventana gráfica de vídeo.

Importante: *las coordenadas obtenidas del objeto de destino StageVideo utilizan las coordenadas de Stage, ya que no forman parte de la lista de visualización estándar.*

VideoEvent.RENDER_STATE Enviado cuando se está utilizando un objeto Video. Indica si se está utilizando la descodificación con aceleración por hardware o software. Si este evento indica descodificación acelerada por hardware, cambie a un objeto StageVideo, si es posible. El destino del evento Video contiene las propiedades `videoWidth` y `videoHeight` que son seguras para su uso a la hora de cambiar el tamaño de la ventana gráfica de vídeo.

Flujo de trabajo para implementar la función StageVideo

Siga estos pasos de nivel superior para implementar la función StageVideo:

1. Detecte el evento `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` para saber cuándo cambia el vector `Stage.stageVideos`. Consulte "Uso del evento StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY" en la página 539.
2. Si el evento `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` informa de que Stage Video está disponible, utilice el objeto vectorial `Stage.stageVideos` dentro del controlador de eventos para acceder a un objeto StageVideo.
3. Asocie un objeto NetStream con `StageVideo.attachNetStream()` o un objeto Camera con `StageVideo.attachCamera()`.
4. Reproduzca el vídeo con `NetStream.play()`.
5. Detecte el evento `StageVideoEvent.RENDER_STATE` en el objeto StageVideo para determinar el estado de reproducción del vídeo. La recepción de este evento también indica que las propiedades width y height del vídeo se han inicializado o han cambiado. Consulte "Uso de los eventos StageVideoEvent.RENDER_STATE y VideoEvent.RENDER_STATE" en la página 541.
6. Detecte el evento `VideoEvent.RENDER_STATE` en el objeto Video. Este evento proporciona los mismos estados que `StageVideoEvent.RENDER_STATE`, de modo que también puede utilizarlo para determinar si la aceleración por GPU está disponible. La recepción de este evento también indica que las propiedades width y height del vídeo se han inicializado o han cambiado. Consulte "Uso de los eventos StageVideoEvent.RENDER_STATE y VideoEvent.RENDER_STATE" en la página 541.

Inicialización de los detectores de eventos StageVideo

Configure los detectores StageVideoAvailabilityEvent y VideoEvent durante la inicialización de la aplicación. Por ejemplo, puede inicializar estos detectores en el controlador de eventos `flash.events.Event.ADDED_TO_STAGE`. Este evento garantiza que la aplicación se encuentra visible en el escenario:

```
public class SimpleStageVideo extends Sprite
    private var nc:NetConnection;
    private var ns:NetStream;

    public function SimpleStageVideo()
    {
        // Constructor for SimpleStageVideo class
        // Make sure the app is visible and stage available
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }

    private function onAddedToStage(event:Event):void
    {
        //...
        // Connections
        nc = new NetConnection();
        nc.connect(null);
        ns = new NetStream(nc);
        ns.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
        ns.client = this;

        // Screen
        video = new Video();
        video.smoothing = true;

        // Video Events
        // the StageVideoEvent.STAGE_VIDEO_STATE informs you whether
        // StageVideo is available
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
            onStageVideoState);
        // in case of fallback to Video, listen to the VideoEvent.RENDER_STATE
        // event to handle resize properly and know about the acceleration mode running
        video.addEventListener(VideoEvent.RENDER_STATE, videoStateChange);
        //...
    }
```

Uso del evento StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY

En el controlador `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`, decida si utilizar un objeto Video o StageVideo en función de la disponibilidad de StageVideo. Si la propiedad `StageVideoAvailabilityEvent.availability` se establece en `StageVideoAvailability.AVAILABLE`, utilice StageVideo. En este caso, se puede confiar en el vector Stage.stageVideos para que incluya uno o varios objetos StageVideo. Obtenga un objeto StageVideo de la propiedad `Stage.stageVideos` y añada el objeto NetStream al mismo. Debido a que el objeto StageVideo siempre aparece en segundo plano, elimine cualquier objeto Video existente(siempre en primer plano). Este controlador de eventos también se emplea para añadir un detector para el evento `StageVideoEvent.RENDER_STATE`.

Si la propiedad `StageVideoAvailabilityEvent.availability` se establece en `StageVideoAvailability.UNAVAILABLE`, no utilice StageVideo ni acceda al vector `Stage.stageVideos`. En este caso, añada el objeto NetStream a un objeto Video. Finalmente, añada el objeto StageVideo o Video al escenario y llame a `NetStream.play()`.

El siguiente código muestra cómo controlar el evento `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`:

```
private var sv:StageVideo;
private var video:Video;

private function onStageVideoState(event:StageVideoAvailabilityEvent):void
{
    // Detect if StageVideo is available and decide what to do in toggleStageVideo
    toggleStageVideo(event.availability == StageVideoAvailability.AVAILABLE);
}

private function toggleStageVideo(on:Boolean):void
{
    // To choose StageVideo attach the NetStream to StageVideo
    if (on)
    {
        stageVideoInUse = true;
        if ( sv == null )
        {
            sv = stage.stageVideos[0];
            sv.addEventListener(StageVideoEvent.RENDER_STATE, stageVideoStateChange);
                sv.attachNetStream(ns);
        }

        if (classicVideoInUse)
        {
            // If you use StageVideo, remove from the display list the
            // Video object to avoid covering the StageVideo object
            // (which is always in the background)
            stage.removeChild ( video );
            classicVideoInUse = false;
        }
    } else
    {
        // Otherwise attach it to a Video object
        if (stageVideoInUse)
            stageVideoInUse = false;
        classicVideoInUse = true;
        video.attachNetStream(ns);
        stage.addChildAt(video, 0);
    }

    if ( !played )
    {
        played = true;
        ns.play(FILE_NAME);
    }
}
```

Importante: *la primera vez que una aplicación accede al elemento vectorial en Stage.stageVideos[0], el rectángulo predeterminado se establece en 0,0,0,0 y las propiedades pan y zoom utilizan los valores predeterminados. Restablezca siempre estos valores a la configuración preferida. Se pueden utilizar las propiedades* `videoWidth` *y* `videoHeight` *del destino del evento* `StageVideoEvent.RENDER_STATE` *o* `VideoEvent.RENDER_STATE` *para calcular las dimensiones de la ventana gráfica del vídeo.*

Descargue el código fuente completo para esta misma aplicación en Getting Started with Stage Video (Introducción a Stage Video; en inglés).

Uso de los eventos StageVideoEvent.RENDER_STATE y VideoEvent.RENDER_STATE

Los objetos StageVideo y Video envían eventos que informan a las aplicaciones del momento en que cambia el entorno de representación. Estos eventos son `StageVideoEvent.RENDER_STATE` y `VideoEvent.RENDER_STATE`.

Un objeto StageVideo o Video distribuye un evento de estado de representación cuando un objeto NetStream se añade y comienza a reproducirse. También envía este evento cuando cambia el entorno de representación; por ejemplo, cuando cambia de tamaño la ventana gráfica del vídeo. Utilice estas notificaciones para restablecer la ventana gráfica a los valores actuales `videoHeight` y `videoWidth` del objeto de destino del evento.

Los estados de representación indicados incluyen:

- `RENDER_STATUS_UNAVAILABLE`
- `RENDER_STATUS_SOFTWARE`
- `RENDER_STATUS_ACCELERATED`

Los estados de representación indican el momento en que se está utilizando la descodificación acelerada con hardware, independientemente de qué clase esté reproduciendo el vídeo actualmente. Compruebe la propiedad `StageVideoEvent.status` para saber si la descodificación necesaria está disponible. Si esta propiedad se define como "unavailable", el objeto StageVideo no puede reproducir el vídeo. Este estado requiere que vuelva a añadir inmediatamente el objeto NetStream a un objeto Video. Otros estados informan a la aplicación de las condiciones de representación actuales.

En la siguiente tabla se describen las implicaciones de todos los valores del estado de representación para los objetos StageVideoEvent y VideoEvent en Flash Player:

	VideoStatus.ACCELERATED	**VideoStatus.SOFTWARE**	**VideoStatus.UNAVAILABLE**
StageVideoEvent	Tanto la descodificación como la presentación se producen en el hardware. (El mejor rendimiento.)	Representación en hardware y descodificación en software. (Rendimiento aceptable.)	No hay recursos de GPU disponibles para representar el vídeo y no se muestra nada. **Retroceso a un objeto Video.**
VideoEvent	Presentación en software y descodificación en hardware. (Rendimiento aceptable solo en un sistema de escritorio moderno. Rendimiento de pantalla completa degradado.)	Presentación en software y descodificación en software. (En lo que respecta al rendimiento en el peor caso. Rendimiento de pantalla completa degradado.)	(N/D)

Espacios de color

Stage Video utiliza las capacidades de hardware subyacentes para admitir espacios de color. El contenido SWF puede proporcionar metadatos que indican su espacio de color preferido. Sin embargo, el hardware de los gráficos del dispositivo determina si ese espacio de color se puede utilizar. Un dispositivo puede admitir varios espacios de color, mientras que otros no admiten ninguno. Si el hardware no admite el espacio de color solicitado, Flash Player intenta encontrar el que más coincida entre los espacios de color compatibles.

Para consultar qué espacios de color admite el hardware, utilice la propiedad `StageVideo.colorSpaces`. Esta propiedad devuelve la lista de espacios de color admitidos en un vector String:

```
var colorSpace:Vector.<String> = stageVideo.colorSpaces();
```

Para saber qué espacio de color está utilizando el vídeo que se está reproduciendo, compruebe la propiedad `StageVideoEvent.colorSpace`. Compruebe esta propiedad en su controlador de eventos para el evento `StageVideoEvent.RENDER_STATE`:

```
var currColorSpace:String;

//StageVideoEvent.RENDER_STATE event handler
private function stageVideoRenderState(event:Object):void
{
    //...
    currColorSpace = (event as StageVideoEvent).colorSpace;
    //...
}
```

Si Flash Player no puede encontrar ningún sustituto para un espacio de color no compatible, Stage Video usa el espacio de color predeterminado BT.601. Por ejemplo, los flujos de vídeo con codificación H.264 suelen utilizar el espacio de color BT.709. Si el hardware del dispositivo no admite BT.709, la propiedad `colorSpace` devuelve `"BT601"`. Un valor `StageVideoEvent.colorSpace` de `"unknown"` indica que el hardware no proporciona ningún medio para consultar el espacio de color.

Si la aplicación considera inaceptable el espacio de color actual, es posible optar por cambiar de un objeto StageVideo a un objeto Video. La clase Video es compatible con todos los espacios de color mediante la composición de software.

Capítulo 26: Trabajo con cámaras

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una cámara conectada al ordenador del usuario puede servir de origen de datos de vídeo, que a su vez se pueden mostrar y manipular con ActionScript. La clase Camera es el mecanismo integrado en ActionScript para trabajar con una cámara de ordenador o de dispositivo.

En dispositivos móviles, es posible utilizar también la clase CameraUI. La clase CameraUI inicia una aplicación de cámara independiente para que el usuario pueda capturar imágenes fijas o vídeo. Cuando el usuario termina, la aplicación puede acceder a la imagen o al vídeo a través de un objeto MediaPromise.

Más temas de ayuda

Christian Cantrell: How to use CameraUI in a Cross-platform Way (Cómo utilizar CameraUI en varias plataformas; en inglés)

Michaël CHAIZE: Android, AIR and the Camera

Aspectos básicos de la clase Camera

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El objeto Camera permite conectar la cámara local del usuario y difundir vídeo localmente (de vuelta al usuario) o de forma remota a un servidor (como Flash Media Server).

Mediante la clase Camera, se pueden utilizar los siguientes tipos de información sobre la cámara del usuario:

- Las cámaras instaladas en el ordenador o el dispositivo del usuario que están disponibles
- Si la cámara está instalada
- Si Flash Player tiene permiso para acceder a la cámara del usuario
- La cámara que está activa en ese momento
- La anchura y la altura del vídeo que se captura

La clase Camera incluye varios métodos y propiedades útiles para utilizar objetos Camera. Por ejemplo, la propiedad `Camera.names` estática contiene un conjunto de nombres de cámara instalados en ese momento en el ordenador del usuario. Asimismo, se puede utilizar la propiedad `name` para mostrar el nombre de la cámara activa en ese momento.

Nota: *al transmitir vídeo de cámara en la red, siempre se deben gestionar las interrupciones de red. Las interrupciones de red pueden suceder por varios motivos, especialmente en los dispositivos móviles.*

Visualización del contenido de la cámara en pantalla

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La conexión a una cámara puede requerir menos código que utilizar las clases NetConnection y NetStream para cargar un vídeo. Asimismo, la clase Camera puede plantear problemas rápidamente, ya que se necesita el permiso de un usuario para que Flash Player se conecte a su cámara antes de que poder acceder a ella.

En el código siguiente se muestra cómo se puede utilizar la clase Camera para conectarse a la cámara local de un usuario:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Nota: *la clase Camera no tiene un método constructor. Para crear una nueva instancia de Camera, se utiliza el método* `Camera.getCamera()` *estático.*

Diseño de la aplicación de cámara

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al programar una aplicación que se conecta a la cámara de un usuario, el código debe hacer lo siguiente:

- Comprobar si el usuario tiene una cámara instalada en ese momento. Gestione el caso donde no haya cámaras disponibles.
- Solo en Flash Player, compruebe si el usuario ha permitido el acceso a la cámara de forma explícita. Por motivos de seguridad, el reproductor muestra el cuadro de diálogo Configuración de Flash Player, en el que el usuario permite o deniega el acceso a su cámara. Esto impide que Flash Player se conecte a la cámara de un usuario y difunda un flujo de vídeo sin su permiso. Si el usuario permite el acceso haciendo clic en el botón correspondiente, la aplicación puede conectarse a la cámara. De lo contrario, la aplicación no podrá acceder a dicha cámara. Las aplicaciones siempre deben controlar ambas situaciones adecuadamente.
- Solo para AIR, compruebe si la clase Camera se admite en los perfiles del dispositivo compatibles con la aplicación.
- La clase Camera no se admite en los navegadores móviles.
- La clase Camera no es compatible en las aplicaciones de AIR móviles que utilizan el modo de representación con GPU.
- En dispositivos móviles, solo una cámara puede estar activa al mismo tiempo.

Más temas de ayuda

Christophe Coenraets: Multi-User Video Tic-Tac-Toe (en inglés)

Mark Doherty: Aplicación Android Radar (código fuente)

Conexión a la cámara de un usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El primer paso de la conexión a la cámara de un usuario consiste en crear una nueva instancia de cámara; se crea una variable de tipo Camera, que se inicializa al valor devuelto del método `Camera.getCamera()` estático.

El siguiente paso es crear un nuevo objeto Video y asociarle el objeto Camera.

El tercer paso consiste en añadir el objeto Video a la lista de visualización. Los pasos 2 y 3 son necesarios, ya que la clase Camera no amplía la clase DisplayObject y no se puede añadir directamente a la lista. Para mostrar el vídeo capturado de la cámara, se crea un nuevo objeto Video y se llama al método `attachCamera()`.

En el código siguiente se muestran estos tres pasos:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Se debe tener en cuenta que si un usuario no dispone de una cámara instalada, la aplicación no mostrará nada.

En situaciones reales, deben llevarse a cabo pasos adicionales para la aplicación. Consulte "Comprobación de que las cámaras están instaladas" en la página 545 y "Detección de permisos para el acceso a una cámara" en la página 546 para obtener más información.

Más temas de ayuda

Lee Brimelow: How to access the camera on Android devices (Cómo acceder a la cámara en los dispositivos Android; en inglés)

Comprobación de que las cámaras están instaladas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Antes de intentar utilizar métodos o propiedades en una instancia de cámara, tal vez se desee comprobar que el usuario tiene una cámara instalada. Existen dos maneras de comprobar si el usuario tiene una cámara instalada:

- Comprobar la propiedad estática `Camera.names` que contiene un conjunto de nombres de cámara disponibles. Normalmente, este conjunto tiene una cadena o ninguna, puesto que la mayoría de usuarios no disponen de más de una cámara instalada a la vez. En el código siguiente se muestra cómo se puede comprobar la propiedad `Camera.names` para ver si el usuario tiene cámaras disponibles:

```
if (Camera.names.length > 0)
{
    trace("User has at least one camera installed.");
    var cam:Camera = Camera.getCamera(); // Get default camera.
}
else
{
    trace("User has no cameras installed.");
}
```

- Comprobar el valor devuelto del método estático `Camera.getCamera()`. Si no hay cámaras instaladas o disponibles, este método devuelve `null`; de lo contrario, devuelve una referencia a un objeto Camera. En el código siguiente se muestra cómo se puede comprobar el método `Camera.getCamera()` para ver si el usuario tiene cámaras disponibles:

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

Debido a que la clase Camera no amplía la clase DisplayObject, no se puede añadir directamente a la lista de visualización utilizando el método `addChild()`. Para mostrar el vídeo capturado de la cámara, se debe crear un nuevo objeto Video y llamar al método `attachCamera()` en la instancia de Video.

Este fragmento de código muestra cómo se puede conectar la cámara, si se dispone de ella; de lo contrario, la aplicación no mostrará nada.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

Cámaras de dispositivos móviles

La clase Camera no se admite en el motor de ejecución de Flash Player en navegadores para móvil.

En aplicaciones de AIR instaladas en dispositivos móviles es posible acceder a la cámara o a las cámaras en el dispositivo. En dispositivos móviles, puede usarse tanto la cámara trasera como la delantera, pero no al mismo tiempo. (Cuando se activa una, se desactiva la otra.) La cámara frontal se refleja horizontalmente en iOS, pero no en Android.

Detección de permisos para el acceso a una cámara

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el entorno limitado de la aplicación AIR, la aplicación puede acceder a cualquier cámara sin el permiso del usuario. Sin embargo, en Android, una aplicación debe especificar el permiso CAMERA de Android en el archivo descriptor de la aplicación.

Antes de que Flash Player pueda mostrar la salida de una cámara, el usuario debe permitir explícitamente a la aplicación que acceda a la cámara. Cuando se llama al método `attachCamera()`, Flash Player muestra el cuadro de diálogo Configuración de Flash Player, que pregunta al usuario si desea que Flash Player acceda a la cámara y al micrófono. Si el usuario hace clic en el botón Permitir, Flash Player muestra la salida de la cámara en la instancia de Video en el escenario. Si el usuario hace clic en el botón de rechazo, Flash Player no puede conectarse a la cámara y el objeto Video no muestra nada.

Si se desea saber si el usuario ha permitido o rechazado el acceso a la cámara, se puede detectar el evento `status` (`StatusEvent.STATUS`) de la cámara, tal y como se muestra en el código siguiente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

La función `statusHandler()` se llama cuando el usuario hace clic en los botones para permitir o denegar el acceso. Se puede detectar el botón en el que ha hecho clic el usuario con uno de estos dos métodos:

- El parámetro `event` de la función `statusHandler()` contiene una propiedad de código que incluye la cadena "Camera.Muted" o "Camera.Unmuted". Si el valor es "Camera.Muted", el usuario ha hecho clic en el botón para denegar el acceso y Flash Player no puede acceder a la cámara. Se puede ver un ejemplo de esto en el fragmento siguiente:

  ```
  function statusHandler(event:StatusEvent):void
  {
      switch (event.code)
      {
          case "Camera.Muted":
              trace("User clicked Deny.");
              break;
          case "Camera.Unmuted":
              trace("User clicked Accept.");
              break;
      }
  }
  ```

- La clase Camera contiene una propiedad de solo lectura denominada `muted`, que especifica si el usuario ha denegado el acceso a la cámara (`true`) o lo ha permitido (`false`) en el panel Privacidad de Flash Player. Se puede ver un ejemplo de esto en el fragmento siguiente:

  ```
  function statusHandler(event:StatusEvent):void
  {
      if (cam.muted)
      {
          trace("User clicked Deny.");
      }
      else
      {
          trace("User clicked Accept.");
      }
  }
  ```

Al buscar el evento de estado que se va a distribuir, se puede escribir código que controle la aceptación o el rechazo del acceso a la cámara por parte del usuario y llevar a cambio la limpieza correspondiente. Por ejemplo, si el usuario hace clic en el botón para denegar el acceso, se puede mostrar un mensaje al usuario que indique que debe hacer clic en el botón para permitir el acceso si desea participar en un chat de vídeo. Asimismo, se puede asegurar de que el objeto Video esté eliminado de la lista de visualización para liberar recursos de sistema.

En AIR, un objeto Camera no distribuye eventos de estado, ya que el permiso para utilizar la cámara no es dinámico.

Aumento de la calidad de vídeo de la cámara

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De forma predeterminada, las nuevas instancias de la clase Video son de 320 píxeles de ancho x 240 píxeles de alto. Para aumentar la calidad de vídeo, siempre debe asegurarse de que el objeto Video coincide con las mismas dimensiones que el vídeo que devuelve el objeto Camera. Se puede obtener la anchura y la altura del objeto Camera mediante las propiedades `width` y `height` de la clase Camera. Asimismo, se pueden establecer las propiedades `width` y `height` del objeto Video para que se correspondan con las dimensiones de los objetos Camera. También es posible pasar la anchura y la altura al método constructor de la clase Video, tal como se muestra en el fragmento siguiente:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Puesto que el método `getCamera()` devuelve una referencia a un objeto Camera (o `null` si no hay cámaras disponibles), se puede acceder a los métodos y las propiedades de la cámara aunque el usuario deniegue el acceso a ella. Esto permite establecer el tamaño de la instancia de vídeo con la altura y la anchura nativas de la cámara.

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Para obtener información sobre el modo de pantalla completa, consulte la sección sobre este tema en "Configuración de las propiedades de Stage" en la página 169.

Supervisión del estado de la cámara

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Camera contiene varias propiedades que permiten controlar el estado actual del objeto Camera. Por ejemplo, el código siguiente muestra varias de las propiedades de la cámara mediante un objeto Timer y una instancia de campo de texto en la lista de visualización:

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

Cada 1/10 de segundo (100 milisegundos) se distribuye el evento `timer` del objeto Timer y la función `timerHandler()` actualiza el campo de texto de la lista de visualización.

Capítulo 27: Uso de la administración de derechos digitales

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

Adobe® Access™ es una solución de protección de contenido. Permite que los propietarios, distribuidores y publicistas de contenido consigan nuevas fuentes de ingresos al proporcionar acceso a contenido con valor añadido. Los editores utilizan Adobe Access para cifrar contenido, crear políticas y emitir licencias. Adobe Flash Player y Adobe AIR incorporan una biblioteca DRM: el módulo Adobe Access. Este módulo permite que el motor de ejecución se comunique con el servidor de licencias de Adobe Access y reproduzca contenido protegido. El motor de ejecución completa así el ciclo del contenido protegido con Adobe Access y distribuido por Flash Media Server.

Con Adobe Access, los proveedores de contenido puede suministrar contenido gratuito o de pago. Por ejemplo, un consumidor puede querer ver un programa de televisión sin publicidad. El consumidor se registra y paga al editor de contenido. Seguidamente, el consumidor introduce sus credenciales de usuario para acceder al programa y reproducirlo sin anuncios.

En otro ejemplo, un consumidor quiere ver contenido sin conexión mientras viaja sin acceso a Internet. Este flujo de trabajo sin conexión se admite en las aplicaciones de AIR. Tras registrarse y pagar al editor por el contenido, el usuario puede acceder y descargar el contenido y la aplicación de AIR asociada desde el sitio web del editor. Con la aplicación de AIR, el usuario puede ver el contenido sin conexión durante el periodo permitido. El contenido también se puede compartir con otros dispositivos del mismo grupo de dispositivos mediante dominios (**Novedad en 3.0**).

Adobe Access también admite acceso anónimo que no requiere autenticación del usuario. Por ejemplo, un editor puede usar acceso anónimo para distribuir contenido basado en publicidad. El acceso anónimo también permite que el editor tenga acceso libre al contenido actual durante un número especificado de días. El proveedor de contenido también puede especificar y restringir el tipo y la versión de reproductor que se necesita para su contenido.

Cuando un usuario intente reproducir contenido protegido en Flash Player o Adobe AIR, la aplicación debe llamar a las API DRM. Las API DRM inician el flujo de trabajo para la reproducción del contenido protegido. El motor de ejecución se pone en contacto con el servidor de licencias a través del módulo Adobe Access. El servidor de licencias autentica al usuario, si es necesario, y emite una licencia para permitir la reproducción del contenido protegido. El motor de ejecución recibe la licencia y descifra el contenido para reproducirlo.

En esta sección se describe cómo hacer posible que la aplicación pueda reproducir contenido protegido con Adobe Access. Con Adobe Access, no es necesario saber cómo descifrar contenido ni mantener políticas. Sin embargo, sí se da por hecho que se está comunicando con el servidor de licencias de Adobe Access para autenticar al usuario y obtener la licencia. También se da por hecho que está diseñando una aplicación para reproducir específicamente contenido protegido con Adobe Access.

Para información general sobre Adobe Access, incluida la creación de normas, consulte la documentación que se incluye con Adobe Access.

Más temas de ayuda

Paquete flash.net.drm

flash.net.NetConnection

flash.net.NetStream

Aspectos básicos del flujo de trabajo de contenido protegido

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

Importante: Flash Player 11.5 y versiones posteriores integra el módulo Adobe Access, por lo que el paso de actualización (llamar a `SystemUpdater.update(SystemUpdaterType.DRM)`) no es necesario. Esto incluye los siguientes navegadores y plataformas:

- Control ActiveX para Flash Player 11.5, para todas las plataforma excepto Internet Explorer en Windows 8 en procesadores Intel
- Plug-in de Flash Player 11.5, para todos los navegadores
- Adobe AIR (escritorio y móvil)

Esto significa que el paso de actualización *sigue siendo necesario* en los casos siguientes:

- Internet Explorer en Windows 8 en procesadores Intel
- Flash Player 11.4 y versiones anteriores, excepto en Google Chrome 22 y versiones posteriores (todas las plataformas) o 21 y versiones posteriores (Windows)

Nota: *sigue siendo seguro llamar a `SystemUpdater.update(SystemUpdaterType.DRM)` en un sistema con Flash Player 11.5 o versión posterior, pero no se descargará nada.*

El siguiente flujo de trabajo de alto nivel muestra cómo una aplicación puede recuperar y reproducir contenido protegido. El flujo de trabajo asume que la aplicación está diseñada específicamente para reproducir contenido protegido con Adobe Access:

1. Obtenga los metadatos del contenido.
2. Gestione las actualizaciones a Flash Player, si es necesario.
3. Compruebe si hay una licencia disponible localmente. Si es así, cárguela y vaya al paso 7. Si no, vaya al paso 4.
4. Compruebe si es necesario autenticarse. Si no lo es, vaya al paso 7.
5. Si se requiere autenticación, obtenga las credenciales para el usuario y envíelas al servidor de licencias.
6. Si se precisa registrar el dominio, únase a él (AIR 3.0 y posterior).
7. Una vez completada la autenticación, descargue la licencia del servidor.
8. Reproduzca el contenido.

Si no se produce ningún error y se autoriza correctamente al usuario a ver el contenido, el objeto NetStream distribuye un objeto DRMStatusEvent. La aplicación comienza a reproducirse después. El objeto DRMStatusEvent contiene la información de la licencia asociada, que identifica la póliza del usuario y sus permisos. Por ejemplo, contiene información relacionada con si el contenido puede verse sin conexión o cuándo caduca la licencia. La aplicación puede aprovechar estos datos para informar al usuario del estado de su póliza. Por ejemplo: la aplicación puede mostrar en una barra de estado el número de días restantes para que el usuario pueda ver el contenido.

Si el usuario tiene permiso para acceder sin conexión, la licencia se guarda en caché y el contenido cifrado se descarga en el equipo del usuario. El contenido está accesible en la duración definida en la caché de la licencia. La propiedad `detail` del evento contiene `"DRM.voucherObtained"`. La aplicación decide dónde guardar el contenido en el equipo para tenerlo a disposición fuera de línea. También se puede realizar la carga previa de licencias mediante la clase DRMManager.

Nota: *el almacenamiento en caché y la precarga de licencias se admite tanto en AIR como en Flash Player. Sin embargo, la descarga y el almacenamiento del contenido cifrado solo se admite en AIR.*

Es responsabilidad de la aplicación gestionar de forma explícita los eventos de error. Estos eventos incluyen casos en que el usuario introduce credenciales válidas pero la licencia que protege el contenido cifrado limita el acceso al contenido. Por ejemplo, un usuario autenticado no puede acceder al contenido si no ha pagado por los derechos. Este caso también puede darse si dos miembros registrados del mismo editor intentan compartir contenido por el que solo uno de ellos ha pagado. La aplicación debe informar al usuario sobre el error y ofrecer una sugerencia como alternativa. Una alternativa habitual es ofrecer instrucciones para registrarse y pagar por los derechos de visionado.

Flujo de trabajo detallado de la API

Flash Player 10.1 y posterior, AIR 2.0 y posterior

Este flujo de trabajo ofrece una vista más detallada del flujo de trabajo de contenido protegido. Este flujo de trabajo describe las API específicas utilizadas para reproducir contenido protegido con Adobe Access.

1. Con ayuda de un objeto URLLoader, cargue los bytes del archivo de metadatos del contenido protegido. Defina este objeto como una variable, por ejemplo, `metadata_bytes`.

   Todo el contenido controlador por Adobe Access tiene metadatos de Adobe Access. Cuando se empaqueta el contenido, estos metadatos se pueden guardar como un archivo de metadatos independiente (.metadata) junto con el contenido. Para obtener más información, consulte la documentación de Adobe Access.

2. Cree una instancia de DRMContentData. Coloque este código en un bloque try-catch:

   ```
   new DRMContentData(metadata_bytes)
   ```

   donde *`metadata_bytes`* es el objeto URLLoader obtenido en el paso 1.

3. (Solo Flash Player) El motor de ejecución busca el módulo Adobe Access. Si no se encuentra, se emite un error IllegalOperationError con código de error DRMErrorEvent 3344 o DRMErrorEvent 3343.

   Para gestionar este error, descargue el módulo Adobe Access con la API SystemUpdater. Una vez descargado el módulo, el objeto SystemUpdater distribuye un evento COMPLETE. Incluya un detector de eventos para este evento que vuelve al paso 2 cuando se distribuye. El siguiente código demuestra estos pasos:

   ```
   flash.system.SystemUpdater.addEventListener(Event.COMPLETE, updateCompleteHandler);
   flash.system.SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
   ```

   ```
   private function updateCompleteHandler (event:Event):void {
       /*redo step 2*/
       drmContentData = new DRMContentData(metadata_bytes);
   }
   ```

   Si se debe actualizar el propio reproductor, se distribuye un evento status. Para obtener más información sobre la gestión de este evento, consulte "Detección de un evento update" en la página 568.

   Nota: *en aplicaciones de AIR, el archivo de instalación de AIR gestiona la actualización del módulo Adobe Access y las posibles actualizaciones necesarias del motor de ejecución.*

4. Cree detectores para detectar los eventos DRMStatusEvent y DRMErrorEvent del objeto DRMManager:

   ```
   DRMManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
   DRMManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
   ```

   En el detector de DRMStatusEvent, confirme que la licencia es válida (no null). En el detector de DRMErrorEvent, gestione eventos DRMErrorEvent. Consulte "Uso de la clase DRMStatusEvent" en la página 560 y "Uso de la clase DRMErrorEvent" en la página 565.

5 Cargue la licencia necesaria para poder reproducir el contenido.

Primero, intente cargar la licencia guardada localmente para reproducir el contenido:

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.LOCAL_ONLY)
```

Una vez completada la carga, el objeto DRMManager distribuye un evento `DRMStatusEvent.DRM_Status`.

6 Si el objeto DRMVoucher no es null, la licencia es válida. Vaya directamente al paso 13.

7 Si el objeto DRMVoucher es null, compruebe el método de autenticación requerido por la póliza para este contenido. Utilice la propiedad `DRMContentData.authenticationMethod`.

8 Si el método de autenticación es `ANONYMOUS`, vaya al paso 13.

9 Si el método de autenticación es `USERNAME_AND_PASSWORD`, su aplicación debe contar con un mecanismo para que el usuario pueda introducir sus credenciales. Envíe estas credenciales al servidor de licencias para autenticar al usuario:

```
DRMManager.authenticate(metadata.serverURL, metadata.domain, username, password)
```

DRMManager distribuye un evento `DRMAuthenticationErrorEvent` si falla la autenticación y un evento `DRMAuthenticationCompleteEvent` si la autenticación se produce correctamente. Cree detectores para estos eventos.

10 Si el método de autenticación es `UNKNOWN`, se debe usar un método de autenticación personalizado. En este caso, el proveedor de contenido ha preparado la autenticación fuera de banda, sin usar las API de ActionScript 3.0. El procedimiento de autenticación personalizado debe generar un símbolo de autenticación que pueda transferirse al método `DRMManager.setAuthenticationToken()`.

11 Si la autenticación falla, la aplicación debe volver al paso 9. Asegúrese de que la aplicación cuenta con un mecanismo para gestionar y limitar el número de fallos de autenticación repetidos. Por ejemplo, tras tres intentos, muestre un mensaje al usuario indicando que la autenticación ha fallado y que no es posible visualizar el contenido.

12 Para usar el símbolo almacenado y evitar tener que pedir al usuario que introduzca sus credenciales, establezca el símbolo con el método `DRMManager.setAuthenticationToken()`. Seguidamente, descargue la licencia del servidor de licencias y reproduzca el contenido como en el paso 8.

13 (opcional) Si la autenticación se realiza correctamente, puede capturar el símbolo de autenticación (un conjunto de bytes que se almacena en memoria). Obtenga este símbolo con la propiedad `DRMAuthenticationCompleteEvent.token`. Puede almacenar y usar el símbolo de autenticación para que el usuario no tenga que introducir sus credenciales cada vez que quiera ver este contenido. El servidor de licencias determina el periodo de validez del símbolo de autenticación.

14 Si la autenticación se produce correctamente, descargue la licencia del servidor de licencias:

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.FORCE_REFRESH)
```

Una vez completada la carga, el objeto DRMManager distribuye un evento DRMStatusEvent.DRM_STATUS. Detecte este evento y, cuando se distribuya, podrá reproducir el contenido.

15 Reproduzca el vídeo creando un objeto NetStream y llamando a su método `play()` después:

```
stream = new NetStream(connection);
stream.addEventListener(DRMStatusEvent.DRM _STATUS, drmStatusHandler);
stream.addEventListener(DRMErrorEvent.DRM_ERROR, drmErrorHandler);
stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
stream.client = new CustomClient();
video.attachNetStream(stream);
stream.play(videoURL);
```

DRMContentData y objetos de sesión

Cuando se crea `DRMContentData`, se utilizará como un objeto de sesión que haga referencia al módulo DRM de Flash Player. Todas las API `DRMManager` que reciban este `DRMContentData` usarán dicho módulo DRM en particular. No obstante, hay dos API `DRMManager` que no utilizan `DRMContentData`. Son:

1. `authenticate()`
2. `setAuthenticationToken()`

Dado que no hay ningún `DRMContentData` asociado, al invocar estas API `DRMManager` se recurrirá al último módulo DRM del disco. Esto puede convertirse en un problema si se produce una actualización del módulo DRM en mitad del flujo de trabajo DRM de la aplicación. Imaginemos el siguiente contexto:

1. La aplicación crea un objeto `DRMContentDatacontentData1`, que utiliza *AdobeCP1* como módulo DRM.
2. La aplicación invoca el método `DRMManager.authenticate(contentData1.serverURL,...)`.
3. La aplicación invoca el método `DRMManager.loadVoucher(contentData1, ...)`.

Si se produce una actualización del módulo DRM antes de que la aplicación llegue al paso 2, el método `DRMManager.authenticate()` dejará de autenticar con *AdobeCP2* como módulo DRM. El método `loadVoucher()` del paso 3 fallará, ya que sigue usando *AdobeCP1* como módulo DRM. La actualización puede haberse producido por la invocación de la actualización del módulo DRM por parte de otra aplicación. Esto se puede evitar invocando la actualización del módulo DRM en el inicio de la aplicación.

Eventos relacionados con DRM

El motor de ejecución distribuye varios eventos cuando una aplicación intenta reproducir contenido protegido:

- DRMDeviceGroupErrorEvent (solo AIR), distribuido por DRMManager
- DRMAuthenticateEvent (solo AIR), distribuido por NetStream
- DRMAuthenticationCompleteEvent, distribuido por DRMManager
- DRMAuthenticationErrorEvent, distribuido por DRMManager
- DRMErrorEvent, distribuido por NetStream y por DRMManager
- DRMStatusEvent, distribuido por NetStream y por DRMManager
- StatusEvent
- NetStatusEvent. Consulte "Detección de un evento update" en la página 568

Para admitir contenido protegido con Adobe Access, añada detectores de eventos para controlar los eventos DRM.

Precarga de licencias para reproducción sin conexión

Adobe AIR 1.5 y posterior

Puede precargar las licencias necesarias para reproducir contenido protegido con Adobe Access. Las licencias precargadas permiten a los usuarios ver el contenido tengan o no una conexión a Internet. (El propio proceso de carga previa requiere una conexión a Internet.) Puede utilizar el método `preloadEmbeddedMetadata()` de la clase NetStream y la clase DRMManager para precargar licencias. En AIR 2.0 y posterior, puede utilizar un objeto DRMContentData para precargar licencias directamente. Es preferible usar esta técnica, ya que permite actualizar el objeto DRMContentData independientemente del contenido. (El método `preloadEmbeddedData()` obtiene un objeto DRMContentData del contenido.)

Uso de DRMContentData

Adobe AIR 2.0 y posterior

En los siguientes pasos se describe el flujo de trabajo para precargar la licencia de un archivo multimedia protegido mediante un objeto DRMContentData.

1. Obtenga los metadatos binarios del contenido empaquetado. Si utiliza Adobe Access Java Reference Packager, este archivo de metadatos se genera automáticamente con la extensión *.metadata*. Puede, por ejemplo, descargar estos metadatos mediante la clase URLLoader.
2. Cree un objeto DRMContentData pasando los metadatos a la función constructora:

   ```
   var drmData:DRMContentData = new DRMContentData( metadata );
   ```

3. El resto de los pasos son idénticos a los del flujo de trabajo descrito en "Aspectos básicos del flujo de trabajo de contenido protegido" en la página 552.

Uso de preloadEmbeddedMetadata()

Adobe AIR 1.5 y posterior

Los siguientes pasos describen el flujo de trabajo para precargar la licencia para un archivo multimedia protegido por DRM usando `preloadEmbeddedMetadata()`:

1. Descargue y almacene el archivo. (Los metadatos DRM solo se pueden precargar desde archivos almacenados localmente.)
2. Cree los objetos NetConnection y NetStream, proporcionando implementaciones para las funciones callback `onDRMContentData()` y `onPlayStatus()` del objeto cliente NetStream.
3. Cree un objeto NetStreamPlayOptions y establezca la propiedad `stream` en la URL del archivo de medios local.
4. Llame al método `preloadEmbeddedMetadata()` de NetStream, transmitiendo el objeto NetStreamPlayOptions e identificando el archivo de medios para analizar.
5. Si el archivo contiene metadatos DRM, se invoca la función callback `onDRMContentData()`. Los metadatos se transmiten a esta función como un objeto DRMContentData.
6. Utilice el objeto DRMContentData para obtener la licencia utilizando el método `loadVoucher()` de DRMManager.

   Si el valor de la propiedad `authenticationMethod` del objeto `DRMContentData` es `flash.net.drm.AuthenticationMethod.USERNAME_AND_PASSWORD`, debe autenticar al usuario en el servidor de derechos multimedia antes de cargar la licencia. Las propiedades `serverURL` y `domain` del objeto DRMContentData se pueden transmitir al método `authenticate()`, junto con las credenciales del usuario.
7. La función `onPlayStatus()` se invoca cuando el análisis del archivo ha finalizado. Si la función `onDRMContentData()` no se ha llamado, el archivo no contiene los metadatos necesarios para obtener una licencia. La ausencia de esta llamada también puede querer decir que Adobe Access no protege este archivo.

El siguiente ejemplo de código para AIR muestra cómo precargar una licencia para un archivo multimedia local:

```
package
{
import flash.display.Sprite;
import flash.events.DRMAuthenticationCompleteEvent;
import flash.events.DRMAuthenticationErrorEvent;
import flash.events.DRMErrorEvent;
import flash.ev ents.DRMStatusEvent;
import flash.events.NetStatusEvent;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.net.NetStreamPlayOptions;
import flash.net.drm.AuthenticationMethod;
import flash.net.drm.DRMContentData;
import flash.net.drm.DRMManager;
import flash.net.drm.LoadVoucherSetting;
public class DRMPreloader extends Sprite
{
    private var videoURL:String = "app-storage:/video.flv";
    private var userName:String = "user";
    private var password:String = "password";
    private var preloadConnection:NetConnection;
    private var preloadStream:NetStream;
    private var drmManager:DRMManager = DRMManager.getDRMManager();
    private var drmContentData:DRMContentData;
    public function DRMPreloader():void {
        drmManager.addEventListener(
            DRMAuthenticationCompleteEvent.AUTHENTICATION_COMPLETE,
            onAuthenticationComplete);
        drmManager.addEventListener(DRMAuthenticationErrorEvent.AUTHENTICATION_ERROR,
            onAuthenticationError);
        drmManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
        drmManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
        preloadConnection = new NetConnection();
        preloadConnection.addEventListener(NetStatusEvent.NET_STATUS, onConnect);
        preloadConnection.connect(null);
    }

    private function onConnect( event:NetStatusEvent ):void
    {
        preloadMetadata();
    }
    private function preloadMetadata():void
    {
        preloadStream = new NetStream( preloadConnection );
        preloadStream.client = this;
        var options:NetStreamPlayOptions = new NetStreamPlayOptions();
        options.streamName = videoURL;
        preloadStream.preloadEmbeddedData( options );
    }
    public function onDRMContentData( drmMetadata:DRMContentData ):void
    {
        drmContentData = drmMetadata;
        if ( drmMetadata.authenticationMethod == AuthenticationMethod.USERNAME_AND_PASSWORD )
        {
            authenticateUser();
        }
        else
```

```
            {
                getVoucher();
            }
    }
    private function getVoucher():void
    {
        drmManager.loadVoucher( drmContentData, LoadVoucherSetting.ALLOW_SERVER );
    }

    private function authenticateUser():void
    {
        drmManager.authenticate( drmContentData.serverURL, drmContentData.domain, userName,
password );
    }
    private function onAuthenticationError( event:DRMAuthenticationErrorEvent ):void
    {
        trace( "Authentication error: " + event.errorID + ", " + event.subErrorID );
    }

    private function onAuthenticationComplete( event:DRMAuthenticationCompleteEvent ):void
    {
        trace( "Authenticated to: " + event.serverURL + ", domain: " + event.domain );
        getVoucher();
    }
    private function onDRMStatus( event:DRMStatusEvent ):void
    {
        trace( "DRM Status: " + event.detail);
        trace("--Voucher allows offline playback = " + event.isAvailableOffline );
        trace("--Voucher already cached          = " + event.isLocal );
        trace("--Voucher required authentication = " + !event.isAnonymous );
    }
    private function onDRMError( event:DRMErrorEvent ):void
    {
        trace( "DRM error event: " + event.errorID + ", " + event.subErrorID + ", " + event.text );
    }
    public function onPlayStatus( info:Object ):void
    {
        preloadStream.close();
    }
}
}
```

Eventos y miembros relacionados con DRM de la clase NetStream

Flash Player 10.1 y posterior, Adobe AIR 1.0 y posterior

La clase NetStream proporciona una conexión de corriente unidireccional continua entre Flash Player o una aplicación de AIR y Flash Media Server o el sistema de archivos local. (La clase NetStream también admite la descarga progresiva). Un objeto NetStream es un canal dentro de un objeto NetConnection. La clase NetStream distribuye cuatro eventos relacionados con DRM:

Evento	Descripción
drmAuthenticate (solo AIR)	Definido por la clase DRMAuthenticateEvent. Este evento se distribuye cuando un objeto NetStream intenta reproducir contenido protegido que requiere credenciales de usuario para su autenticación antes de la reproducción. Entre las propiedades de este evento se encuentran header, usernamePrompt, passwordPrompt y urlPrompt, que pueden utilizarse para obtener y configurar las credenciales del usuario. Este evento sucede reiteradas veces hasta que el objeto NetStream recibe unas credenciales de usuario válidas.
drmError	Definido en la clase DRMErrorEvent y distribuido cuando un objeto NetStream intenta reproducir contenido protegido y se produce un error relacionado con DRM. Por ejemplo: si falla la autorización del usuario, se distribuye un objeto de evento de error de DRM. Este error puede producirse porque el usuario no ha adquirido los derechos para ver el contenido. También puede aparecer porque el proveedor de contenido no admite la aplicación de visionado.
drmStatus	Definido en la clase DRMStatusEvent. Este evento se distribuye cuando el contenido protegido comienza a reproducirse (cuando el usuario está autenticado y autorizado para reproducir el contenido). El objeto DRMStatusEvent contiene información relativa a la licencia. La licencia contiene información sobre si el contenido puede estar disponible fuera de línea o cuándo vence la licencia y deja de poder verse el contenido.
status	Definido en events.StatusEvent y solo se distribuye cuando la aplicación intenta reproducir contenido protegido invocando el método NetStream.play(). El valor de la propiedad de código de estado es "DRM.encryptedFLV".

La clase incluye los siguientes métodos específicos de DRM, solo para su uso en AIR:

Método	Descripción
resetDRMVouchers()	Elimina todos los datos de la licencia de administración de derechos digitales (DRM) almacenados en caché local. Para que el usuario pueda acceder al contenido cifrado, la aplicación debe volver a descargar las licencias. Por ejemplo, el siguiente código elimina todas las licencias de la caché: NetStream.resetDRMVouchers();
setDRMAuthenticationCredentials()	Pasa al objeto NetStream una serie de credenciales de autenticación, a saber; nombre de usuario, contraseña y tipo de autenticación, para fines de autenticación. Los tipos de autenticación válidos son `"drm"` y `"proxy"`. Con el tipo de autenticación `"drm"`, la credenciales proporcionadas se autentican con el servidor Adobe Access. Con el tipo de autenticación `"proxy"`, las credenciales se autentican con el servidor proxy y deben coincidir con las que exige el servidor proxy. Por ejemplo, una empresa puede requerir que la aplicación se autentique con un servidor proxy antes de que el usuario pueda acceder a Internet. La opción de proxy permite este tipo de autenticación. A menos que se utilice la autenticación anónima, después de la autenticación proxy el usuario aún necesita autenticarse con Adobe Access para obtener la licencia y poder reproducir el contenido. Se puede utilizar `setDRMAuthenticationCredentials()` una segunda vez, con la opción "`drm`", para realizar la autenticación con Adobe Access.
preloadEmbeddedMetadata()	Analiza un archivo de medios local para metadatos incorporados. Cuando se encuentran metadatos relacionados con DRM, AIR llama a la función callback `onDRMContentData()`.

Asimismo, en AIR, un objeto NetStream llama a las funciones callback `onDRMContentData()` y `onPlayStatus()` como resultado de una llamada al método `preloadEmbeddedMetaData()`. La función `onDRMContentData()` se llama cuando los metadatos DRM se encuentran en un archivo de medios. La función `onPlayStatus()` se llama cuando el archivo se ha analizado. Las funciones `onDRMContentData()` y `onPlayStatus()` se deben definir en el objeto `client` asignado a la instancia de NetStream. Si utiliza el mismo objeto NetStream para precargar licencias y reproducir contenido, espere la llamada a `onPlayStatus()` generada mediante `preloadEmbeddedMetaData()` antes de comenzar la reproducción.

En el siguiente código para AIR se configuran el nombre de usuario ("administrator"), la contraseña ("password") y el tipo de autenticación "drm" para autenticar al usuario. El método setDRMAuthenticationCredentials() debe proporcionar credenciales que coincidan con las guardadas y aceptadas por el proveedor de contenido. Estas credenciales son las mismas que permiten al usuario ver el contenido. No se incluye aquí el código para reproducir el vídeo y verificar que se haya realizado una conexión satisfactoria al flujo de vídeo.

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                            drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Uso de la clase DRMStatusEvent

Flash Player 10.1, Adobe AIR 1.0 y posterior

Un objeto NetStream distribuye un objeto DRMStatusEvent cuando el contenido protegido por Adobe Access comienza a reproducirse correctamente. (Esto sucede cuando la licencia se verifica y el usuario está autenticado y autorizado para ver el contenido). El objeto DRMStatusEvent también se distribuye para usuarios anónimos si se les permite el acceso. Se verifica la licencia para ver si se permite el acceso los usuarios anónimos -que no necesitan autenticación- para que puedan reproducir el contenido. Existen varias razones por las que un usuario anónimo puede no tener acceso. Por ejemplo, un usuario anónimo no tiene acceso al contenido cuando la licencia ha caducado.

El objeto DRMStatusEvent contiene información relacionada con la licencia. Se trata de información sobre si la licencia puede estar disponible fuera de línea o cuándo vence y deja de poder verse el contenido. La aplicación puede aprovechar estos datos para comunicar el estado de la póliza del usuario y los permisos de la misma.

Propiedades de DRMStatusEvent

Flash Player 10.1, Adobe AIR 1.0 y posterior

La clase DRMStatusEvent incluye las siguientes propiedades. Algunas propiedades aparecían en versiones de AIR posteriores a la versión 1.0. Para obtener información detallada sobre las versiones, consulte la *Referencia de ActionScript 3.0*.

En el caso de las propiedades no admitidas en Flash Player 10.1, la clase DRMVoucher proporciona propiedades similares para Flash Player.

Propiedad	Descripción
contentData	Objeto DRMContentData que contiene los metadatos DRM incorporados en el contenido.
detail (solo AIR)	Una cadena que explica el contexto del evento del estado. En DRM 1.0, el único valor válido es DRM.voucherObtained.
isAnonymous (solo AIR)	Indica si el contenido, protegido con Adobe Access, está disponible sin que el usuario tenga que proporcionar credenciales de autenticación (true) o no (false). Un valor "false" significa que el usuario debe proporcionar un nombre de usuario y una contraseña que coincida con la que conoce y espera el proveedor del contenido.
isAvailableOffline (solo AIR)	Indica si el contenido, protegido con Adobe Access, puede estar disponible fuera de línea (true) o no (false). Para que el contenido con protección digital esté disponible fuera de línea, su licencia debe estar en caché en el ordenador del usuario.
isLocal	Indica si la licencia necesaria para reproducir el contenido está almacenada en caché localmente.
offlineLeasePeriod (solo AIR)	La cantidad de días restantes durante los cuales se puede ver el contenido fuera de línea.
policies (solo AIR)	Un objeto personalizado que puede contener propiedades DRM personalizadas.
voucher	Objeto DRMVoucher.
voucherEndDate (solo AIR)	La fecha en que vence indefectiblemente la licencia y el contenido ya no se podrá ver.

Creación de un controlador para el evento DRMStatusEvent

Flash Player 10.1, Adobe AIR 1.0 y posterior

En el siguiente ejemplo se crea un controlador de eventos que produce la información sobre el estado del contenido protegido por DRM para el objeto NetStream que originó el evento. Añada este controlador de eventos a un objeto NetStream que apunte a contenido protegido.

```
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
```

Uso de la clase DRMAuthenticateEvent

Adobe AIR 1.0 y posterior

El objeto DRMAuthenticateEvent se distribuye cuando un objeto NetStream intenta reproducir contenido protegido que requiera credenciales de usuario para autenticación antes de la reproducción.

El controlador del evento DRMAuthenticateEvent es responsable de obtener las credenciales exigidas (nombre de usuario, contraseña y tipo) y de pasar los valores al método `NetStream.setDRMAuthenticationCredentials()` para su validación. Cada aplicación de AIR debe proveer un mecanismo para obtener las credenciales de usuario. Por ejemplo, la aplicación proporcionaría al usuario una interfaz sencilla para introducir los valores de nombre de usuario y contraseña. Asimismo, ofrece un mecanismo para gestionar y limitar los intentos repetidos de autenticación.

Propiedades de DRMAuthenticateEvent

Adobe AIR 1.0 y posterior

La clase DRMAuthenticateEvent incluye las siguientes propiedades:

Propiedad	Descripción
authenticationType	Indica si las credenciales facilitadas son para autenticación con Adobe Access (“drm”) o un servidor proxy (“proxy”). Por ejemplo: la opción "proxy" permite a la aplicación realizar la autenticación ante un servidor proxy si se necesita antes de que el usuario pueda acceder a Internet. A menos que se utilice la autenticación anónima, después de la autenticación proxy el usuario aún necesita autenticarse con Adobe Access para obtener la licencia y poder reproducir el contenido. Se puede utilizar setDRMAuthenticationcredentials() una segunda vez, con la opción “drm”, para realizar la autenticación con Adobe Access.
header	El encabezado del archivo de contenido cifrado suministrado por el servidor. Contiene información acerca del contexto del contenido cifrado. Esta cadena de encabezado se puede pasar a la aplicación Flash para permitir que construya un cuadro de diálogo de introducción de nombre de usuario y contraseña. La cadena de encabezado se puede aprovechar como instrucciones del cuadro de diálogo. Por ejemplo, el encabezado puede ser del tipo “Escriba su nombre de usuario y contraseña”.
netstream	El objeto NetStream que inició este evento.
passwordPrompt	Una solicitud de la credencial de contraseña, proporcionada por el servidor. La cadena puede incluir una instrucción sobre el tipo de contraseña que se requiere.
urlPrompt	Una solicitud de una cadena de URL, proporcionada por el servidor. La cadena puede indicar el lugar donde se envían el nombre de usuario y la contraseña.
usernamePrompt	Una solicitud de la credencial de nombre de usuario, proporcionada por el servidor. La cadena puede incluir una instrucción sobre el tipo de nombre de usuario que se requiere. Por ejemplo: un proveedor de contenido puede exigir que el nombre de usuario sea una dirección de correo electrónico.

Las cadenas mencionadas anteriormente las suministra solo el servidor FMRMS. El servidor Adobe Access no utiliza estas cadenas.

Creación de un controlador para el evento DRMAuthenticateEvent

Adobe AIR 1.0 y posterior

En el siguiente ejemplo se crea un controlador de eventos que pasa una serie de credenciales de autenticación programadas al objeto NetStream que originó el evento. (No se incluye aquí el código para reproducir el vídeo y verificar que se haya realizado una conexión satisfactoria al flujo de vídeo).

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Creación de una interfaz para recuperar las credenciales de usuario

Adobe AIR 1.0 y posterior

Si el contenido protegido requiere autenticación del usuario, la aplicación de AIR normalmente tiene que recuperar las credenciales de autenticación del usuario a través de una interfaz de usuario.

Lo siguiente es un ejemplo en Flex de una interfaz de usuario sencilla para recuperar credenciales de usuario. Consta de un objeto de panel que contiene dos objetos TextInput, uno para cada una de las credenciales de nombre de usuario y contraseña. El panel también contiene un botón que inicia el método `credentials()`.

```
<mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute" title="Login">
    <mx:TextInput x="110" y="46" id="uName"/>
    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
    <mx:Text x="35" y="48" text="Username:"/>
    <mx:Text x="35" y="78" text="Password:"/>
    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
</mx:Panel>
```

El método `credentials()` es un método definido por el usuario que pasa al método `setDRMAuthenticationCredentials()` los valores de nombre de usuario y contraseña. Una vez pasados estos valores, el método `credentials()` restaura los valores de los objetos TextInput.

```
<mx:Script>
    <![CDATA[
        public function credentials():void
        {
            videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
            uName.text = "";
            pWord.text = "";
        }
    ]]>
</mx:Script>
```

Una forma de implementar este tipo de interfaz sencilla es incluir el panel como parte de un nuevo estado. El nuevo estado se origina del estado base cuando se emite el objeto DRMAuthenticateEvent. El siguiente ejemplo contiene un objeto VideoDisplay con un atributo de origen que apunta a un archivo FLV protegido. En ese caso, el método `credentials()` se modifica para que también devuelva la aplicación a su estado base. Este método lo hace tras pasar las credenciales del usuario y restaurar los valores del objeto TextInput.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="800"
    height="500"
    title="DRM FLV Player"
    creationComplete="initApp()" >

    <mx:states>
        <mx:State name="LOGIN">
            <mx:AddChild position="lastChild">
                    <mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute"
                            title="Login">
                    <mx:TextInput x="110" y="46" id="uName"/>
                    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
                    <mx:Text x="35" y="48" text="Username:"/>
                    <mx:Text x="35" y="78" text="Password:"/>
                    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
                    <mx:Button x="193" y="115" label="Reset" click="uName.text='';
                            pWord.text='';"/>
                </mx:Panel>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Script>
        <![CDATA[
                import flash.events.DRMAuthenticateEvent;
            private function initApp():void
            {
                videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                                            drmAuthenticateEventHandler);
            }

            public function credentials():void
            {
                videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
                uName.text = "";
                pWord.text = "";
                currentState='';
            }

            private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
            {
                currentState='LOGIN';
            }
        ]]>
    </mx:Script>

    <mx:VideoDisplay id="video" x="50" y="25" width="700" height="350"
        autoPlay="true"
        bufferTime="10.0"
        source="http://www.example.com/flv/Video.flv" />
</mx:WindowedApplication>
```

Uso de la clase DRMErrorEvent

Flash Player 10.1 y posterior, Adobe AIR 1.0 y posterior

Adobe Flash Player y Adobe AIR distribuyen un objeto DRMErrorEvent cuando un objeto NetStream que intenta reproducir contenido protegido se encuentra con un error relacionado con DRM. Si las credenciales del usuario no son válidas en una aplicación de AIR, el objeto DRMAuthenticateEvent se distribuye una y otra vez hasta que el usuario introduce las credenciales correctas o la aplicación no acepta más intentos. La aplicación es responsable de detectar si hay otros eventos de error de DRM para detectar, identificar y controlar los errores relacionados con DRM.

Incluso con credenciales de usuario válidas, las condiciones de la licencia del contenido pueden impedir que pueda ver contenido cifrado. Por ejemplo, a un usuario se le puede denegar el acceso al intentar ver contenido en una aplicación no autorizada. Una aplicación no autorizada es aquella no validada por el editor del contenido cifrado. En este caso se distribuye un objeto DRMErrorEvent.

Los eventos de error también pueden activarse si el contenido está dañado o si la versión de la aplicación no coincide con lo que se especifica en la licencia. La aplicación debe proveer un mecanismo adecuado para controlar los errores.

Propiedades de DRMErrorEvent

Flash Player 10.1 y posterior, Adobe AIR 1.0 y posterior

Para obtener una lista completa de errores, consulte Errores en tiempo de ejecución en la Referencia de ActionScript 3.0. Los errores relacionados con DRM comienzan en el error 3300.

Creación de un controlador para el evento DRMErrorEvent

Flash Player 10.1 y posterior, Adobe AIR 1.0 y posterior

El ejemplo siguiente crea un controlador de eventos para el objeto NetStream que dio origen al evento. Se llama cuando el objeto NetStream encuentra un error al intentar reproducir el contenido protegido. Normalmente, cuando una aplicación encuentra un error, lleva a cabo una serie de tareas de limpieza. Seguidamente, informa al usuario del error y ofrece opciones para resolver el problema.

```
private function drmErrorEventHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
```

Uso de la clase DRMManager

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

Utilice la clase DRMManager para administrar licencias y sesiones del servidor de derechos de medios en aplicaciones.

Administración de licencias (solo AIR)

Cada vez que un usuario reproduce contenido protegido, el motor de ejecución obtiene la licencia necesaria para ver el contenido y la guarda en caché. Si la aplicación guarda el archivo localmente, y la licencia permite la reproducción sin conexión, el usuario puede ver el contenido en la aplicación de AIR. Esta reproducción local sin conexión se produce incluso si no hay disponible ninguna conexión con el servidor de derechos digitales. Con el método DRMManager y NetStream `preloadEmbeddedMetadata()`, puede guardar en caché previamente la licencia. La aplicación no deberá obtener la licencia necesaria para ver el contenido. Por ejemplo, la aplicación puede descargar el archivo de medios y posteriormente obtener la licencia mientras que el usuario esté en línea.

Para precargar una licencia, utilice el método `preloadEmbeddedMetadata()` de NetStream para obtener un objeto DRMContentData. El objeto DRMContentData contiene la dirección URL y el dominio del servidor de derechos de medios que puede proporcionar la licencia y describe si se requiere la autenticación del usuario. Con esta información, puede llamar al método `loadVoucher()` de DRMManager para obtener y almacenar en caché la licencia. El flujo de trabajo para precarga de licencias se describe más detalladamente en "Precarga de licencias para reproducción sin conexión" en la página 555.

Administración de sesión

También puede utilizar DRMManager para autenticar al usuario en un servidor de derechos de medios y para administrar sesiones persistentes.

Llame al método `authenticate()` de DRMManager para establecer una sesión con el servidor de derechos de medios. Una vez que la autenticación se lleva a cabo correctamente, DRMManager distribuye un objeto DRMAuthenticationCompleteEvent. Este objeto contiene un token de sesión. Este token se puede guardar para establecer futuras sesiones de modo que el usuario no tenga que proporcionar sus credenciales de cuenta. Transmita el símbolo al método `setAuthenticationToken()` para establecer una nueva sesión autenticada. (La configuración del servidor que generó el token determina su caducidad y otros atributos. El código de la aplicación de AIR no debe interpretar la estructura de datos del token, ya que esta estructura pueda cambiar con futuras actualizaciones de AIR.)

Los token de autenticación se pueden transferir a otros equipos. Para proteger los token, puede almacenarlos en el almacén local cifrado de AIR. Consulte "Almacenamiento local cifrado" en la página 740 para obtener más información.

Eventos DRMStatus

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

DRMManager distribuye un objeto DRMStatusEvent cuando la llamada al método `loadVoucher()` se completa correctamente.

Si se obtiene una licencia, la propiedad `detail` (solo AIR) del objeto de evento tendrá el valor: "DRM.voucherObtained" y la propiedad `voucher` contendrá el objeto DRMVoucher.

Si no se obtiene ninguna licencia, la propiedad `detail` (solo AIR) aún tendrá el valor: "DRM.voucherObtained"; sin embargo, la propiedad `voucher` será `null`. No se puede obtener una licencia si, por ejemplo, se utiliza LoadVoucherSetting de *localOnly* y no existen ninguna licencia almacenada en la caché local.

Si la llamada a `loadVoucher()` no se realiza correctamente, quizá debido a un error de comunicación o autenticación, DRMManager distribuye un objeto DRMErrorEvent o un objeto DRMAuthenticationErrorEvent.

Eventos DRMAuthenticationComplete

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

DRMManager distribuye un objeto DRMAuthenticationCompleteEvent cuando un usuario se autentica correctamente mediante una llamada al método `authenticate()`.

El objeto DRMAuthenticationCompleteEvent contiene un token reutilizable que se puede usar para mantener la autenticación del usuario a través de sesiones de la aplicación. Transmita este token al método `setAuthenticationToken()` de DRMManager para restablecer la sesión. (El creador del token define sus atributos, como la caducidad. Adobe no proporciona ninguna API para examinar los atributos del token.)

Eventos DRMAuthenticationError

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

DRMManager distribuye un objeto DRMAuthenticationErrorEvent cuando un usuario no se puede autenticar correctamente mediante una llamada a los métodos `authenticate()` o `setAuthenticationToken()`.

Uso de la clase DRMContentData

Flash Player 10.1 y posterior, Adobe AIR 1.5 y posterior

El objeto DRMContentData contiene las propiedades de metadatos de contenido protegido con Adobe Access. Las propiedades DRMContentData contienen la información necesaria para obtener una licencia para ver el contenido. Puede utilizar la clase DRMContentData para obtener el archivo de metadatos asociado al contenido, tal como se describe en "Flujo de trabajo detallado de la API" en la página 553.

Para obtener más información, consulte la clase DRMContentData en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Actualización de Flash Player para admitir Adobe Access

Flash Player 10.1 y posterior

Importante: Flash Player 11.5 y versiones posteriores integra el módulo Adobe Access, por lo que el paso de actualización (llamar a `SystemUpdater.update(SystemUpdaterType.DRM)`) no es necesario. Esto incluye los siguientes navegadores y plataformas:

- Control ActiveX para Flash Player 11.5, para todas las plataforma excepto Internet Explorer en Windows 8
- Plug-in de Flash Player 11.5, para todos los navegadores
- Adobe AIR (escritorio y móvil)

Esto significa que el paso de actualización *sigue siendo necesario* en los casos siguientes:

- Internet Explorer en Windows 8
- Flash Player 11.4 y versiones anteriores, excepto en Google Chrome 22 y versiones posteriores (todas las plataformas) o 21 y versiones posteriores (Windows)

Nota: *sigue siendo seguro llamar a* `SystemUpdater.update(SystemUpdaterType.DRM)` *en un sistema con Flash Player 11.5 o versión posterior, pero no se descargará nada.*

Para admitir Adobe Access, Flash Player requiere el módulo Adobe Access. Cuando Flash intenta reproducir contenido protegido, el motor de ejecución indica si se debe descargar el módulo o una nueva versión de Flash Player. De este modo, Flash Player da a los desarrolladores de SWF la opción de no actualizar si no lo desean.

En la mayoría de los casos, para reproducir contenido protegido los desarrolladores de SWF actualizan al módulo Adobe Access o al reproductor requeridos. Para actualizar, puede utilizar la API SystemUpdater para obtener la versión más reciente del módulo Adobe Access o de Flash Player.

La API SystemUpdater solo permite una actualización cada vez. El código de error 2202 indica que se está llevando a cabo una actualización en la instancia actual del motor de ejecución o en otra distinta. Por ejemplo, si se lleva a cabo una actualización en una instancia de Flash Player en Internet Explorer, no es posible hacerlo en una instancia de Flash Player que se ejecute en Firefox.

La API SystemUpdater se admite únicamente en plataformas de escritorio.

Nota: *en versiones anteriores a Flash Player 10.1, utilice el mecanismo de actualización admitido en versiones anteriores del reproductor (descarga manual e instalación desde www.adobe.com o ExpressInstall). Asimismo, el archivo de instalación de AIR gestiona las actualizaciones necesarias de Adobe Access y no admite la API SystemUpdater.*

Detección de un evento update

Flash Player 10.1 y posterior

Cuando se requiere una actualización del módulo Adobe Access, el objeto NetStream distribuye un evento NetStatusEvent con un valor de código de `DRM.UpdateNeeded`. Este valor indica que el objeto NetStream no puede reproducir el flujo protegido con ninguno de los módulos Adobe Access instalados. Detecte este evento y llame al siguiente código:

```
SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

Este código actualiza el módulo Adobe Access instalado en el reproductor. No es necesario el consentimiento del usuario para actualizar el módulo.

Si no se encuentra el módulo Adobe Access, se emite un error. Consulte el paso 3 de "Flujo de trabajo detallado de la API" en la página 553.

Nota: *si se llama a play() en un flujo cifrado en reproductores anteriores a la versión 10.1, se emite un evento NetStatusEvent con valor de código de NetStream.Play.StreamNotFound. En reproductores anteriores, utilice el mecanismo de actualización admitido en dichos reproductores (descarga manual e instalación desde www.adobe.com o ExpressInstall).*

Cuando se necesita una actualización del propio reproductor, el objeto SystemUpdater distribuye un evento StatusEvent con un valor de código de `DRM.UpdateNeededButIncompatible`. Para actualizar el reproductor sí es necesario el consentimiento del usuario. En su aplicación, incluya una interfaz para que el usuario dé su consentimiento y para iniciar la actualización del reproductor. Detecte el evento StatusEvent y llame al siguiente código:

```
SystemUpdater.update(flash.system.SystemUpdaterType.SYSTEM);
```

Este código inicia la actualización del reproductor.

Encontrará documentación sobre más eventos de la clase SystemUpdater en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Una vez finalizada la actualización del reproductor, se redirige al usuario a la página en la que se inició la actualización. Se descarga el módulo Adobe Access y el flujo comienza a reproducirse.

Licencias fuera de banda

Flash Player 11 y posterior, Adobe AIR 3.0 y posterior

Las licencias también se pueden obtener fuera de banda (sin conectarse al servidor de licencias Adobe Access). Se consigue almacenando la licencia en el disco y en la memoria mediante el método `storeVoucher`.

Para reproducir vídeos cifrados en Flash Player y AIR, es preciso que los respectivos motores de ejecución obtengan la licencia DRM de dichos vídeos. La licencia DRM contiene la clave de cifrado del vídeo y se genera mediante el servidor de licencias Adobe Access que el cliente haya implantado.

Normalmente, el motor de ejecución de Flash Player/AIR obtiene esta solicitud de licencia del servidor de licencias Adobe Access indicado en los metadatos DRM del vídeo (clase `DRMContentData`). La aplicación de Flash/AIR puede activar la solicitud de la licencia llamando al método `DRMManager.loadVoucher()`. O bien el motor de ejecución de Flash Player/AIR solicitará automáticamente una licencia al inicio de la reproducción del vídeo cifrado si no hay ninguna licencia para el contenido en el disco ni en la memoria. En cualquier caso, el rendimiento de la aplicación de Flash/AIR se ve afectado por la comunicación con el servidor de licencias Adobe Access.

`DRMManager.storeVoucher()` permite que la aplicación de Flash/AIR pueda enviar licencias DRM al motor de ejecución de Flash Player/AIR que ha obtenido fuera de banda. El motor de ejecución puede omitir el proceso de solicitud de licencias y utilizar las licencias reenviadas para reproducir vídeos cifrados. Sigue siendo preciso que la licencia DRM sea generada por el servidor de licencias Adobe Access para poder obtenerla fuera de banda. No obstante, existe la opción de alojar las licencias en cualquier servidor HTTP en vez de publicarlas directamente en el servidor de licencias Adobe Access.

`DRMManager.storeVoucher()` también se usa para admitir que varios dispositivos puedan compartir licencias DRM. En Adobe Access 3.0, esta función se denomina “Soporte de dominios”. Si su implantación admite este caso, puede registrar varios equipos en un grupo de dispositivos con el método `DRMManager.addToDeviceGroup()`. Si hay un equipo con una licencia de dominio válida para un contenido determinado, la aplicación de AIR puede extraer las licencias DRM serializadas con el método `DRMVoucher.toByteArray()` y en el resto de los equipos puede importar las licencias con el método `DRMManager.storeVoucher()`.

Registro de dispositivos

Las licencias DRM están vinculadas en el equipo del usuario. Por ello, las aplicaciones de Flash/AIR necesitarán un ID exclusivo para que el equipo del usuario haga referencia al objeto de la licencia DRM serializada. En el siguiente contexto se describe el proceso de registro de un dispositivo:

Asumiremos que ha llevado a cabo las siguientes tareas:

- Ha configurado el SDK del servidor Adobe Access.
- Ha definido un servidor HTTP para obtener licencias pregeneradas.
- Ha creado una aplicación de Flash para visualizar el contenido protegido.

La fase de registro del dispositivo incluye los pasos siguientes:

1. La aplicación de Flash crea un ID generado de forma aleatoria.

2 La aplicación de Flash invoca el método `DRMManager.authenticate()`. La aplicación debe incluir el ID generado de forma aleatoria en la solicitud de autenticación. Por ejemplo, debe incluir el ID en el campo del nombre de usuario.

3 La acción mencionada en el paso 2 hará que Adobe Access envíe una solicitud de autenticación al servidor del cliente. Esta solicitud incluye el certificado del dispositivo.

   a El servidor extrae de la solicitud el certificado del dispositivo y el ID generado y lo guarda.

   b El subsistema del cliente pregenera licencias para este certificado del dispositivo, las guarda y garantiza el acceso de modo que las asocia con el ID generado.

4 El servidor responde a la solicitud con un mensaje “de éxito”.

5 La aplicación de Flash guarda el ID generado localmente en un objeto local compartido (Local Shared Object, LSO por sus siglas en inglés).

Tras el registro del dispositivo, la aplicación de Flash utiliza el ID generado del mismo modo que habría usado el ID de dispositivo en el esquema anterior:

1 La aplicación de Flash intentará localizar el ID generado en el LSO.

2 Si encuentra el ID generado, la aplicación de Flash lo utilizará durante la descarga de las licencias pregeneradas. La aplicación de Flash enviará las licencias al cliente de Adobe Access para su consumo con el método `DRMManager.storeVoucher()`.

3 Si no encuentra el ID generado, la aplicación de Flash deberá llevar a cabo los pasos del procedimiento de registro del dispositivo.

Restauración de valores de fábrica

Si el usuario del dispositivo invoca la opción de restauración de valores de fábrica, deberá purgarse el certificado del dispositivo. Para poder continuar reproduciendo contenido protegido, la aplicación de Flash deberá llevar a cabo de nuevo los pasos del procedimiento de registro del dispositivo. Si la aplicación de Flash envía una licencia pregenerada caducada, el cliente de Adobe Access la rechazará, ya que la licencia estará cifrada para un ID de dispositivo antiguo.

Soporte de dominios

Flash Player 11 y posterior, Adobe AIR 3.0 y posterior

Si los metadatos del contenido especifican que es necesario registrar el dominio, la aplicación de AIR puede invocar una API para unirse a un grupo de dispositivos. Esta acción desencadena una solicitud de registro de dominio que se envía al servidor del dominio. Una vez emitida la licencia en un grupo de dispositivos, esta se puede exportar y compartir con otros dispositivos que se hayan unido al grupo de dispositivos.

Seguidamente, la información del grupo de dispositivos se utiliza en el objeto `VoucherAccessInfo` de `DRMContentData`, que después se utiliza para presentar la información necesaria para recuperar y consumir adecuadamente una licencia.

Reproducción de contenido cifrado con soporte de dominios

Para reproducir contenido con Adobe Access, siga estos pasos:

1. Con `VoucherAccessInfo.deviceGroup`, compruebe si es necesario llevar a cabo el registro del grupo de dispositivos.
2. Si es necesario autenticarse:
   - **a** Utilice la propiedad `DeviceGroupInfo.authenticationMethod` para saber si es necesario autenticarse.
   - **b** Si es necesario autenticarse, autentique al usuario mediante UNA de las siguientes acciones:
     - Obtenga el nombre de usuario y la contraseña del usuario. Invoque `DRMManager.authenticate(deviceGroup.serverURL, deviceGroup.domain, username, password)`.
     - Obtenga un símbolo de autenticación en caché/pregenerado e invoque `DRMManager.setAuthenticationToken()`.
   - **c** Invoque `DRMManager.addToDeviceGroup()`.
3. Para obtener la licencia del contenido, lleve a cabo una de las tareas siguientes:
   - **a** Utilice el método `DRMManager.loadVoucher()`.
   - **b** Obtenga la licencia de un dispositivo distinto registrado en el mismo grupo de dispositivos. Proporcione la licencia a `DRMManager` a través del método `DRMManager.storeVoucher()`.
4. Reproduzca el contenido cifrado con el método `NetStream.play()`.

Para exportar la licencia del contenido, cualquiera de los dispositivos puede proporcionar los bytes sin procesar de la licencia con el método `DRMVoucher.toByteArray()` tras haber obtenido la licencia del servidor de licencias Adobe Access. Los proveedores de contenido suelen limitar el número de dispositivos de cada grupo de dispositivos. Si se alcanza el límite, es posible llamar al método `DRMManager.removeFromDeviceGroup()` de un dispositivo no utilizado antes de registrar el dispositivo actual.

Previsualización de licencias

La aplicación de Flash puede enviar una solicitud de previsualización de licencia, esto es, la aplicación puede llevar a cabo una operación de previsualización antes de preguntar al usuario si quiere comprar el contenido y determinar al mismo tiempo si el equipo del usuario cumple todos los requisitos necesarios para la reproducción. La previsualización de licencias hace referencia a la posibilidad del cliente para previsualizar una licencia (ver qué derechos otorga), en contraposición a la previsualización del contenido (ver un pequeño fragmento del contenido antes de decidir si se realiza la compra). Algunos de los parámetros exclusivos de cada equipo son: salidas disponibles y estado de protección, versión de motor de ejecución/DRM disponible y nivel de seguridad del cliente DRM. El modo de previsualización de licencias permite al cliente del motor de ejecución/DRM probar la lógica de negocio del servidor de licencias y proporcionar información al usuario para que pueda tomar una decisión informada. Así, el cliente puede ver qué aspecto tiene una licencia válida pero no recibe la clave real para descifrar el contenido. El soporte para previsualización de licencias es opcional y solo es necesario si se implementa una aplicación personalizada que utilice esta funcionalidad.

Entrega de contenido

Adobe Access ignora el mecanismo de entrega del contenido, ya que Flash Player se abstrae de la capa de red y simplemente proporciona contenido protegido al subsistema de Adobe Access. De este modo, el contenido se puede entregar por HTTP, flujo dinámico HTTP, RTMP o RTMPE.

Sin embargo, pueden surgir problema debidos a la necesidad de metadatos del contenido protegido (normalmente, como un archivo '.metadata') antes de que Adobe Access pueda descifrar el contenido. En concreto, con el protocolo RTMP/RTMPE, solo se pueden entregar datos FLV y F4V al cliente a través de Flash Media Server (FMS). Por este motivo, el cliente debe recuperar los metadatos por otros medios. Una opción para solucionar este problema es alojar los metadatos en un servidor web HTTP e implementar el reproductor de vídeo cliente para que recupere los metadatos correspondientes en función del contenido reproducido.

```
private function getMetadata():void{

    extrapolated-path-to-metadata = "http://metadatas.mywebserver.com/" + videoname;
    var urlRequest : URLRequest = new URLRequest(extrapolated-path-to-the-metadata + ".metadata");
    var urlStream : URLStream = new URLStream();
    urlStream.addEventListener(Event.COMPLETE, handleMetadata);
    urlStream.addEventListener(IOErrorEvent.NETWORK_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.IO_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.VERIFY_ERROR, handleIOError);
    try{
        urlStream.load(urlRequest);
     }catch(se:SecurityError){
        videoLog.text += se.toString() + "\n";
     }catch(e:Error){
        videoLog.text += e.toString() + "\n";
     }
}
```

Open Source Media Framework

Open Source Media Framework (OSMF por sus siglas en inglés) es un marco basado en ActionScript que ofrece total flexibilidad y control para crear intensas experiencias multimedia. Para obtener más información sobre OSMF, visite el sitio de desarrolladores de OSMF.

Flujo de trabajo para reproducir contenido protegido

1 Cree una instancia de `MediaPlayer`.

```
player = new MediaPlayer();
```

2 Registre el evento `MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE` en el reproductor. Este evento se distribuirá si el contenido DRM está protegido.

```
player.addEventListener(MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE,
onDRMCapabilityChange);
```

3 En el controlador de eventos, obtenga la instancia de `DRMTrait`. `DRMTrait` es la interfaz a través de la que se invocan métodos relacionados con DRM, por ejemplo, `authenticate()`. Cuando se carga contenido protegido con DRM, OSMF lleva a cabo acciones de validación de DRM y distribuye eventos de estado. Añada un controlador de eventos `DRMEvent.DRM_STATE_CHANGE` a `DRMTrait`.

```
private function onDRMCapabilityChange
    (event :MediaPlayerCapabilityChangeEvent) :void
        {
            if (event.type == MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE
                && event.enabled)
            {
                drmTrait = player.media.getTrait(MediaTraitType.DRM) as DRMTrait;
                drmTrait.addEventListener
                    (DRMEvent.DRM_STATE_CHANGE, onDRMStateChange);
            }
        }
```

4 Gestione los eventos DRM en el método `onDRMStateChange()`.

```
private function onDRMStateChange(event :DRMEvent) :void
{
    trace ( "DRMState: ",event.drmState);
    switch(event.drmState)
    {
        case DRMState.AUTHENTICATION_NEEDED:
            // Identity-based content
            var authPopup :AuthWindow = AuthWindow.create(_parentWin);
            authPopup.serverURL = event.serverURL;
            authPopup.addEventListener("dismiss", function () :void {
                trace ("Authentication dismissed");
                if(_drmTrait != null)
                {
                    //Ignore authentication. Just
                    //try to acquire a license.
                    _drmTrait.authenticate(null, null);
                }
            });
            authPopup.addEventListener("authenticate",
                            function (event :AuthWindowEvent) :void {
                if(_drmTrait != null)
                {
                    _drmTrait.authenticate(event.username, event.password);
                }
            });
            authPopup.show();
            break;
        case DRMState.AUTHENTICATING:
            //Display any authentication message.
            trace("Authenticating...");
            break;
        case DRMState.AUTHENTICATION_COMPLETE:
            // Start to retrieve voucher and playback.
            // You can display the voucher information at this point.
            if(event.token)
            // You just received the authentication token.
            {
                trace("Authentication success. Token: \n", event.token);
            }
            else
            // You have got the voucher.
            {
                trace("DRM License:");
```

```
                    trace("Playback window period: ",
                        !isNaN(event.period) ? event.period == 0 ?
                        "<unlimited>" : event.period : "<none>");
                    trace("Playback window end date: ",
                        event.endDate != null ? event.endDate : "<none>");
                    trace("Playback window start date: ",
                        event.startDate != null ? event.startDate : "<none>");
                }
                break;
            case DRMState.AUTHENTICATION_ERROR:
                trace ("DRM Error:", event.mediaError.errorID +
                    "[" + DRMErrorEventRef.getDRMErrorMnemonic
                    (event.mediaError.errorID) + "]");
                //Stop everything.
                player.media = null;
                break;
            case DRMState.DRM_SYSTEM_UPDATING:
                Logger.log("Downloading DRM module...");
                break;
            case DRMState.UNINITIALIZED:
                break;
        }
    }
```

Capítulo 28: Cómo añadir contenido PDF en AIR

Adobe AIR 1.0 y posterior

Las aplicaciones que se ejecutan en Adobe® AIR® pueden representar no solo contenido SWF y HTML, sino además, contenido PDF. Las aplicaciones de AIR representan el contenido PDF utilizando la clase HTMLLoader, el motor WebKit y el plug-in Adobe® Reader®. En una aplicación de AIR, el contenido PDF puede ocupar toda la altura y anchura de la aplicación, o solo una parte de la interfaz. El plugin del navegador de Adobe Reader controla la visualización de archivos PDF en una aplicación de AIR. Las modificaciones de la interfaz de barra de herramientas de Reader (como las de posición, anclaje y visibilidad) se mantienen al visualizar posteriormente archivos PDF tanto en las aplicaciones de AIR como en el navegador.

Importante: *para poder representar contenido PDF en AIR, el usuario debe tener instalado Adobe Reader o Adobe® Acrobat® versión 8.1 o posterior.*

Detección de la capacidad de PDF

Adobe AIR 1.0 y posterior

Si el usuario no tiene instalada la versión 8.1 o posterior de Adobe Reader o Adobe Acrobat, el contenido PDF no se visualiza en las aplicaciones de AIR. Para detectar si un usuario puede representar contenido PDF, compruebe primero la propiedad `HTMLLoader.pdfCapability`. Esta propiedad se define en una de las siguientes constantes de la clase HTMLPDFCapability:

Constante	Descripción
HTMLPDFCapability.STATUS_OK	Se detecta una versión suficiente (8.1 o posterior) de Adobe Reader y se puede cargar contenido PDF en un objeto HTMLLoader.
HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND	No se detecta ninguna versión de Adobe Reader. Un objeto HTMLLoader no puede mostrar contenido PDF.
HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD	Se detecta una versión de Adobe Reader pero es demasiado antigua. Un objeto HTMLLoader no puede mostrar contenido PDF.
HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD	Se detecta una versión suficiente (8.1 o posterior) de Adobe Reader, pero la versión de Adobe Reader que está configurada para gestionar el contenido PDF es anterior a Reader 8.1. Un objeto HTMLControl no puede visualizar contenido PDF. Un objeto HTMLLoader no puede mostrar contenido PDF.

en Windows, si está funcionando Adobe Acrobat o Adobe Reader versión 7.x o posterior en el sistema del usuario, esa es la versión que se utiliza aunque se instale una versión posterior compatible con la carga de PDF cargado. En este caso, si el valor de la propiedad `pdfCampability` es `HTMLPDFCapability.STATUS_OK`, cuando una aplicación de AIR intenta cargar contenido PDF, la versión anterior de Acrobat o Reader presenta un aviso (y no se emite ninguna excepción en la aplicación de AIR). Si esta es una situación posible para los usuarios, considere facilitarles instrucciones para que cierren Acrobat mientras ejecutan la aplicación de AIR. Puede convenir que aparezcan estas instrucciones si el contenido PDF no se carga en un plazo de tiempo aceptable.

En Linux, AIR busca Adobe Reader en la RUTA exportada por el usuario (si contiene el comando acroread) y en el directorio /opt/Adobe/Reader.

El siguiente código detecta si un usuario puede visualizar contenido PDF en una aplicación de AIR. Si el usuario no puede visualizar PDF, se rastrea el código de error que corresponde al objeto de error HTMLPDFCapability:

```
 if(HTMLLoader.pdfCapability == HTMLPDFCapability.STATUS_OK)
{
    trace("PDF content can be displayed");
}
else
{
    trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

Carga de contenido PDF

Adobe AIR 1.0 y posterior

Se puede añadir contenido PDF a una aplicación de AIR creando una instancia de HTMLLoader, configurando sus dimensiones y cargando la ruta de un archivo PDF.

El siguiente ejemplo carga un archivo PDF desde un sitio externo. Sustituya el valor de URLRequest por la ruta a un archivo PDF externo disponible.

```
 var request:URLRequest = new URLRequest("http://www.example.com/test.pdf");
pdf = new HTMLLoader();
pdf.height = 800;
pdf.width = 600;
pdf.load(request);
container.addChild(pdf);
```

También se puede cargar contenido desde las URL de archivos y esquemas de URL específicos para AIR, como app y app-storage. En el siguiente ejemplo el código carga el archivo test.pdf en el subdirectorio de PDF del directorio de la aplicación:

app:/js_api_reference.pdf

Para obtener más información sobre esquemas de URL de AIR, consulte “Esquemas de URI” en la página 846.

Uso de scripts con el contenido PDF

Adobe AIR 1.0 y posterior

Se puede utilizar JavaScript para controlar el contenido PDF del mismo modo que en una página web visualizada en el navegador.

Las extensiones de JavaScript para Acrobat ofrecen las siguientes funciones, entre otras:

- Control de la ampliación y del desplazamiento por la página
- Procesamiento de formularios dentro del documento
- Control de eventos multimedia

Para obtener todos los detalles de la extensiones de JavaScript para Adobe Acrobat, consulte el Centro de desarrollo de Adobe Acrobat en http://www.adobe.com/devnet/acrobat/javascript.html (en inglés).

Aspectos básicos de la comunicación HTML-PDF

Adobe AIR 1.0 y posterior

JavaScript en una página HTML puede enviar un mensaje a JavaScript en contenido PDF llamando al método `postMessage()` del objeto DOM que representa el contenido PDF. Por ejemplo, tomemos el siguiente contenido PDF incorporado:

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

El siguiente código JavaScript del contenido HTML envía un mensaje a JavaScript del archivo PDF:

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

El archivo PDF puede incluir JavaScript para recibir este mensaje. Se puede añadir código JavaScript a archivos PDF en algunos contextos, entre ellos los contextos a nivel de documento, carpeta, página, campo y lote. Aquí solo trataremos el contexto a nivel de documento, que define los scripts que se evalúan al abrirse el documento PDF.

Un archivo PDF puede añadir una propiedad `messageHandler` al objeto `hostContainer`. La propiedad `messageHandler` es un objeto que define las funciones del controlador para responder a los mensajes. En el siguiente ejemplo, el código define la función para gestionar mensajes que recibe del contenedor host (que es el contenido HTML que incorpora el archivo PDF) el archivo PDF:

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

El código JavaScript de una página HTML puede llamar al método `postMessage()` del objeto PDF contenido en la página. Al llamar a este método se envía un mensaje (`"Hello from HTML"`) a JavaScript a nivel de documento en el archivo PDF:

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

Para ver un ejemplo más avanzado, así como información sobre el uso de Acrobat 8 para añadir JavaScript a un archivo PDF, consulte Creación de scripts entre contenidos PDF en Adobe AIR.

Uso de scripts con el contenido PDF en ActionScript

Adobe AIR 1.0 y posterior

El código ActionScript (en contenido SWF) no puede comunicarse directamente con JavaScript en contenido PDF. Sin embargo, ActionScript puede comunicarse con el código JavaScript en la página HTML cargada en un objeto HTMLLoader que carga contenido PDF, y ese código JavaScript puede comunicarse con el JavaScript que está cargado en el archivo PDF. Para obtener más información, consulte “Programación con HTML y JavaScript en AIR” en la página 1019.

Limitaciones conocidas del contenido PDF en AIR

Adobe AIR 1.0 y posterior

El contenido PDF en Adobe AIR tiene las limitaciones siguientes:

- El contenido PDF no se visualiza en una ventana (un objeto NativeWindow) que es transparente (para la que la propiedad `transparent` está definida en `true`).
- El orden de visualización de un archivo PDF no funciona de la misma manera que para otros objetos de visualización en una aplicación de AIR. Si bien el contenido PDF se ajusta correctamente de acuerdo con el orden de visualización de HTML, siempre permanece encima del contenido en el orden de visualización de la aplicación de AIR.

- Si se modifican determinadas propiedades visuales de un objeto HTMLLoader que contiene un documento PDF, el documento PDF será invisible. Entre estas propiedades se incluyen `filters`, `alpha`, `rotation` y `scaling`. El cambio de estas propiedades hace que el contenido PDF resulte invisible hasta que las propiedades se vuelvan a establecer. También es el caso si se modifican estas propiedades para un contenedor de objetos de visualización que contenga el objeto HTMLLoader.
- El contenido PDF solo se puede ver si la propiedad `scaleMode` del objeto Stage del objeto NativeWindow que incluye el contenido PDF se establece en `StageScaleMode.NO_SCALE`. Cuando se define con otro valor, el contenido PDF no podrá verse.
- Si se hace clic en un vínculo a contenido dentro del archivo PDF, se actualiza la posición de desplazamiento del contenido PDF. Si se hace clic en un vínculo a contenido situado fuera del archivo PDF, el objeto HTMLLoader se redirige al objeto que contiene el PDF (aunque el destino del vínculo sea una nueva ventana).
- Los flujos de trabajo de comentarios de PDF no funcionan en AIR.

Capítulo 29: Aspectos básicos de la interacción con el usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación puede crear interactividad utilizando ActionScript 3.0 para responder a la actividad del usuario. Para comprender esta sección es necesario conocer el modelo de eventos de ActionScript 3.0. Para obtener más información, consulte “Gestión de eventos” en la página 127.

Captura de entradas del usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La interacción con el usuario, ya sea con el teclado, el ratón, la cámara o una combinación de estos dispositivos, es la base de interactividad. En ActionScript 3.0, identificar la interacción del usuario y responder a la misma requieren básicamente detectar eventos.

La clase InteractiveObject, una subclase de la clase DisplayObject, proporciona la estructura común de eventos y la funcionalidad necesaria para controlar la interacción con el usuario. Las instancias de la clase InteractiveObject no se crean directamente. Los objetos de visualización, como SimpleButton, Sprite, TextField y diversos componentes de Flex y la herramienta de edición de Flash heredan su modelo de interacción con el usuario de esta clase y, por tanto, comparten una estructura común. Esto significa que las técnicas que aprenda y el código que escriba para controlar la interacción del usuario en un objeto derivado de InteractiveObject son aplicables a todos los demás.

Conceptos y términos importantes

Antes de continuar, es importante familiarizarse con los siguientes términos clave relacionados con la interacción del usuario:

Código de caracteres Código numérico que representa un carácter del juego de caracteres actual (asociado con la pulsación de una tecla en el teclado). Por ejemplo, "D" y "d" tienen distintos códigos de carácter, aunque se crean con la misma tecla en un teclado inglés de EE.UU.

Menú contextual Menú que aparece cuando un usuario hace clic con el botón derecho o utiliza una combinación específica de teclado-ratón. Los comandos de menú contextual suelen aplicarse directamente al objeto en el que se ha hecho clic. Por ejemplo, un menú contextual para una imagen puede contener un comando para mostrar la imagen en una ventana independiente y un comando para descargarla.

Selección Indicación de que un elemento seleccionado está activo y es el destino de una interacción de teclado o ratón.

Código de tecla Código numérico correspondiente a una tecla física del teclado.

Más temas de ayuda

InteractiveObject

Keyboard

Mouse

ContextMenu

Administración de la selección

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto interactivo puede recibir la selección mediante programación o mediante una acción del usuario. Además, si la propiedad `tabEnabled` se establece en `true`, el usuario puede pasar la selección de un objeto a otro mediante la tecla Tabulador. Hay que tener en cuenta que el valor de `tabEnabled` es `false` de manera predeterminada, salvo en los siguientes casos:

- En un objeto SimpleButton, el valor es `true`.
- Para un campo de texto de entrada, el valor es `true`.
- En un objeto Sprite o MovieClip con `buttonMode` establecido en `true`, el valor es `true`.

En cada una de estas situaciones, se puede añadir un detector de `FocusEvent.FOCUS_IN` o `FocusEvent.FOCUS_OUT` para proporcionar un comportamiento adicional cuando cambie la selección. Esto es especialmente útil en los campos de texto y formularios, pero también se puede utilizar en objetos Sprite, MovieClip o en cualquier objeto que herede de la clase InteractiveObject. El siguiente ejemplo muestra cómo activar el cambio de selección de un objeto a otro con la tecla Tabulador y cómo responder al evento de selección posterior. En este caso, cada cuadrado cambia de color cuando se selecciona.

Nota: *Flash Professional utiliza métodos abreviados de teclado para administrar la selección; por lo tanto, para simular correctamente la administración de la selección, hay que probar los archivos SWF en un navegador o AIR en lugar de en Flash.*

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
    square.x = startX;
```

```
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Detección de tipos de entrada

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Con Flash Player 10.1 y Adobe AIR 2, se introdujo la posibilidad de probar el entorno del motor de ejecución para comprobar la compatibilidad de tipos de entrada específicos. Es posible utilizar ActionScript para probar si el dispositivo en el que está implantado el motor de ejecución:

- Admite entrada con lápiz stylus o con los dedos (o no admite ninguna entrada táctil).
- Tiene un teclado virtual o físico para el usuario (o no tiene ningún teclado).
- Muestra un cursor (si no es así, las funciones que dependen de la posición del cursor sobre los objetos no funcionarán).

Las API de ActionScript de detección de entrada incluyen:

- Propiedad flash.system.Capabilities.touchScreenType: valor proporcionado en tiempo de ejecución que indica el tipo de entrada que se admite en el entorno actual.
- Clase flash.system.TouchScreenType : clase de constantes de valores de enumeración para la propiedad Capabilities.touchScreenType.
- Propiedad flash.ui.Mouse.supportsCursor: valor proporcionado en tiempo de ejecución que indica si hay disponible un cursor persistente o no.
- Propiedad flash.ui.Keyboard.physicalKeyboardType: valor proporcionado en tiempo de ejecución que indica si hay disponible un teclado completo físico, si solo hay disponible un teclado numérico o si no hay ningún teclado.
- Clase flash.ui.KeyboardType: clase de constantes de valores de enumeración para la propiedad flash.ui.Keyboard.physicalKeyboardType.
- Propiedad flash.ui.Keyboard.hasVirtualKeyboard: valor proporcionado en tiempo de ejecución que indica si se ofrece un teclado virtual al usuario (bien en lugar de un teclado físico o además del teclado físico).

Las API de detección de entrada permiten aprovechar las capacidades del dispositivo del usuario u ofrecer alternativas si dichas capacidades no están presentes. Estas API son especialmente útiles para desarrollar aplicaciones móviles y con tecnología táctil. Por ejemplo, si tiene una interfaz para dispositivo móvil con botones demasiado pequeños para un lápiz stylus, puede proporcionar una interfaz alternativa con botones más grandes que el usuario puede tocar con los dedos. El siguiente código es para una aplicación con una función llamada createStylusUI() que asigna un conjunto de elementos de interfaz de usuario indicados para lápices stylus. Otra función, llamada createTouchUI(), asigna otro conjunto de elementos de interfaz de usuario indicados para operaciones táctiles:

```
if(Capabilities.touchscreenType == TouchscreenType.STYLUS ){
    //Construct the user interface using small buttons for a stylus
    //and allow more screen space for other visual content
    createStylusUI();
} else if(Capabilities.touchscreenType = TouchscreenType.FINGER){
    //Construct the user interface using larger buttons
    //to capture a larger point of contact with the device
    createTouchUI();
}
```

Cuando desarrolle aplicaciones para distintos entornos de entrada, tenga en cuenta la siguiente tabla de compatibilidad:

Entorno	**supportsCursor**	**touchscreenType == TÁCTIL**	**touchscreenType == STYLUS**	**touchscreenType == NINGUNO**
Escritorio tradicional	true	false	false	true
Dispositivos táctiles capacitivos (tablets, PDA y teléfonos que detectan toques suaves de los dedos, como el iPhone de Apple o Palm Pre)	false	true	false	false
Dispositivos táctiles duros (tablets, PDA y teléfonos que detectan toques precisos y con presión, como el HTC Fuze)	false	false	true	false
Dispositivos no táctiles (teléfonos y dispositivos que pueden ejecutar aplicaciones pero que no disponen de pantallas táctiles)	false	false	false	true

Nota: *las distintas plataformas de dispositivos pueden admitir muchas combinaciones de tipos de entrada. Utilice esta tabla como guía general.*

Capítulo 30: Entradas de teclado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación puede capturar y responder a entradas de teclado y manipular un IME para que los usuarios puedan escribir caracteres de texto con una codificación distinta de ASCII en idiomas multibyte. Para comprender esta sección es necesario conocer el modelo de eventos de ActionScript 3.0. Para obtener más información, consulte "Gestión de eventos" en la página 127.

Para obtener información sobre el tipo de teclado admitido (físico, virtual, alfanumérico o numérico de 12 botones) en tiempo de ejecución, consulte "Detección de tipos de entrada" en la página 582.

Un editor de método de entrada (IME) permite a los usuarios introducir caracteres y símbolos complejos mediante un teclado estándar. Puede utilizar las clases IME para que los usuarios puedan aprovechar el IME de su sistema en las aplicaciones.

Más temas de ayuda

flash.events.KeyboardEvent

flash.system.IME

Captura de entradas de teclado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos de visualización que heredan su modelo de interacción de la clase InteractiveObject pueden responder a eventos del teclado mediante detectores de eventos. Por ejemplo, se puede colocar un detector de eventos en el escenario para detectar entradas de teclado y responder a las mismas. En el código siguiente, un detector de eventos captura una pulsación de tecla y se muestran las propiedades de nombre de tecla y código de tecla:

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Alguna teclas, como la tecla Ctrl, generan eventos aunque no tengan un glifo de representación.

En el ejemplo de código anterior, el detector de eventos del teclado captura la entrada de teclado para todo el escenario. También se puede escribir un detector de eventos para un objeto de visualización específico en el escenario; este detector de eventos se activaría cuando se seleccionara el objeto.

En el siguiente ejemplo, las pulsaciones de teclas se reflejan en el panel Salida cuando el usuario escribe en la instancia de TextField. Si se mantiene presionada la tecla Mayús, el color del borde de TextField cambiará temporalmente a rojo.

En este código se supone que hay una instancia de TextField denominada `tf` en el objeto Stage (el escenario).

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

La clase TextField también notifica un evento `textInput` que se puede detectar cuando un usuario introduce texto. Para obtener más información, consulte "Captura de entradas de texto" en la página 397.

Nota: *en el motor de ejecución de AIR se puede cancelar un evento keyboard. En el motor de ejecución de Flash Player, no se puede cancelar un evento keyboard.*

Códigos de teclas y códigos de caracteres

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede acceder a las propiedades `keyCode` y `charCode` de un evento de teclado para determinar la tecla que se ha presionado y activar a continuación otras acciones. La propiedad `keyCode` es un valor numérico que se corresponde con el valor de una tecla del teclado. La propiedad `charCode` es el valor numérico de la tecla en el juego de caracteres actual. (El juego de caracteres predeterminado es UTF-8, que es compatible con ASCII.)

La principal diferencia entre el código de tecla y los valores de caracteres es que el valor del código de tecla representa una tecla determinada del teclado (el 1 de un teclado es distinto del 1 de la fila superior, pero la tecla que genera "1" y la tecla que genera "!" es la misma) y el valor de carácter representa un carácter determinado (los caracteres R y r son distintos).

Nota: *para obtener información sobre las asignaciones entre las teclas y sus valores de código de caracteres en ASCII, consulte la clase flash.ui.Keyboard en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.*

Las correspondencias entre las teclas y sus códigos de tecla dependen del dispositivo y del sistema operativo. Por esta razón, no se deben utilizar asignaciones de teclas para activar acciones. Se deben utilizar los valores constantes predefinidos que proporciona la clase Keyboard para hacer referencia a las propiedades `keyCode` apropiadas. Por ejemplo, en lugar de utilizar la asignación de tecla para la tecla Mayús, se debe utilizar la constante `Keyboard.SHIFT` (como se indica en el ejemplo de código anterior).

Precedencia de KeyboardEvent

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al igual que con los otros eventos, la secuencia de eventos de teclado se determina por la jerarquía de objetos de visualización, no en el orden en que se asignan métodos `addEventListener()` en el código.

Por ejemplo, supongamos que se coloca un campo de texto `tf` dentro de un clip de película denominado `container` y que se añade un detector de eventos para un evento de teclado en ambas instancias, como se muestra en el siguiente ejemplo:

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode)
+ " (key code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

Debido a que hay un detector en el campo de texto y en su contenedor principal, se llama dos veces a la función `reportKeyDown()` por cada pulsación en el campo de texto. Hay que tener en cuenta que por cada tecla pulsada, el campo de texto distribuye un evento antes de que el clip de película `contenedor` distribuya un evento.

El sistema operativo y el navegador web procesarán los eventos de teclado antes que Adobe Flash Player o AIR. Por ejemplo, en Microsoft Internet Explorer, si se pulsan Ctrl+W se cierra la ventana del navegador antes de que un archivo SWF contenido pueda distribuir un evento de teclado.

Uso de la clase IME

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase IME permite manipular el IME del sistema operativo en Flash Player o Adobe AIR.

También se puede utilizar código ActionScript para determinar lo siguiente:

- Si hay instalado un IME en el equipo del usuario (`Capabilities.hasIME`).
- Si el IME está activado o desactivado en el equipo del usuario (`IME.enabled`).
- El modo de conversión utilizado por el IME actual (`IME.conversionMode`).

Se puede asociar un campo de entrada de texto con un contexto de IME específico. Al cambiar de un campo de entrada a otro, también se puede cambiar el IME a Hiragana (Japonés), números de anchura completa, números de anchura media, entrada directa, etc..

Un IME permite al usuario introducir caracteres de texto en idiomas multibyte, como chino, japonés o coreano, con una codificación distinta de ASCII.

Para más información sobre los IME, consulte la documentación del sistema operativo para el que desarrolla la aplicación. Para ver recursos adicionales, consulte los siguientes sitios web:

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/library/mac/navigation/

- http://www.java.sun.com/

Nota: *si un IME no está activo en el ordenador del usuario, fallarán todas las llamadas a los métodos o propiedades IME que no sean* `Capabilities.hasIME`*. Tras activar manualmente un IME, las siguientes llamadas de ActionScript a métodos y propiedades IME funcionarán como se esperaba. Por ejemplo, si se utiliza un IME japonés, debe estar activado antes de llamar a cualquier método o propiedad del IME.*

Comprobar si un IME está instalado y activado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Antes de llamar a cualquiera de los métodos o propiedades del IME, siempre se debe comprobar si en el equipo del usuario hay actualmente un IME instalado y activado. El código siguiente ilustra la manera de comprobar que el usuario tiene un IME instalado y activado antes de llamar a sus métodos:

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

El código anterior comprueba primero si el usuario tiene un IME instalado mediante la propiedad `Capabilities.hasIME`. Si esta propiedad está establecida en `true`, el código consulta la propiedad `IME.enabled` para comprobar si el IME del usuario está activado actualmente.

Determinar el modo de conversión del IME activado actualmente

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al crear aplicaciones multilingües, es posible que sea necesario determinar el modo de conversión que el usuario tiene activo actualmente. El código siguiente ilustra la manera de comprobar si el usuario tiene un IME instalado y, en caso afirmativo, qué modo de conversión de IME está activo actualmente:

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

El código anterior comprueba primero si el usuario tiene un IME instalado. A continuación comprueba el modo de conversión utilizado por el IME actual comparando el valor de la propiedad `IME.conversionMode` con cada una de las constantes de la clase IMEConversionMode.

Definición del modo de conversión del IME

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se cambia el modo de conversión del IME del usuario, es necesario asegurarse de que el código está incluido en un bloque `try..catch`, ya que definir un modo de conversión utilizando la propiedad `conversionMode` puede generar un error si el IME no puede establecer el modo de conversión. El código siguiente muestra cómo utilizar un bloque `try..catch` al establecer la propiedad `IME.conversionMode`:

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

El código anterior crea primero un campo de texto, que se utiliza para mostrar un mensaje de estado al usuario. A continuación, si el IME está instalado, el código lo activa y establece el modo de conversión en Coreano. Si el equipo del usuario no tiene un IME coreano instalado, Flash Player o AIR emite un error que es detectado por el bloque `try..catch`. El bloque `try..catch` muestra el mensaje de error en el campo de texto creado previamente.

Desactivar el IME para determinados campos de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En algunos casos, es posible que se desee desactivar el IME del usuario mientras este escribe caracteres. Por ejemplo, si hay un campo de texto que solo acepta entradas numéricas, es posible que no interese que aparezca el IME y ralentice la entrada de datos.

En el ejemplo siguiente se ilustra la manera de detectar eventos `FocusEvent.FOCUS_IN` y `FocusEvent.FOCUS_OUT`, y de desactivar el IME del usuario:

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

Este ejemplo crea dos campos de entrada de texto, `phoneTxt` y `nameTxt`, y después añade dos detectores de eventos al campo de texto `phoneTxt`. Cuando el usuario establece la selección en el campo de texto `phoneTxt`, se distribuye un evento `FocusEvent.FOCUS_IN` y se desactiva el IME. Cuando el campo de texto `phoneTxt` deja de estar seleccionado, se distribuye el evento `FocusEvent.FOCUS_OUT` para volver a activar el IME.

Detección de eventos de composición del IME

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se establece una cadena de composición, se distribuyen eventos de composición del IME. Por ejemplo, si el usuario activa su IME y escribe una cadena en japonés, se distribuye el evento `IMEEvent.IME_COMPOSITION` en cuanto el usuario seleccione la cadena de composición. Para poder detectar el evento `IMEEvent.IME_COMPOSITION`, hay que añadir un detector de eventos a la propiedad estática `ime` de la clase System (`flash.system.System.ime.addEventListener(...)`), como se indica en el siguiente ejemplo:

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

El código anterior crea dos campos de texto y los añade a la lista de visualización. El primer campo de texto, `inputTxt`, es un campo de entrada de texto que permite al usuario escribir texto en japonés. El segundo campo de texto, `outputTxt`, es un campo de texto dinámico que muestra mensajes de error al usuario o reproduce la cadena en japonés que el usuario escribe en el campo de texto `inputTxt`.

Teclados virtuales

Flash Player 10.2 y posterior, AIR 2.6 y posterior

Los dispositivos móviles, como teléfonos y tabletas, suelen incluir un teclado virtual y no uno físico. Las clases de la API Flash permiten al usuario:

- Detectar cuándo aparece el teclado virtual y cuándo se cierra.
- Impedir que el teclado aparezca.

- Determinar el área del escenario que queda oscurecida cuando aparece el teclado virtual.
- Crear objetos interactivos que hacen que aparezca el teclado virtual cuando se seleccionan. (No se admite en aplicaciones de AIR en iOS.)
- (Solo AIR) Desactivar el comportamiento de desplazamiento automático para que la aplicación pueda modificar su propia visualización y dejar sitio para el teclado.

Control del comportamiento del teclado virtual

El motor de ejecución abre automáticamente el teclado virtual cuando el usuario toca dentro de un campo de texto o un objeto interactivo configurado de forma especial. Cuando se abre el teclado, el motor de ejecución sigue las convenciones de la plataforma nativa a la hora de desplazar y cambiar el tamaño del contenido de la aplicación para que el usuario pueda ver el texto conforme escribe.

Cuando se abre el teclado, el objeto seleccionado distribuye los siguientes eventos en secuencia:

Evento `softKeyboardActivating`: se distribuye inmediatamente antes de que el teclado comience a elevarse sobre el escenario. Si se llama al método `preventDefault()` del objeto de evento distribuido, el teclado virtual no se abrirá.

Evento `softKeyboardActivate`: se distribuye un vez que el control del evento `softKeyboardActivating` se haya completado. Cuando el objeto seleccionado distribuye este evento, la propiedad `softKeyboardRect` del objeto Stage se ha actualizado para reflejar el área del escenario oscurecida por el teclado virtual. Este evento no se puede cancelar.

Nota: *si el teclado cambia de tamaño como, por ejemplo, cuando el usuario cambia el tipo de teclado, el objeto seleccionado distribuye un segundo evento softKeyboardActivate.*

Evento `softKeyboardDeactivate`: se distribuye cuando el teclado virtual se cierra por cualquier motivo. Este evento no se puede cancelar.

El siguiente ejemplo añade dos objetos TextField en el escenario. El objeto TextField superior evita que el teclado se eleve cuando se toca el campo y lo cierra si ya se ha elevado. El objeto TextField inferior muestra el comportamiento predeterminado. El ejemplo indica los eventos de teclado de pantalla distribuidos por ambos campos de texto.

```
package
{
import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.SoftKeyboardEvent;
public class SoftKeyboardEventExample extends Sprite
{
    private var tf1:TextField = new TextField();
    private var tf2:TextField = new TextField();

    public function SoftKeyboardEventExample()
    {
        tf1.width = this.stage.stageWidth;
        tf1.type = TextFieldType.INPUT;
        tf1.border = true;
        this.addChild( tf1 );

        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, preventSoftKe
yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, preventSoftKe
yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, preventSoftKeyboard
);
```

```
        tf2.border = true;
        tf2.type = TextFieldType.INPUT;
        tf2.width = this.stage.stageWidth;
        tf2.y = tf1.y + tf1.height + 30;
        this.addChild( tf2 );

        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, allowSoftKeyboard);
    }

    private function preventSoftKeyboard( event:SoftKeyboardEvent ):void
    {
            event.preventDefault();
            this.stage.focus = null; //close the keyboard, if raised
            trace( "tf1 dispatched: " + event.type + " -- " + event.triggerType );
    }
    private function allowSoftKeyboard( event:SoftKeyboardEvent )    :void
    {
            trace( "tf2 dispatched: " + event.type + " -- " + event.triggerType );
    }
}
}
```

Adición de compatibilidad con el teclado virtual para los objetos interactivos

Flash Player 10.2 y versiones posteriores, AIR 2.6 y versiones posteriores (pero no se admite en iOS)

Normalmente, el teclado virtual solo se abre cuando se toca un objeto TextField. Puede configurar una instancia de la clase InteractiveObject para abrir el teclado virtual cuando se seleccione.

Para configurar una instancia de InteractiveObject y abrir el teclado virtual defina su propiedad `needsSoftKeyboard` como `true`. Donde el objeto esté asignado a la propiedad focus del escenario, se abrirá automáticamente el teclado virtual. Además, es posible mostrar y ocultar el teclado llamando al método `requestSoftKeyboard()` de InteractiveObject.

El siguiente ejemplo muestra cómo programar un objeto InteractiveObject para que actúe como campo de entrada de texto. La clase TextInput mostrada en el ejemplo define la propiedad `needsSoftKeyboard` para que el teclado aparezca cuando sea necesario. Seguidamente, el objeto detecta eventos `keyDown` e inserta el carácter escrito en el campo.

El ejemplo utiliza el motor de texto de Flash para agregar y visualizar cualquier texto introducido y controla los eventos importantes. Por motivos de simplicidad, el ejemplo no implementa un campo de texto con todas sus funciones.

```
package  {
    import flash.geom.Rectangle;
    import flash.display.Sprite;
    import flash.text.engine.TextElement;
    import flash.text.engine.TextBlock;
    import flash.events.MouseEvent;
    import flash.events.FocusEvent;
    import flash.events.KeyboardEvent;
    import flash.text.engine.TextLine;
    import flash.text.engine.ElementFormat;
    import flash.events.Event;

    public class TextInput extends Sprite
    {

        public var text:String = " ";
        public  var textSize:Number = 24;
        public var textColor:uint = 0x000000;
        private var _bounds:Rectangle = new Rectangle( 0, 0, 100, textSize );
        private var textElement: TextElement;
        private var textBlock:TextBlock = new  TextBlock();

        public function TextInput( text:String = "" )
        {
            this.text = text;
            this.scrollRect = _bounds;
            this.focusRect= false;

            //Enable keyboard support
            this.needsSoftKeyboard = true;
            this.addEventListener(MouseEvent.MOUSE_DOWN, onSelect);
            this.addEventListener(FocusEvent.FOCUS_IN, onFocusIn);
            this.addEventListener(FocusEvent.FOCUS_OUT, onFocusOut);

            //Setup text engine
            textElement = new TextElement( text, new ElementFormat( null, textSize, textColor ) );
            textBlock.content = textElement;
            var firstLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8 );
            firstLine.x = 4;
            firstLine.y = 4 + firstLine.totalHeight;
            this.addChild( firstLine );

        }

        private function onSelect( event:MouseEvent ):void
        {
            stage.focus = this;
        }
        private function onFocusIn( event:FocusEvent ):void
        {
            this.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onFocusOut( event:FocusEvent ):void
        {
            this.removeEventListener( KeyboardEvent.KEY_UP, onKey );
        }
```

```
        private function onKey( event:KeyboardEvent ):void
        {
            textElement.replaceText( textElement.text.length, textElement.text.length,
String.fromCharCode( event.charCode ) );
            updateText();
        }
        public function set bounds( newBounds:Rectangle ):void
        {
            _bounds = newBounds.clone();
            drawBackground();
            updateText();
            this.scrollRect = _bounds;

            //force update to focus rect, if needed
            if( this.stage!= null && this.focusRect && this.stage.focus == this )
                this.stage.focus = this;
        }

        private function updateText():void
        {
            //clear text lines
            while( this.numChildren > 0 ) this.removeChildAt( 0 );

            //and recreate them
            var textLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8);
            while ( textLine)
            {
                textLine.x = 4;
                if( textLine.previousLine != null )
                {
                    textLine.y = textLine.previousLine.y +
                                 textLine.previousLine.totalHeight + 2;
                }
                              else
                {
                    textLine.y = 4 + textLine.totalHeight;
                }
                this.addChild(textLine);
                textLine = textBlock.createTextLine(textLine, _bounds.width - 8 );
            }
        }

        private function drawBackground():void
        {
            //draw background and border for the field
            this.graphics.clear();
            this.graphics.beginFill( 0xededed );
            this.graphics.lineStyle( 1, 0x000000 );
            this.graphics.drawRect( _bounds.x + 2, _bounds.y + 2, _bounds.width - 4,
_bounds.height - 4);
            this.graphics.endFill();
        }
    }
}
```

La siguiente clase de aplicación principal muestra cómo usar la clase TextInput y gestiona el diseño de la aplicación cuando el teclado aparece o cambia la orientación del dispositivo. La clase principal crea un objeto TextInput y define sus límites para que llene todo el escenario. La clase ajusta el tamaño del objeto TextInput si aparece el teclado o el escenario cambia de tamaño. La clase detecta eventos del teclado virtual desde el objeto TextInput y cambia de tamaño los eventos del escenario. Independientemente de la causa del evento, la aplicación determina el área visible del escenario y cambia de tamaño el control de entrada para llenarla. Naturalmente, en una aplicación real sería necesario usar un algoritmo de diseño mucho más sofisticado.

```
package  {

    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.geom.Rectangle;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;

    public class CustomTextField extends MovieClip {

        private var customField:TextInput = new TextInput("Input text: ");

        public function CustomTextField() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;
            this.addChild( customField );
            customField.bounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );

            //track soft keyboard and stage resize events
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onDisplayAreaChange );
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onDisplayAreaChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            //Fill the stage if possible, but avoid the area covered by a keyboard
            var desiredBounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );
            if( this.stage.stageHeight - this.stage.softKeyboardRect.height <
desiredBounds.height )
                desiredBounds.height = this.stage.stageHeight -
this.stage.softKeyboardRect.height;

            customField.bounds = desiredBounds;
        }
    }
}
```

Nota: *el escenario solo distribuye eventos de cambio de tamaño como respuesta a un cambio de orientación si la propiedad* `scaleMode` *se establece como* `noScale`*. En otros modos, las dimensiones del escenario no cambian; en su lugar, se escala el contenido para compensar.*

Control de cambios de la pantalla de la aplicación

AIR 2.6 y posterior

En AIR es posible desactivar el comportamiento de desplazamiento y cambio de tamaño asociado a la aparición de un teclado virtual si se establece el elemento `softKeyboardBehavior` del archivo descriptor de la aplicación como `none`:

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

Cuando se desactiva el comportamiento predeterminado, es responsabilidad de la aplicación llevar a cabo los ajustes necesarios para la visualización. Cuando se muestra el teclado, se distribuye un evento softKeyboardActivate. Cuando se distribuye el evento `softKeyboardActivate`, la propiedad `softKeyboardRect` del escenario contiene las dimensiones del área oscurecida por el teclado abierto. Utilice estas dimensiones para mover o cambiar de tamaño el contenido y que se visualice correctamente cuando se muestra el teclado y el usuario está escribiendo. (Cuando el teclado se cierra, las dimensiones del rectángulo softKeyboardRect son todas de cero.)

Cuando el teclado se cierra, se distribuye un evento `softKeyboardDeactivate` y la pantalla de la aplicación puede volver a normal.

```
package  {
    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;
    import flash.display.InteractiveObject;
    import flash.text.TextFieldType;
    import flash.text.TextField;

    public class PanningExample extends MovieClip {

        private var textField:TextField = new TextField();

        public function PanningExample() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;

            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
            textField.height = 200;
            textField.type = TextFieldType.INPUT;
            textField.border = true;
            textField.wordWrap = true;
            textField.multiline = true;

            this.addChild( textField );

            //track soft keyboard and stage resize events
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onKeyboardChange );
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onKeyboardChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
```

```
            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
        }

        private function onKeyboardChange( event:SoftKeyboardEvent ):void
        {
            var field:InteractiveObject = textField;
            var offset:int = 0;

            //if the softkeyboard is open and the field is at least partially covered
            if( (this.stage.softKeyboardRect.y != 0) && (field.y + field.height >
this.stage.softKeyboardRect.y) )
                offset = field.y + field.height - this.stage.softKeyboardRect.y;

            //but don't push the top of the field above the top of the screen
            if( field.y - offset < 0 ) offset += field.y - offset;

            this.y = -offset;
        }
    }
}
```

Nota: *en Android, hay casos (incluido el modo de pantalla completa) en los que el sistema operativo no expone las dimensiones exactas del teclado. En esa situación, no queda más remedio que calcular el tamaño. También, en orientación horizontal, el teclado IME nativo en pantalla completa se utiliza para la introducción de texto. Este teclado IME cuenta con un campo de introducción de texto incorporado y oscurece todo el escenario. No hay modo alguno de visualizar un teclado horizontal que no rellene la pantalla.*

Capítulo 31: Entradas de ratón

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación puede crear interactividad capturando y respondiendo a las entradas de ratón. Para comprender esta sección es necesario conocer el modelo de eventos de ActionScript 3.0. Para obtener más información, consulte "Gestión de eventos" en la página 127.

Para obtener más información sobre el tipo de ratón admitido (cursor persistente, lápiz stylus o entrada táctil) en tiempo de ejecución, consulte "Detección de tipos de entrada" en la página 582.

Más temas de ayuda

flash.ui.Mouse

flash.events.MouseEvent

"Entrada táctil, multitáctil y de gestos" en la página 606

Captura de entradas de ratón

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los clics del ratón crean eventos de ratón que pueden utilizarse para activar la funcionalidad de interacción. Se puede añadir un detector de eventos al objeto Stage para detectar los eventos de ratón que se produzcan en cualquier parte del archivo SWF. También se pueden añadir detectores de eventos a objetos del escenario que heredan de InteractiveObject (por ejemplo, Sprite o MovieClip); estos detectores se activan cuando se hace clic en el objeto.

Tal y como sucede con los eventos de teclado, los eventos de ratón se propagarán. En el siguiente ejemplo, dado que `square` es un objeto secundario de Stage, el evento se distribuirá tanto desde el objeto Sprite `square` como desde el objeto Stage cuando se haga clic en el cuadrado:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" +
event.localX + "," + event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

En el ejemplo anterior, el evento de ratón contiene información de posición sobre el clic. Las propiedades `localX` y `localY` contienen la ubicación donde tuvo lugar el clic en el objeto secundario inferior de la cadena de visualización. Por ejemplo, si se hace clic en la esquina superior izquierda de `square`, se notifican las coordenadas locales [0,0], ya que es el punto de registro de `square`. Como alternativa, las propiedades `stageX` y `stageY` hacen referencia a las coordenadas globales del clic en el escenario. El mismo clic notifica [50,50] para estas coordenadas, ya que `square` se movió a estas coordenadas. Ambos pares de coordenadas pueden ser útiles, dependiendo de cómo se desee responder a la interacción del usuario.

Nota: *en modo de pantalla completa, es posible configurar la aplicación para usar el bloqueo del ratón. El bloqueo del ratón desactiva el cursor y permite el movimiento ilimitado del ratón. Para obtener más información, consulte* *"Trabajo con el modo de pantalla completa" en la página 171.*

El objeto MouseEvent también contiene las propiedades booleanas `altKey`, `ctrlKey` y `shiftKey`. Se pueden utilizar estas propiedades para comprobar si también se está pulsando la tecla Alt, Ctrl o Mayús a la vez que se hace clic con el ratón.

Arrastre de objetos Sprites alrededor del escenario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede permitir a los usuarios arrastrar un objeto Sprite alrededor del escenario utilizando el método `startDrag()` de la clase Sprite. Esto se muestra en el siguiente ejemplo de código:

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

Para obtener más información, consulte la sección sobre la creación de interacción con el arrastre de ratón en "Cambio de posición" en la página 178.

Arrastrar y colocar en AIR

En Adobe AIR, se puede habilitar la compatibilidad con las operaciones de arrastrar y colocar para que los usuarios puedan arrastrar datos dentro y fuera de la aplicación. Para obtener más información, consulte "Operación de arrastrar y colocar en AIR" en la página 633.

Personalización del cursor del ratón

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El cursor (puntero) del ratón puede ocultarse o cambiarse por cualquier objeto de visualización en el escenario. Para ocultar el cursor del ratón, es necesario llamar al método `Mouse.hide()`. Para personalizar el cursor hay que llamar a `Mouse.hide()`, detectar el evento `MouseEvent.MOUSE_MOVE` en el objeto Stage y establecer las coordenadas de un objeto de visualización (el cursor personalizado) en las propiedades `stageX` y `stageY` del evento. El ejemplo siguiente muestra una ejecución básica de esta tarea:

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

Ejemplo de entrada de ratón: WordSearch

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En este ejemplo se ilustra la interacción del usuario mediante la gestión de eventos del ratón. Los usuarios deben crear la mayor cantidad posible de palabras a partir de una cuadrícula aleatoria de letras moviéndose horizontalmente o verticalmente por la cuadrícula, pero no pueden utilizar dos veces una misma letra. Este ejemplo muestra las siguientes características de ActionScript 3.0:

- Generación de una cuadrícula de componentes de forma dinámica
- Respuesta a eventos de ratón
- Mantenimiento de una puntuación según la interacción del usuario

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación WordSearch se encuentran en la carpeta Samples/WordSearch. La aplicación consta de los siguientes archivos:

Archivo	Descripción
WordSearch.as	Clase que proporciona la funcionalidad principal de la aplicación.
WordSearch.fla o WordSearch.mxml	Archivo principal de la aplicación Flex (MXML) o Flash (FLA).
dictionary.txt	Un archivo que se utiliza para determinar si las palabras escritas puntúan y están bien escritas.

Carga de un diccionario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear un juego consistente en encontrar palabras hay que utilizar un diccionario. En el ejemplo se incluye un archivo de texto denominado dictionary.txt, que contiene una lista de palabras separadas por retornos de carro. Después de crear un conjunto denominado `words`, la función `loadDictionary()` solicita este archivo y, una vez cargado correctamente, lo convierte en una cadena larga. Esta cadena se puede analizar para generar un conjunto de palabras con el método `split()`, dividiéndola en cada instancia del retorno de carro (código de carácter 10) o línea nueva (código de carácter 13). Este análisis se realiza en la función `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

Creación de la interfaz de usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una vez almacenadas las palabras, se puede configurar la interfaz de usuario. Cree dos instancias de Button: una para enviar una palabra y otra para borrar una palabra que ya se ha formado. En cada caso, hay que responder a entradas de usuario detectando el evento `MouseEvent.CLICK` difundido por el botón y llamando a continuación a la función. En la función `setupUI()`, este código crea los detectores en los dos botones:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Generación de un tablero de juego

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El tablero del juego es una cuadrícula de letras aleatorias. En la función `generateBoard()` se crea una cuadrícula bidimensional anidando un bucle dentro de otro. El primer bucle incrementa las filas y el segundo incrementa el número total de columnas por fila. Cada una de las celdas creadas por estas filas y columnas contiene un botón que representa una letra del tablero.

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number,
totalCols:Number, buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

Aunque solo hay una línea que añade un detector para un evento `MouseEvent.CLICK`, como está en un bucle `for` se asignará un detector a cada instancia de Button. Además, a cada botón se le asigna un nombre derivado de su posición de fila y columna, lo que proporciona una manera sencilla de hacer referencia a la fila y columna de cada botón en otras partes del código.

Creación de palabras a partir de entradas de usuario

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las palabras pueden escribirse seleccionando letras contiguas horizontal o verticalmente, pero nunca se puede usar dos veces la misma letra. Cada clic genera un evento de ratón y hace que se compruebe que la palabra que el usuario está escribiendo es la continuación correcta de las letras en las que se hizo clic previamente. En caso contrario, se elimina la palabra anterior y se inicia otra nueva. Esta comprobación se produce en el método `isLegalContinuation()`.

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") +
3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") +
3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") +
3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") +
3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
             (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

Los métodos `charAt()` e `indexOf()` de la clase String recuperan las filas y columnas adecuadas del botón en el que se acaba de hacer clic y del botón en el que se hizo clic previamente. El método `isLegalContinuation()` devuelve `true` si la fila o columna no cambia, y si la fila o columna que ha cambiado corresponde a un solo incremento con respecto a la anterior. Si se desea cambiar las reglas del juego y permitir la formación de palabras en diagonal, se pueden eliminar las comprobaciones de filas o columnas que no cambian; la línea final sería como la siguiente:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

Comprobación de las palabras formadas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para completar el código del juego son necesarios mecanismos para comprobar las palabras formadas y actualizar la puntuación. El método `searchForWord()` realiza estas funciones:

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

Esta función recorre todas las palabras del diccionario. Si la palabra del usuario coincide con una palabra del diccionario, se devuelve su posición en el diccionario. El método `submitWord()` comprueba entonces la respuesta y actualiza la puntuación si la posición es válida.

Personalización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al principio de la clase hay varias constantes. Se puede modificar el juego modificando estas variables. Por ejemplo, se puede cambiar la cantidad de tiempo de juego incrementando la variable `TOTAL_TIME`. O se puede incrementar ligeramente la variable `PERCENT_VOWELS` para aumentar la probabilidad de encontrar palabras.

Capítulo 32: Entrada táctil, multitáctil y de gestos

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Las funciones de control de eventos táctiles de la plataforma Flash incluyen entrada desde uno o varios puntos de contacto en dispositivos táctiles. El motor de ejecución de Flash gestiona eventos que combinan varios puntos de contacto con movimiento para crear un gesto. En otras palabras, Flash interpreta dos tipos de entrada:

Táctil Entrada con un solo dispositivo señalador como un dedo, un lápiz stylus u otra herramienta en un dispositivo táctil. Algunos dispositivos admiten varios puntos simultáneos de contacto con un dedo o un lápiz stylus.

Multitáctil Entrada con más de un punto simultáneo de contacto.

Gesto Entrada interpretada por un dispositivo o sistema operativo como respuesta a uno o varios eventos táctiles. Por ejemplo, un usuario gira dos dedos a la vez y el dispositivo o sistema operativo interpreta que dicha entrada táctil es un gesto de giro. Algunos gestos se realizan con un dedo o punto de contacto y otros requieren varios puntos táctiles. El dispositivo o sistema operativo establece el tipo de gesto que se asigna a la entrada.

Tanto la entrada táctil como la de gesto puede ser multitáctil en función del dispositivo del usuario. ActionScript proporciona una API para gestionar eventos táctiles, eventos de gesto y para hacer un seguimiento individual de los eventos táctiles con entrada multitáctil.

Nota: *la detección eventos táctiles y de gestos puede consumir una cantidad importante de recursos de procesamiento (equivalentes al procesamiento de varios fotogramas por segundo), dependiendo del dispositivo informático y del sistema operativo. Suele ser mejor utilizar los eventos del ratón cuando realmente no es necesaria la funcionalidad adicional que proporcionan los gestos o el toque. Cuando se utilizan eventos de gestos o táctiles, considere la reducción de cambios gráficos que pueden suceder, especialmente cuando estos eventos se pueden distribuir rápidamente, durante una operación de zoom, giro o panorámica. Por ejemplo, se puede detener la animación en un componente mientras que el usuario lo cambia de tamaño utilizando un gesto de zoom.*

Más temas de ayuda

flash.ui.Multitouch

flash.events.TouchEvent

flash.events.GestureEvent

flash.events.TransformGestureEvent

flash.events.GesturePhase

flash.events.PressAndTapGestureEvent

Paul Trani: Touch Events and Gestures on Mobile (Eventos táctiles y de gestos en Mobile; en inglés)

Mike Jones: Virtual Game Controllers

Fundamentos de la entrada táctil

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Cuando la plataforma Flash se ejecuta en un entorno que admite entrada táctil, las instancias de InteractiveObject pueden detectar eventos táctiles y llamar a los controladores. Generalmente, los eventos táctiles, multitáctiles y de gestos se gestionan del mismo modo que otros eventos en ActionScript (consulte "Gestión de eventos" en la página 127 para obtener información básica sobre la gestión de eventos con ActionScript).

No obstante, para que el motor de ejecución de Flash pueda interpretar un gesto táctil, debe estar ejecutándose en un entorno de hardware y software que admite la entrada táctil y multitáctil. Consulte "Detección de tipos de entrada" en la página 582 para ver un gráfico en el que se comparan los distintos tipos de pantallas táctiles. Además, si el motor de ejecución se ejecuta en una aplicación contenedora (por ejemplo, un navegador), el contenedor transfiere la entrada al motor de ejecución. En algunos casos, el entorno de hardware y sistema operativo actual admiten gestos multitáctiles, pero el navegador que contiene el motor de ejecución de Flash interpreta la entrada y no la transfiere al motor de ejecución. O simplemente obvia toda la entrada.

En el siguiente diagrama se muestra el flujo de entrada del usuario al motor de ejecución:

Flujo de entrada del usuario al motor de ejecución de la plataforma Flash

Afortunadamente, la API de ActionScript para desarrollar aplicaciones táctiles incluye clases, métodos y propiedades para determinar la compatibilidad de entrada táctil y multitáctil en el entorno del motor de ejecución. La API utilizada para determinar la compatibilidad de entrada táctil es la "API de detección" para la gestión de eventos táctiles.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes para la creación de aplicaciones de control de eventos táctiles:

API de detección Los métodos y propiedades utilizados para probar si el entorno del motor de ejecución admite eventos táctiles y otros modos de entrada.

Evento táctil Acción de entrada realizada en un dispositivo táctil con un único punto de contacto.

Punto de contacto Punto de contacto de un único evento táctil. Incluso si el dispositivo no admite entrada de gestos, puede admitir varios puntos táctiles simultáneos.

Secuencia táctil Serie de eventos que representan todo un toque individual. Estos eventos incluyen un movimiento inicial, ninguno o varios, o uno al final.

Evento multitáctil Una acción de entrada realizada en un dispositivo táctil con varios puntos de contacto (por ejemplo, más de un dedo).

Evento de gesto Una acción de entrada realizada en un dispositivo táctil con movimientos complejos. Por ejemplo, un gesto es tocar una pantalla con dos dedos y moverlos simultáneamente alrededor del perímetro de un círculo imaginario para indicar un giro.

Fases Puntos concretos de tiempo en el flujo del evento (como el punto inicial y el final).

Lápiz stylus Herramienta para interactuar con una pantalla táctil. Un lápiz stylus aporta más precisión que el dedo humano. Algunos dispositivos solo reconocen la entrada de determinados tipos de lápiz stylus. Los dispositivos no reconocen entrada de lápiz stylus tal vez no reconozcan varios puntos de contacto simultáneos o con los dedos.

Pulsar y tocar Tipo de gesto de entrada multitáctil donde el usuario presiona un dedo en el dispositivo táctil y toca con otro dedo o dispositivo señalador. Este gesto suele simular la acción de clic con el botón derecho del ratón en las aplicaciones multitáctiles.

La estructura de la API de entrada táctil

La API de entrada táctil de ActionScript está diseñada para abordar el hecho de que el control de la entrada táctil depende del entorno de hardware y software del motor de ejecución de Flash. La API de entrada táctil básicamente aborda tres necesidades de implementación de aplicaciones táctiles: la detección, los eventos y las fases. Coordine estas API para producir una experiencia de usuario predecible y fiable; incluso si ignora qué dispositivo se utilizará cuando está desarrollando una aplicación.

Detección

La API de detección permite probar el entorno de hardware y software en el motor de ejecución. Los valores proporcionados por el motor de ejecución determinan la entrada táctil disponible en el motor de ejecución de Flash en ese contexto. Asimismo, puede utilizar las propiedades y métodos de detección para que la aplicación reaccione ante eventos de ratón (no solo ante eventos táctiles, por si alguna entrada táctil no se admite en el entorno). Para obtener más información, consulte "Detección con soporte táctil" en la página 609.

Eventos

ActionScript gestiona eventos de entrada táctil con detectores y controladores de eventos, como lo hace con los demás. Sin embargo, también se debe tener en cuenta la gestión de eventos de entrada táctil:

- El dispositivo o el sistema operativo puede interpretar un toque de varias formas, bien como una secuencia de toques o, de forma colectiva, como un gesto.
- Un solo toque en un dispositivo táctil (con un dedo, lápiz stylus o dispositivo señalador) siempre distribuye también un evento de ratón. Puede gestionar el evento de ratón con los tipos de eventos de la clase MouseEvent. O puede diseñar la aplicación para que solo responda a eventos táctiles. También puede diseñar una aplicación para que responda a ambos.
- Una aplicación puede responder a varios eventos táctiles simultáneos y controlar cada uno de ellos por separado.

Normalmente, la API de detección se utiliza para controlar condicionalmente eventos gestionados por la aplicación y para hacerlo de un modo determinado. Una vez que la aplicación conoce el entorno del motor de ejecución, puede llamar al controlador adecuado o establecer el objeto de evento correcto cuando el usuario interactúe con la aplicación. O la aplicación puede indicar que una entrada concreta no se puede controlar en el entorno actual y ofrecer al usuario una alternativa o información. Para obtener más información, consulte "Gestión de eventos táctiles" en la página 610 y "Gestión de eventos de gestos" en la página 615.

Fases

En aplicaciones táctiles y multitáctiles, los objetos de eventos táctiles contienen propiedades para hacer un seguimiento de las fases de la interacción del usuario. Escriba ActionScript para gestionar fases como la inicial, la de actualización o la final de la entrada del usuario para que pueda lograrlo. Responda a fases de eventos para que los objetos visuales cambien a medida que el usuario toca y mueve el punto táctil en la pantalla. O utilice las fases para realizar un seguimiento de las propiedades concretas de un gesto a medida que este evoluciona.

En eventos de puntos táctiles, observe el tiempo que el usuario permanece en un objeto interactivo concreto. Una aplicación puede realizar el seguimiento de fases con varios puntos táctiles simultáneos por separado y gestionar cada uno como corresponda.

En un gesto, la información específica sobre la transformación del gesto se interpreta a media que ocurre. Siga las coordenadas del punto de contacto (o varios, si hay más de uno) a medida que se mueve por la pantalla.

Detección con soporte táctil

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Utilice las propiedades de la clase Multitouch para definir el ámbito de entrada táctil admitido por la aplicación. Seguidamente, pruebe el entorno para garantizar que los eventos gestionados por ActionScript se admiten. En concreto, primero establezca el tipo de entrada táctil de su aplicación. Las opciones son: punto táctil, gesto o ninguno (interprete todas las entradas táctiles como clics de ratón y utilice solo controladores de evento de ratón). Luego utilice las propiedades y métodos de la clase Multitouch para asegurarse de que el motor de ejecución se encuentra en un entorno que admite la entrada táctil requerida por la aplicación. Pruebe el entorno del motor de ejecución para ver si admite los tipos de entrada táctil (o si puede interpretar gestos) y responda en consecuencia.

Nota: *las propiedades de la clase Multitouch son propiedades static y no pertenecen a instancias de ninguna clase. Utilícelas con la sintaxis Multitouch.property, por ejemplo:*

```
var touchSupport:Boolean = Multitouch.supportsTouchEvents;
```

Defina el tipo de entrada

El motor de ejecución de Flash debe saber qué tipo de entrada táctil interpretar, ya que un evento táctil puede tener muchos elementos o fases. Si toco la pantalla táctil de un dispositivo simplemente con un dedo, ¿el motor de ejecución distribuye un evento táctil? ¿O espera a un gesto? ¿O el motor de ejecución sigue el evento táctil como evento de ratón pulsado? Una aplicación que admita entrada táctil debe establecer el tipo de eventos táctiles que se gestionan para el motor de ejecución de Flash. Utilice la propiedad `Multitouch.inputMode` para establecer el tipo de entrada táctil para el motor de ejecución. El modo de entrada puede ser uno de estos tres:

Ninguno No se proporciona gestión especial para eventos táctiles. Establezca: `Multitouch.inputMode=MultitouchInputMode.NONE` y utilice la clase MouseEvent para gestionar la entrada.

Puntos táctiles únicos Toda la entrada táctil se interpreta por separado y todos los puntos táctiles se supervisan y se controlan. Establezca: `Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT` y utilice la clase TouchEvent para gestionar la entrada.

Entrada de gesto El dispositivo o sistema operativo interpreta la entrada como una combinación compleja de movimientos de dedos en la pantalla. El dispositivo o sistema operativo asigna colectivamente el movimiento a un solo evento de entrada de gesto. Establezca: `Multitouch.inputMode=MultitouchInputMode.GESTURE` y utilice las clases TransformGestureEvent, PressAndTapGestureEvent o GestureEvent para gestionar la entrada.

Consulte "Gestión de eventos táctiles" en la página 610 para ver un ejemplo que utiliza la propiedad `Multitouch.inputMode` para establecer el tipo de entrada antes de controlar un evento táctil.

Comprobación de la compatibilidad con entrada táctil

Otras propiedades de la clase Multitouch proporcionan valores para perfeccionar la aplicación en el entorno actual compatible con entrada táctil. El motor de ejecución de Flash facilita valores para el número de puntos táctiles simultáneos permitidos o para los gestos disponibles. Si el motor de ejecución está en un entorno que no admite control de eventos táctiles necesarios para la aplicación, debe ofrecer una alternativa al usuario. Por ejemplo, gestión de eventos de ratón o información sobre las funciones disponibles en el entorno actual.

También puede utilizar la API para compatibilidad de teclado, entrada táctil y ratón: consulte "Detección de tipos de entrada" en la página 582.

Para obtener más información sobre la comprobación de compatibilidad, consulte "Solución de problemas" en la página 618.

Gestión de eventos táctiles

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Los eventos táctiles básicos se gestionan del mismo modo que otros eventos, como eventos de ratón en ActionScript. Puede detectar una serie de eventos táctiles definidos por las constantes de tipo event en la clase TouchEvent.

Nota: *para la entrada de puntos multitáctiles (como tocar el dispositivo con más de un dedo), el primer punto de contacto distribuye un evento mouse y un evento touch.*

Para gestionar un evento táctil básico:

1. Defina la aplicación para que gestione eventos táctiles estableciendo la propiedad `flash.ui.Multitouch.inputMode` como `MultitouchInputMode.TOUCH_POINT`.
2. Asocie un detector de eventos a la instancia de una clase que herede propiedades de la clase InteractiveObject, por ejemplo, Sprite o TextField.
3. Especifique el tipo de evento táctil para gestionar.
4. Llame a una función de controlador de eventos para hacer algo como respuesta al evento.

Por ejemplo, el siguiente código muestra un mensaje cuando el cuadrado dibujado en mySprite se toca en un dispositivo táctil:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

Propiedades de eventos táctiles

Cuando se produce un evento, se crea un objeto de evento. El objeto TouchEvent contiene información sobre la ubicación y las condiciones del evento táctil. Puede utilizar las propiedades del objeto de evento para recuperar dicha información.

Por ejemplo, el siguiente código crea un objeto TouchEvent *evt* y visualiza la propiedad `stageX` del objeto de evento (la coordenada x del punto en el espacio del escenario en que se produjo la entrada táctil) en el campo de texto:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
myTextField.text = evt.stageX.toString;
myTextField.y = 50;
addChild(myTextField);
}
```

Consulte la clase TouchEvent para saber qué propiedades hay disponibles en el objeto de evento.

Nota: *no todas las propiedades TouchEvent se admiten en todos los entornos de motor de ejecución. Por ejemplo, no todos los dispositivos táctiles pueden detectar la cantidad de presión que el usuario aplica a la pantalla táctil. Así, la propiedad* `TouchEvent.pressure` *no se admite en dichos dispositivos. Intente probar una propiedad concreta para garantizar que la aplicación funciona y consulte* *"Solución de problemas" en la página 618* *para obtener más información.*

Fases de evento táctil

Realice el seguimiento en varias fases de un objeto InteractiveObject del mismo modo que haría con eventos de ratón. Y realice el seguimiento de eventos táctiles desde el principio, mitad y final de la interacción táctil. La clase TouchEvent proporciona valores para la gestión de los eventos `touchBegin`, `touchMove` y `touchEnd`.

Por ejemplo, podría utilizar eventos `touchBegin`, `touchMove` y `touchEnd` para dar al usuario información visual a medida que toca y mueve un objeto de visualización:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
myTextField.width = 200;
myTextField.height = 20;
addChild(myTextField);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
function onTouchBegin(event:TouchEvent) {
    myTextField.text = "touch begin" + event.touchPointID;
}
function onTouchMove(event:TouchEvent) {
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    myTextField.text = "touch end" + event.touchPointID;
}
```

Nota: *el detector táctil inicial se asocia a mySprite, pero los detectores de movimiento y finalización del evento táctil no se asocian. Si el dedo del usuario o el dispositivo señalador se mueve por delante del objeto de visualización, el escenario sigue detectando el evento táctil.*

ID de punto táctil

La propiedad `TouchEvent.touchPointID` es esencial a la hora de escribir aplicaciones que respondan a entrada táctil. El motor de ejecución de Flash asigna a cada punto táctil un único valor `touchPointID`. Cada vez que una aplicación responde a las fases o movimiento de una entrada táctil, compruebe `touchPointID` antes de gestionar el evento. Los métodos de arrastre de entrada táctil de la clase Sprite utilizan la propiedad `touchPointID` como parámetro para que se gestione la instancia de entrada correcta. La propiedad `touchPointID` garantiza que un controlador de eventos responde a la entrada táctil correcta. En caso contrario, el controlador de eventos responde a cualquier instancia de tipo táctil (como todos los eventos `touchMove`) en el dispositivo y produce un comportamiento impredecible. La propiedad es especialmente importante si el usuario arrastra objetos.

Utilice la propiedad `touchPointID` para gestionar una secuencia táctil completa. Una secuencia táctil tiene un evento `touchBegin`, cero o más eventos `touchMove` y un evento `touchEnd`, todos con el mismo valor de `touchPointID`.

El siguiente ejemplo establece una variable *touchMoveID* para probar el valor correcto de `touchPointID` antes de responder a un evento de movimiento. En caso contrario, otra entrada táctil activará el controlador de eventos también. Tenga en cuenta que los detectores de las fases de movimiento y final se encuentran en el escenario, no en el objeto de visualización. El escenario detecta fases de movimiento o final en caso de que la acción táctil del usuario vaya más allá de los límites del objeto de visualización.

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
addChild(myTextField);
myTextField.width = 200;
myTextField.height = 20;
var touchMoveID:int = 0;

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
function onTouchBegin(event:TouchEvent) {
    if(touchMoveID != 0) {
        myTextField.text = "already moving. ignoring new touch";
        return;
    }
    touchMoveID = event.touchPointID;

    myTextField.text = "touch begin" + event.touchPointID;
    stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
}
function onTouchMove(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch";
        return;
    }
    mySprite.x = event.stageX;
    mySprite.y = event.stageY;
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch end";
        return;
    }
    touchMoveID = 0;
    stage.removeEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.removeEventListener(TouchEvent.TOUCH_END, onTouchEnd);
    myTextField.text = "touch end" + event.touchPointID;
}
```

Tocar y arrastrar

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Se han añadido dos métodos a la clase Sprite para aportar compatibilidad adicional para aplicaciones táctiles con entrada multitáctil: `Sprite.startTouchDrag()` y `Sprite.stopTouchDrag()`. Estos métodos se comportan del mismo modo que `Sprite.startDrag()` y `Sprite.stopDrag()` en eventos de ratón. Sin embargo, tenga en cuenta que tanto el método `Sprite.startTouchDrag()` como `Sprite.stopTouchDrag()` toman valores de `touchPointID` como parámetros.

El motor de ejecución asigna el valor de `touchPointID` al objeto de evento para un evento táctil. Utilice este valor para responder a un punto táctil concreto en caso de que el entorno admita varios puntos táctiles simultáneos (incluso si no admite gestos). Para obtener más información sobre la propiedad `touchPointID` consulte "ID de punto táctil" en la página 612.

El siguiente código muestra un sencillo controlador de eventos de inicio y detención de arrastre para un evento táctil. La variable *bg* es un objeto de visualización que contiene *mySprite*:

```
mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(e:TouchEvent) {
    e.target.startTouchDrag(e.touchPointID, false, bg.getRect(this));
    trace("touch begin");
}

function onTouchEnd(e:TouchEvent) {
    e.target.stopTouchDrag(e.touchPointID);
    trace("touch end");
}
```

A continuación se muestra un ejemplo más avanzado que combina el arrastre con fases de eventos táctiles:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(evt:TouchEvent) {
    evt.target.startTouchDrag(evt.touchPointID);
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}

function onTouchMove(evt:TouchEvent) {
    evt.target.alpha = 0.5;
}

function onTouchEnd(evt:TouchEvent) {
    evt.target.stopTouchDrag(evt.touchPointID);
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
```

Gestión de eventos de gestos

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

Gestione los eventos de gestos del mismo modo que lo haría con eventos táctiles básicos. Puede detectar una serie de eventos de gestos definidos por las constantes de tipo de evento en las clases TransformGestureEvent, GestureEvent y PressAndTapGestureEvent.

Para gestionar un evento táctil de gesto:

1. Defina la aplicación para que gestione entrada de gestos estableciendo la propiedad `flash.ui.Multitouch.inputMode` como `MultitouchInputMode.GESTURE`.
2. Asocie un detector de eventos a la instancia de una clase que herede propiedades de la clase InteractiveObject, por ejemplo, Sprite o TextField.
3. Especifique el tipo de evento de gesto que va a gestionar.
4. Llame a una función de controlador de eventos para hacer algo como respuesta al evento.

Por ejemplo, el siguiente código muestra un mensaje cuando el cuadrado dibujado en *mySprite* se toca en un dispositivo táctil:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, swipehandler);

function swipehandler(evt:TransformGestureEvent): void {
myTextField.text = "I've been swiped";
myTextField.y = 50;
addChild(myTextField);
}
```

Los eventos de toque con dos dedos se gestionan del mismo modo, pero con la clase GestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, taphandler);

function taphandler(evt:GestureEvent): void {
myTextField.text = "I've been two-finger tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

Los eventos de pulsar y tocar también se gestionan del mismo modo, pero con la clase PressAndTapGestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP, taphandler);

function taphandler(evt:PressAndTapGestureEvent): void {
myTextField.text = "I've been press-and-tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

Nota: *no todos los eventos GestureEvent, TransformGestureEvent y PressAndTapGestureEvent se admiten en todos los entornos de motor de ejecución. Por ejemplo, no todos los dispositivos táctiles pueden detectar toques con varios dedos. Por ello, los eventos* `gestureSwipe` *de InteractiveObject no se admiten en dichos dispositivos. Intente probar un evento concreto para garantizar que la aplicación funciona y consulte* *"Solución de problemas" en la página 618* *para obtener más información.*

Propiedades de eventos de gestos

Los eventos de gestos tienen un ámbito más reducido de propiedades de eventos que los eventos táctiles básicos. Puede acceder del mismo modo, desde el objeto de evento en la función del controlador de eventos.

Por ejemplo, el siguiente código gira *mySprite* cuando el usuario lleva a cabo un gesto de giro. El campo de texto muestra la cantidad de giro desde el último gesto (cuando pruebe este código, gírelo varias veces para ver el cambio en los valores):

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var mySpriteCon:Sprite = new Sprite();
var myTextField:TextField = new TextField();
myTextField.y = 50;
addChild(myTextField);

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(-20,-20,40,40);
mySpriteCon.addChild(mySprite);
mySprite.x = 20;
mySprite.y = 20;
addChild(mySpriteCon);

mySprite.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rothandler);

function rothandler(evt:TransformGestureEvent): void {
evt.target.parent.rotationZ += evt.target.rotation;
myTextField.text = evt.target.parent.rotation.toString();
}
```

Nota: *no todas las propiedades TransformGestureEvent se admiten en todos los entornos de motor de ejecución. Por ejemplo, no todos los dispositivos táctiles pueden detectar el giro de un gesto en la pantalla. Por ello, la propiedad* `TransformGestureEvent.rotation` *no se admite en estos dispositivos. Intente probar una propiedad concreta para garantizar que la aplicación funciona y consulte* *"Solución de problemas" en la página 618* *para obtener más información.*

Fases de gestos

Además, los eventos de gestos se pueden supervisar por fases y sus propiedades se analizan a medida que se produce el gesto. Por ejemplo, puede supervisar coordenadas x a medida que un objeto se mueve con un gesto de barrido. Utilice estos valores para dibujar una línea en todos los puntos del trazado una vez finalizado el barrido. O cambie visualmente un objeto de visualización a medida que se arrastra por la pantalla con un gesto de desplazamiento. Cambie el objeto de nuevo cuando haya terminado el gesto de desplazamiento.

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {

    evt.target.localX++;

if (evt.phase==GesturePhase.BEGIN) {
    myTextField.text = "Begin";
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}
if (evt.phase==GesturePhase.UPDATE) {
    myTextField.text = "Update";
    evt.target.alpha = 0.5;
}
if (evt.phase==GesturePhase.END) {
    myTextField.text = "End";
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
}
```

Nota: *la frecuencia de la fase de actualización depende del entorno del motor de ejecución. Algunas combinaciones de sistema operativo y hardware no permiten actualización alguna.*

La fase de gestos lo es "todo" para los eventos de gestos sencillos

Algunos objetos de eventos de gestos no siguen fases individuales del evento de gesto, sino que llenan la propiedad phase del objeto evento con el valor all. Los gestos sencillos de barrido y toque con dos dedos no siguen las distintas fases del evento. La propiedad `phase` del objeto de evento para un objeto InteractiveObject que detecta eventos `gestureSwipe` o `gestureTwoFingerTap` siempre es `all` una vez distribuido el evento:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, onSwipe);
mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, onTwoTap);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onSwipe(swipeEvt:TransformGestureEvent):void {
    myTextField.text = swipeEvt.phase // Output is "all"
}
function onTwoTap(tapEvt:GestureEvent):void {
    myTextField.text = tapEvt.phase // Output is "all"
}
```

Solución de problemas

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

La compatibilidad de hardware y software con entradas táctiles cambia a pasos agigantados. Esta referencia no contiene una lista de todas las combinaciones de dispositivos y sistemas operativos que admiten tecnología multitáctil. Ofrece, no obstante, información sobre el uso de la API de detección para determinar si su aplicación se ha desarrollado en un dispositivo compatible con tecnología multitáctil y proporciona sugerencias para solucionar posibles problemas en el código ActionScript.

Los motores de ejecución de Flash responden a eventos táctiles según la información que el dispositivo, sistema operativo o software (por ejemplo, un navegador) transfiere al motor de ejecución. Esta dependencia del entorno de software complica la documentación de la compatibilidad con la tecnología multitáctil. Algunos dispositivos interpretan un gesto o movimiento táctil de modo distinto. ¿El giro se define con dos dedos girando al mismo tiempo? ¿El giro se define con un solo dedo dibujando un círculo en la pantalla? En función del entorno de hardware y software, el gesto de giro podría ser incluso un gesto totalmente distinto. Así, es el dispositivo quien informa al sistema operativo de la entrada del usuario. Posteriormente, el sistema operativo transfiere esta información al motor de ejecución. Si el motor de ejecución está en un navegador, el software del navegador a veces interpreta el gesto o evento táctil y no necesita transferir la entrada al motor de ejecución. Este comportamiento es similar al de las “teclas de función”: puede insertar una combinación de teclas específica para Flash Player haga algo dentro del navegador y este mantiene un menú abierto.

Las API individuales y las clases mencionan si no son compatibles con determinados sistemas operativos. Puede analizar las distintas entradas de las API aquí, comenzando por la clase Multitouch: http://help.adobe.com/es_ES/FlashPlatform/reference/actionscript/3/flash/ui/Multitouch.html.

A continuación encontrará descripciones sobre gestos y toques habituales:

Desplazamiento Mueva un dedo de izquierda a derecha o de derecha a izquierda. Algunos dispositivos requieren el uso de dos dedos para este gesto.

Girar Toque con dos dedos y muévalos en círculo (como si trazase un círculo imaginario en una superficie con dos dedos). El punto de giro se establece en el punto medio entre los dos puntos de contacto de los dedos.

Barrido Mueva rápidamente tres dedos de izquierda a derecha, de derecha a izquierda, de arriba abajo o de abajo arriba.

Zoom Toque con dos dedos y muévalos separándolos entre sí para ampliar y acercándolos para reducir el zoom.

Pulsar y tocar Mueva o pulse con un dedo y toque la superficie con otro.

Cada dispositivo tiene su propia documentación para explicar los gestos admitidos y cómo realizarlos en el dispositivo. En general, el usuario debe levantar todos los dedos del dispositivo entre los distintos gestos, aunque esto depende del sistema operativo.

Si su aplicación no responde a eventos o gestos táctiles, pruebe lo siguiente:

1. ¿Dispone de detectores de eventos para eventos táctiles o de gestos asociados a una clase de objeto que herede de la clase InteractiveObject? Solo las instancias de InteractiveObject pueden detectar eventos táctiles y de gestos
2. ¿Está probando la aplicación en Flash Professional CS5? Si es así, intente publicar y probar la aplicación, ya que Flash Professional puede interceptar la interacción.
3. Empiece con gestos sencillos y observe qué sucede (el siguiente ejemplo de código está tomado de la entrada API de `Multitouch.inputMode`):

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField()

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taplistener);

function taplistener(e:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

   Toque el rectángulo. Si este ejemplo funciona, sabrá que su entorno admite un toque simple. Puede continuar con gestiones más complicadas.

   Probar la compatibilidad de gestos es más complicado. Un dispositivo o sistema operativo individual puede admitir todas las combinaciones de gestos o ninguna.

   A continuación incluimos una sencilla prueba del gesto de zoom:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_ZOOM , onZoom);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a zoom gesture";
addChild(myTextField);

function onZoom(evt:TransformGestureEvent):void {
    myTextField.text = "Zoom is supported";
}
```

Realice un gesto de zoom en el dispositivo y vea si en el campo de texto aparece el mensaje `Zoom is supported` (Se admite zoom). El detector de eventos se añade al escenario para que pueda realizar el gesto en cualquier parte de la aplicación de prueba.

A continuación incluimos una sencilla prueba del gesto de barrido:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a pan gesture";
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {
    myTextField.text = "Pan is supported";
}
```

Realice un gesto de barrido en el dispositivo y vea si en el campo de texto aparece el mensaje `Pan is supported` (Se admite barrido). El detector de eventos se añade al escenario para que pueda realizar el gesto en cualquier parte de la aplicación de prueba.

Ciertas combinaciones de sistemas operativos y dispositivos pueden admitir ambos gestos, otras admitir solo uno y otras ninguno. Pruebe su entorno de desarrllo de aplicación para asegurarse.

Problemas conocidos

A continuación se detallan problemas conocidos con la entrada táctil:

1. Mobile Internet Explorer en el sistema operativo Windows Mobile aplica el zoom automáticamente al contenido de archivos SWF:

   Este comportamiento de zoom de Internet Explorer se puede anular añadiendo lo siguiente a la página HTML que contiene el archivo SWF:

   ```
   <head>
   <meta name="viewport" content="width=device-width, height=device-height, initial-
   scale=1.0">
   </head>
   ```

2. En Windows 7 (y posiblemente otros sistemas operativos), el usuario debe levantar el dispositivo señalador (o el dedo) de la pantalla entre los distintos gestos. Por ejemplo, para girar y ampliar una imagen:
   - Primero se realiza el gesto de giro.
   - Luego se levantan los dedos de la pantalla.
   - Y luego se han de volver a colocar los dedos sobre la pantalla para realizar el gesto de zoom.
3. Windows 7 (y posiblemente en otros sistemas operativos), los gestos de giro y zoom no siempre generan una fase “update” si el usuario lleva a cabo el gesto demasiado deprisa.
4. Windows 7 Starter Edition no admite la tecnología multitáctil. Consulte el foro de AIR Labs para saber más: http://forums.adobe.com/thread/579180?tstart=0
5. En Mac OS 10.5.3 y versiones posteriores, el valor `Multitouch.supportsGestureEvents` siempre es `true`, incluso si el hardware no admite eventos de gestos.

Capítulo 33: Copiar y pegar

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Utilice las clases de la API del portapapeles para copiar información al portapapeles del sistema y pegarla en otro lado. Entre los formatos de datos que se pueden transferir entre una aplicación que se ejecuta en Adobe® Flash® Player o Adobe® AIR® se incluyen:

- Texto
- Texto en formato HTML
- Datos en formato de texto enriquecido
- Objetos serializados
- Referencias a objetos (solo válido en la aplicación original)
- Mapas de bits (solo AIR)
- Archivos (solo AIR)
- Cadenas URL (solo AIR)

Aspectos básicos de la operación de copiar y pegar

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

La API de copiar y pegar contiene las clases siguientes.

Paquete	Clases
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

La propiedad estática `Clipboard.generalClipboard` representa el portapapeles del sistema operativo. La clase Clipboard ofrece métodos para leer y escribir datos en objetos clipboard.

La clases HTMLLoader (en AIR) y TextField implementan el comportamiento predeterminado para los métodos abreviados de teclado normales de copiar y pegar. Para implementar el comportamiento de métodos abreviados de teclado para copiar y pegar componentes personalizados, se pueden detectar directamente estas pulsaciones de teclas. También se pueden utilizar comandos de menú nativos junto con equivalentes a las teclas para responder a las pulsaciones de teclas de forma indirecta.

Un mismo objeto Clipboard puede poner a disposición distintas representaciones de la misma información para aumentar la probabilidad de que otra aplicación pueda comprender y utilizar los datos. Por ejemplo, una imagen puede incluirse como datos de imagen, objeto Bitmap serializado o archivo. La representación de los datos en un formato determinado puede diferirse de modo que el formato no se cree hasta el momento de leerse los datos en ese formato.

Lectura y escritura en el portapapeles del sistema

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Para leer el portapapeles del sistema operativo, llame al método `getData()` del objeto `Clipboard.generalClipbooard` y transfiera el nombre del formato que se quiere leer:

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

Nota: *el contenido que se ejecuta en Flash Player o en un entorno limitado que no pertenezca a la aplicación en AIR solo puede llamar al método `getData()` en un controlador de eventos para un evento `paste`. Es decir, únicamente el código que se ejecuta en el entorno limitado de la aplicación de AIR puede llamar al método `getData()` fuera de un controlador de eventos `paste`.*

Para escribir en el portapapeles, añada los datos al objeto `Clipboard.generalClipboard` en uno o varios formatos. Los datos que ya existan en el mismo formato se sobrescriben automáticamente. No obstante, conviene siempre vaciar el portapapeles del sistema antes de escribirle nuevos datos para asegurarse de que también se supriman los datos no relacionados que pueda haber en algún otro formato.

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

Nota: *el contenido que se ejecuta en Flash Player o en un entorno limitado que no pertenezca a la aplicación en AIR solo puede llamar al método `setData()` en un controlador de eventos de usuario, por ejemplo, eventos de teclado o de ratón, o de eventos `copy` o `cut`. Es decir, únicamente el código que se ejecuta en el entorno limitado de la aplicación de AIR puede llamar al método `setData()` desde fuera de un controlador de eventos de usuario.*

Operaciones de pegar y copiar HTML en AIR

Adobe AIR 1.0 y posterior

El entorno HTML en Adobe AIR proporciona su propia serie de eventos y comportamiento predeterminado para copiar y pegar. solo el código que se ejecute en el entorno limitado de la aplicación tiene acceso directo al portapapeles del sistema a través del objeto `Clipboard.generalClipboard` de AIR. El código JavaScript en un entorno limitado ajeno a la aplicación tiene acceso al portapapeles a través del objeto de evento distribuido como respuesta a uno de los eventos de copiar o pegar distribuidos por un elemento de un documento HTML.

Entre los eventos de copiar y pegar se encuentran: `copy`, `cut` y `paste`. El objeto distribuido para estos eventos proporciona acceso al portapapeles a través de la propiedad `clipboardData`.

Comportamiento predeterminado

Adobe AIR 1.0 y posterior

De forma predeterminada, AIR copia elementos seleccionados como respuesta al comando de copiar, que se puede generar mediante un método abreviado de teclado o un menú contextual. En las regiones editables AIR corta texto como respuesta al comando de cortar o pega texto en el lugar seleccionado como respuesta al comando de pegar.

Para impedir que se produzca el comportamiento predeterminado, el controlador de eventos puede llamar al método `preventDefault()` del objeto de evento distribuido.

Uso de la propiedad clipboardData del objeto de evento

Adobe AIR 1.0 y posterior

La propiedad `clipboardData` del objeto de evento distribuido como resultado de uno de los eventos de copiar o pegar permite leer y escribir datos en el portapapeles.

Para escribir en el portapapeles al controlar un evento de copiar o cortar, utilice el método `setData()` del objeto `clipboardData`, pasando los datos a copiar y el tipo MIME:

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

Para tener acceso a los datos que se pegan se puede utilizar el método `getData()` del objeto `clipboardData`, pasando el tipo MIME del formato de los datos. Los formatos disponibles los notifica la propiedad `types`.

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

Solo se tiene acceso al método `getData()` y a la propiedad `types` en el objeto de evento distribuido por el evento `paste`.

El siguiente ejemplo ilustra cómo suprimir el comportamiento predeterminado de copiar y pegar en una página HTML. El controlador de eventos `copy` pone el texto copiado en cursiva y lo copia en el portapapeles como texto en formato HTML. El controlador de eventos `cut` copia los datos seleccionados en el portapapeles y los borra del documento. El controlador de eventos `paste` inserta el contenido del portapapeles en formato HTML y aplica el estilo negrita al texto insertado.

```
<html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

Formatos de datos para Clipboard

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Los formatos para Clipboard describen los datos que se colocan en un objeto Clipboard. Flash Player o AIR convierten automáticamente los formatos de datos estándar entre tipos de datos de ActionScript y formatos del portapapeles del sistema. Además, los objetos de la aplicación se pueden transferir entre aplicaciones basadas en ActionScript mediante formatos definidos por la aplicación.

Un objeto Clipboard puede contener representaciones de la misma información en distintos formatos. Por ejemplo, un objeto Clipboard que representa a un elemento Sprite puede incluir un formato de referencia para utilizarse dentro de la misma aplicación, un formato serializado para utilizarlo con otra aplicación que se ejecute en Flash Player o AIR, un formato de mapa de bits para utilizarse en un editor de imágenes y un formato de lista de archivos (posiblemente con representación aplazada para codificar un archivo PNG) para copiar o arrastrar una representación del elemento Sprite al sistema de archivos.

Formatos de datos estándar

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Las constantes que definen los nombres de formatos estándar se proporcionan en la clase ClipboardFormats:

Constante	Descripción
TEXT_FORMAT	Los datos en formato de texto se traducen a y de la clase String de ActionScript.
HTML_FORMAT	Texto con marcado HTML.
RICH_TEXT_FORMAT	Los datos en formato de texto enriquecido se traducen a y de la clase ByteArray de ActionScript. El marcado RTF no se interpreta ni traduce de forma alguna.
BITMAP_FORMAT	(Solo AIR) Los datos con formato de mapa de bits se convierten con la clase BitmapData de ActionScript.
FILE_LIST_FORMAT	(Solo AIR) Los datos con formato de lista de archivos se convierten en un conjunto de objetos File de ActionScript.
URL_FORMAT	(Solo AIR) Los datos con formato URL se convierten con la clase String de ActionScript.

Al copiar y pegar datos como respuesta a un evento `copy`, `cut`, o `paste` en contenido HTML alojado en una aplicación de AIR, es necesario usar tipos MIME en lugar de cadenas ClipboardFormat. Los tipos de datos MIME válidos son:

Tipo MIME	Descripción
Texto	"text/plain"
URL	"text/uri-list"
Mapa de bits	"image/x-vnd.adobe.air.bitmap"
Lista de archivos	"application/x-vnd.adobe.air.file-list"

Nota: *los datos en formato de texto enriquecido no están disponibles en la propiedad* `clipboardData` *del objeto de evento distribuido como resultado de un evento* `paste` *en contenido HTML.*

Formatos de datos personalizados

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Se pueden utilizar formatos personalizados definidos por la aplicación para transferir objetos como referencias o como copias serializadas. Las referencias solo son válidas en la misma aplicación. Los objetos serializados se pueden transferir entre aplicaciones, pero solo pueden utilizarse con objetos que permanecen válidos al serializarse y deserializarse. Los objetos suelen poder serializarse si sus propiedades son tipos sencillos u objetos serializables.

Para añadir un objeto serializado a un objeto Clipboard, establezca el parámetro *serializable* en `true` al llamar al método `Clipboard.setData()`. El nombre del formato puede ser el de uno de los formatos estándar o una cadena arbitraria definida por la aplicación.

Modos de transferencia

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Cuando un objeto se escribe en el portapapeles con un formato de datos personalizado, los datos del objeto se pueden leer en el portapapeles, bien como una referencia o como una copia serializada del objeto original. Existen cuatro modos de transferencia que determinan si los objetos se transfieren como referencias o como copias serializadas:

Modo de transferencia	Descripción
ClipboardTransferModes.ORIGINAL_ONLY	Solo se devuelve una referencia. Si no se dispone de ninguna referencia, se devuelve un valor nulo.
ClipboardTransferModes.ORIGINAL_PREFFERED	Se devuelve una referencia, si hay una disponible. En caso contrario, se devuelve una copia serializada.
ClipboardTransferModes.CLONE_ONLY	Solo se devuelve una copia serializada. Si no hay ninguna disponible, se devuelve un valor null.
ClipboardTransferModes.CLONE_PREFFERED	Se devuelve una copia serializada, si hay una disponible. En caso contrario, se devuelve una referencia.

Lectura y escritura de formatos de datos personalizados

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Al escribir un objeto en el portapapeles, puede utilizar cualquier cadena que no comience con los prefijos reservados `air:` o `flash:` para el parámetro *format*. Utilice la misma cadena que el formato para leer el objeto. Los ejemplos siguientes ilustran cómo se leen y escriben objetos en el portapapeles:

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Para extraer un objeto serializado del objeto Clipboard (tras una operación de soltar y pegar), utilice el mismo nombre de formato y los modos de transferencia `CLONE_ONLY` o `CLONE_PREFFERED`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

Siempre se añade una referencia al objeto Clipboard. Para extraer la referencia del objeto portapapeles (tras una operación de colocar o pegar), en lugar de la copia serializada utilice los modos de transferencia `ORIGINAL_ONLY` o `ORIGINAL_PREFFERED`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Las referencias solo son válidas si el objeto Clipboard viene de la aplicación de actual. Utilice el modo de transferencia `ORIGINAL_PREFFERED` para tener acceso a la referencia cuando está disponible, y la copia serializada cuando la referencia no está disponible.

Representación aplazada

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

Si la creación de un formato de datos requiere mucho procesamiento, puede utilizar la representación aplazada si suministra una función que proporcione los datos a petición. Solo se llama a la función si el receptor de la operación de colocar o pegar solicita datos en el formato diferido.

La función de representación se añade a un objeto Clipboard utilizando el método `setDataHandler()`. La función debe devolver los datos en el formato adecuado. Por ejemplo: si llamó a `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)`, la función `writeText()` debe devolver una cadena.

Si se añade un formato de datos del mismo tipo a un objeto Clipboard con el método `setData()`, esos datos tendrán prioridad sobre la versión aplazada (no se llama nunca a la función de representación). La función de representación puede o no volver a llamarse si se solicita acceso una segunda vez a los mismos datos del portapapeles.

Nota: *en Mac OS X, la representación aplazada funciona solo con formatos de datos personalizados. En los formatos de datos estándar, se llama a la función de representación inmediatamente.*

Pegar texto con una función de representación aplazada

Flash Player 10 y posterior, Adobe AIR 1.0 y posterior

El ejemplo siguiente ilustra cómo se implementa una función de representación aplazada.

Cuando el usuario pulsa el botón Copiar, la aplicación borra el portapapeles del sistema para garantizar que no queda contenido de operaciones anteriores con el portapapeles. El método `setDataHandler()` establece entonces la función `renderData()` como procesador del portapapeles.

Cuando el usuario selecciona el comando Pegar en el menú contextual del campo de texto de destino, la aplicación accede al portapapeles y establece el texto de destino. Dado que el formato de los datos de texto del portapapeles se ha establecido con una función y no con una cadena, el portapapeles llama a la función `renderData()`. La función `renderData()` devuelve el texto que estaba en el texto de origen, que se asigna entonces al texto de destino.

Tenga en cuenta que si se modifica el texto de origen antes de pulsar el botón Pegar, la modificación se reflejará en el texto pegado, incluso cuando se produce después de haber pulsado el botón de copiar. Esto se debe a que la función de representación no copia el texto de origen sino hasta pulsar el botón de pegar. (Al utilizar la representación aplazada en una aplicación real, puede ser conveniente guardar o proteger de alguna manera los datos de origen para evitar que suceda este problema).

Ejemplo de Flash

```
package {
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;
    import flash.desktop.ClipboardTransferMode;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;
    import flash.events.MouseEvent;
    import flash.events.Event;
    public class DeferredRenderingExample extends Sprite
    {
        private var sourceTextField:TextField;
        private var destination:TextField;
        private var copyText:TextField;
        public function DeferredRenderingExample():void
        {
            sourceTextField = createTextField(10, 10, 380, 90);
            sourceTextField.text = "Neque porro quisquam est qui dolorem "
                + "ipsum quia dolor sit amet, consectetur, adipisci velit.";
```

```
            copyText = createTextField(10, 110, 35, 20);
            copyText.htmlText = "<a href='#'>Copy</a>";
            copyText.addEventListener(MouseEvent.CLICK, onCopy);

            destination = createTextField(10, 145, 380, 90);
            destination.addEventListener(Event.PASTE, onPaste);
        }
        private function createTextField(x:Number, y:Number, width:Number,
                        height:Number):TextField
        {
            var newTxt:TextField = new TextField();
            newTxt.x = x;
            newTxt.y = y;
            newTxt.height = height;
            newTxt.width = width;
            newTxt.border = true;
            newTxt.multiline = true;
            newTxt.wordWrap = true;
            newTxt.type = TextFieldType.INPUT;
            addChild(newTxt);
            return newTxt;
        }
        public function onCopy(event:MouseEvent):void
        {
            Clipboard.generalClipboard.clear();
            Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                            renderData);
        }
        public function onPaste(event:Event):void
        {
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                    sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Ejemplo de Flex

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326"
height="330" applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text =
Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur,
adipisci velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

Capítulo 34: Entrada de acelerómetro

Flash Player 10.1 y posterior, Adobe AIR 2 y posterior

La clase Accelerometer distribuye eventos basados en la actividad detectada por el sensor de movimiento del dispositivo. Estos datos representan la ubicación del dispositivo o el movimiento a lo largo de un eje tridimensional. Cuando el dispositivo se mueve, el sensor detecta este movimiento y devuelve las coordenadas de aceleración del dispositivo. La clase Accelerometer proporciona métodos para consultar si se admite o no acelerómetro y también para definir los eventos de aceleración que se distribuyen.

Los ejes del acelerómetro se normalizan en la orientación de la pantalla, no en la orientación física del dispositivo. Cuando el dispositivo reorienta la pantalla, los ejes del acelerómetro también se reorientan. De este modo, el eje y siempre es casi vertical cuando el usuario sostiene el teléfono en una posición de visualización normal en orientación vertical; no importa en qué dirección se gire el teléfono. Si la orientación automática está desactivada como, por ejemplo, cuando el contenido SWF en un navegador está en modo de pantalla completa, los ejes del acelerómetro no se reorientan conforme se gira el dispositivo.

Más temas de ayuda

flash.sensors.Accelerometer

flash.events.AccelerometerEvent

Comprobación de la compatibilidad del acelerómetro

Utilice la propiedad `Accelerometer.isSupported` para probar si el entorno del motor de ejecución tiene capacidad para usar esta función:

```
if (Accelerometer.isSupported)
{
    // Set up Accelerometer event listeners and code.
}
```

La clase Accelerometer y sus miembros son accesibles para las versiones del motor de ejecución incluidas en cada entrada de la API. Sin embargo, el entorno actual en tiempo de ejecución determina la disponibilidad de esta función. Por ejemplo, puede compilar código usando las propiedades de la clase Accelerometer para Flash Player 10.1, pero deberá usar la propiedad `Accelerometer.isSupported` para comprobar la disponibilidad de la función Accelerometer en el dispositivo del usuario. Si `Accelerometer.isSupported` es `true` en tiempo de ejecución, se admite la función Accelerometer.

Detección de cambios del acelerómetro

Para utilizar el sensor del acelerómetro, cree una instancia del objeto Accelerometer y registre los eventos `update` que distribuya. El evento `update` es un objeto de evento Accelerometer. Este evento tiene cuatro propiedades y todas son números:

- `accelerationX`: aceleración en el eje x, medida en g. El eje x se ejecuta de izquierda a derecha en el dispositivo cuando está en la posición vertical. (El dispositivo está en orientación vertical y con la parte superior hacia arriba.) La aceleración es positiva si el dispositivo se mueve hacia la derecha.
- `accelerationY`: aceleración en el eje y, medida en g. El eje y se ejecuta de abajo arriba en el dispositivo cuando está en la posición vertical. (El dispositivo está en orientación vertical y con la parte superior hacia arriba.) La aceleración es positiva si el dispositivo se mueve hacia arriba con relación a este eje.
- `accelerationZ`: aceleración en el eje z, medida en g. El eje z se ejecuta perpendicular a la cara del dispositivo. La aceleración es positiva si mueve el dispositivo de modo que la parte frontal apunte hacia arriba. La aceleración es negativa si la parte frontal del dispositivo apunta hacia el suelo.
- `timestamp`: el número de milisegundos en el momento del evento desde que se inicializó el motor de ejecución.

1 g es la aceleración estándar debida a la gravedad, aproximadamente 9,8 m/s^2.

A continuación incluimos un ejemplo básico que muestra datos del acelerómetro en un campo de texto:

```
var accl:Accelerometer;
if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(evt:AccelerometerEvent):void
{
    accTextField.text = "acceleration X: " + evt.accelerationX.toString() + "\n"
            + "acceleration Y: " + evt.accelerationY.toString() + "\n"
            + "acceleration Z: " + evt.accelerationZ.toString()
}
```

Para usar este ejemplo, debe crear un campo de texto `accTextField` y añadirlo a la lista de visualización antes de utilizar este código.

Puede ajustar el intervalo de tiempo deseado para los eventos de acelerómetro llamando al método `setRequestedUpdateInterval()` del objeto Accelerometer. Este método toma un parámetro, `interval`, que es el intervalo de actualización requerido, en milisegundos:

```
var accl:Accelerometer;
accl = new Accelerometer();
accl.setRequestedUpdateInterval(1000);
```

El tiempo real entre las actualizaciones de acelerómetro puede ser mayor o menor que este valor. Cualquier cambio en el intervalo de actualización afecta a todos los detectores registrados. Si no llama al método `setRequestedUpdateInterval()`, la aplicación recibe actualizaciones basadas en el intervalo predeterminado del dispositivo.

Los datos del acelerómetro tienen cierto grado de imprecisión. Puede utilizar una media aproximada de los datos recientes para mejorar los datos. Por ejemplo, a continuación se analizan diversas lecturas recientes del acelerómetro con la lectura actual para obtener un resultado redondeado:

```
var accl:Accelerometer;
var rollingX:Number = 0;
var rollingY:Number = 0;
var rollingZ:Number = 0;
const FACTOR:Number = 0.25;

if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.setRequestedUpdateInterval(200);
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(event:AccelerometerEvent):void
{
    accelRollingAvg(event);
    accTextField.text = rollingX + "\n" +  rollingY + "\n" + rollingZ + "\n";
}

function accelRollingAvg(event:AccelerometerEvent):void
{
    rollingX = (event.accelerationX * FACTOR) + (rollingX * (1 - FACTOR));
    rollingY = (event.accelerationY * FACTOR) + (rollingY * (1 - FACTOR));
    rollingZ = (event.accelerationZ * FACTOR) + (rollingZ * (1 - FACTOR));
}
```

Sin embargo, esta media aproximada solo es deseable si el intervalo de actualización del acelerómetro es pequeño.

Capítulo 35: Operación de arrastrar y colocar en AIR

Adobe AIR 1.0 y posterior

Utilice las clases de la API de arrastrar y colocar de Adobe® AIR™ para obtener compatibilidad con los gestos de arrastrar y colocar de la interfaz de usuario. Un *gesto* en este sentido significa una acción por parte del usuario, por intermedio del sistema operativo y la aplicación, que expresa la intención de copiar, mover o vincular información. Un gesto de *arrastrar hacia fuera* es cuando el usuario arrastra un objeto hacia fuera para sacarlo de un componente o una aplicación. Un gesto de *arrastrar hacia dentro* es cuando el usuario arrastra un objeto desde fuera para introducirlo en un componente o una aplicación.

Con la API de arrastrar y colocar, un usuario puede arrastrar datos entre aplicaciones y entre los componentes de una misma aplicación. Los formatos de transferencia compatibles incluyen:

- Mapas de bits
- Archivos
- Texto en formato HTML
- Texto
- Datos en formato de texto enriquecido
- URL
- Promesas de archivo
- Objetos serializados
- Referencias de objetos (válidas solamente en la aplicación de origen)

Aspectos básicos de la operación de arrastrar y colocar en AIR

Adobe AIR 1.0 y posterior

Para ver una explicación rápida y ejemplos de código del uso de la operación de arrastrar y colocar en una aplicación de AIR, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Supporting drag-and-drop and copy-and-paste (Compatibilidad con las funciones de arrastrar y colocar, y copiar y pegar, en inglés) (Flex)
- Supporting drag-and-drop and copy-and-paste (Compatibilidad con las funciones de arrastrar y colocar, y copiar y pegar, en inglés) (Flash)

La API de arrastrar y colocar contiene las clases siguientes.

Paquete	Clases
flash.desktop	• NativeDragManager • NativeDragOptions • Clipboard • URLFilePromise • IFilePromise Las constantes que se utilizan con la API de arrastrar y colocar se definen en las clases siguientes: • NativeDragActions • ClipboardFormat • ClipboardTransferModes
flash.events	NativeDragEvent

Etapas de gestos de arrastrar y colocar

El gesto de arrastrar y colocar tiene tres etapas:

Inicio *Un usuario inicia una operación de arrastrar y colocar arrastrando desde un componente, o un elemento de un componente, mientras mantiene pulsado el botón del ratón.* El componente que es el origen del elemento arrastrado suele designarse como iniciador de la operación de arrastrar y distribuye los eventos `nativeDragStart` y `nativeDragComplete`. Una aplicación de Adobe AIR inicia una operación de arrastrar llamando al método `NativeDragManager.doDrag()` como respuesta a un evento `mouseDown` o `mouseMove`.

Si la operación de arrastrar se inicia fuera de una aplicación de AIR, no hay ningún objeto iniciador que distribuya eventos `nativeDragStart` o `nativeDragComplete`.

Arrastrar *El usuario mantiene pulsado el botón del ratón al desplazar el cursor a otro componente, a otra aplicación o al escritorio.* Siempre y cuando esté en curso la operación de arrastrar, el objeto iniciador distribuye eventos `nativeDragUpdate`. (Sin embargo, este evento no se distribuye en AIR para Linux.) Cuando el usuario pasa el ratón sobre un posible destino en una aplicación de AIR, el destino distribuye un evento `nativeDragEnter`. El controlador de eventos puede examinar el objeto de evento para determinar si los datos arrastrados están disponibles en un formato aceptado por el destino y, en caso afirmativo, dejar al usuario colocar los datos en él llamando al método `NativeDragManager.acceptDragDrop()`.

Siempre y cuando el gesto de arrastrar permanezca sobre un objeto interactivo, dicho objeto distribuye eventos `nativeDragOver`. Cuando el gesto de arrastrar abandona el objeto interactivo, distribuye un evento `nativeDragExit`.

Colocar *El usuario suelta el ratón sobre un destino idóneo.* Si el destino es una aplicación de AIR o un componente, el objeto de destino distribuye un evento `nativeDragDrop`. El controlador de eventos tiene acceso a los datos transferidos desde el objeto de evento. Si el destino se encuentra fuera de AIR, el sistema operativo u otra aplicación controla la colocación. En ambos casos, el objeto iniciador distribuye un evento `nativeDragComplete` (si la operación de arrastrar empezó en AIR).

La clase NativeDragManager controla tanto los gestos de arrastrar hacia dentro como los de arrastrar hacia fuera. Todos los miembros de la clase NativeDragManager son estáticos: no cree una instancia de esta clase.

Objeto Clipboard

Los datos que se arrastran dentro o fuera de una aplicación o un componente se contienen en un objeto Clipboard. Un mismo objeto Clipboard puede poner a disposición distintas representaciones de la misma información para aumentar la probabilidad de que otra aplicación pueda comprender y utilizar los datos. Por ejemplo, una imagen puede incluirse como datos de imagen, objeto Bitmap serializado o archivo. La representación de los datos en un formato determinado puede diferirse a una función de representación a la que no se llama hasta el momento de leerse los datos.

Una vez iniciado un gesto de arrastrar, solo se tiene acceso al objeto Clipboard desde un controlador de eventos para los eventos `nativeDragEnter`, `nativeDragOver`, y `nativeDragDrop`. Después de finalizado el gesto de arrastrar, no se puede volver a leer o utilizar el objeto Clipboard.

Un objeto de aplicación puede transferirse como referencia y como objeto serializado. Las referencias solo son válidas en la aplicación de origen. Las transferencias de objetos serializados son válidos entre aplicaciones de AIR, pero solo pueden utilizarse con objetos que permanecen válidos al serializarse y deserializarse. Los objetos que se serializan se convierten al formato de mensaje de acción para ActionScript 3 (AMF3), un formato de transferencia de datos basado en cadenas.

Trabajo con la arquitectura de Flex

En la mayoría de los casos es mejor utilizar la API de arrastrar y colocar de Adobe® Flex™ al crear aplicaciones de Flex. La arquitectura de Flex proporciona un conjunto de funciones equivalentes cuando se ejecuta una aplicación de Flex en AIR (utiliza el método NativeDragManager de AIR de modo interno). Flex mantiene además un conjunto de funciones más limitado si una aplicación o un componente se ejecutan en el entorno más limitado del navegador. Las clases de AIR no pueden utilizarse en componentes o aplicaciones que se ejecuten fuera del entorno del motor de ejecución de AIR.

Compatibilidad con el gesto de arrastrar hacia fuera

Adobe AIR 1.0 y posterior

Para tener compatibilidad con el gesto de arrastrar hacia fuera hay que crear un objeto Clipboard como respuesta a un evento `mouseDown` y enviarlo al método `NativeDragManager.doDrag()`. La aplicación puede entonces detectar el evento `nativeDragComplete` en el objeto iniciador para determinar qué medidas tomar cuando el usuario finalice o abandone el gesto.

Preparación de datos para la transferencia

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para preparar datos o un objeto para arrastrarlos, cree un objeto Clipboard y añada la información a transferirse en uno o varios formatos. Para pasar datos que pueden traducirse automáticamente a formatos nativos del portapapeles se pueden utilizar formatos de datos estándar; para pasar objetos, pueden utilizarse formatos definidos por la aplicación.

Si la conversión en un formato determinado de la información a transferirse requiere mucho procesamiento, se puede indicar el nombre de una función de controlador para que realice la conversión. La función se llama únicamente si el componente o la aplicación de recepción puede leer el formato asociado.

Para obtener más información sobre los formatos de portapapeles, consulte “Formatos de datos para Clipboard” en la página 624.

El siguiente ejemplo muestra cómo se crea un objeto Clipboard que contiene un mapa de bits en diversos formatos: un objeto Bitmap, un formato de mapa de bits nativo y un formato de lista de archivos que contiene el archivo desde el que se cargó originalmente el mapa de bits:

```
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
public function createClipboard(image:Bitmap, sourceFile:File):Clipboard{
    var transfer:Clipboard = new Clipboard();
    transfer.setData("CUSTOM_BITMAP", image, true); //Flash object by value and by reference
    transfer.setData(ClipboardFormats.BITMAP_FORMAT, image.bitmapData, false);
    transfer.setData(ClipboardFormats.FILE_LIST_FORMAT, new Array(sourceFile), false);
    return transfer;
}
```

Inicio de una operación de arrastrar hacia fuera

Adobe AIR 1.0 y posterior

Para iniciar una operación de arrastrar, llame al método `NativeDragManager.doDrag()` como respuesta a un evento mouseDown. El método `doDrag()` es un método estático que admite los parámetros siguientes:

Parámetro	Descripción
initiator	El objeto en el cual origina la acción de arrastrar y que distribuye los eventos `dragStart` y `dragComplete`. El iniciador debe ser un objeto interactivo.
clipboard	El objeto Clipboard que contiene los datos a transferirse. Hay referencias al objeto Clipboard en los objetos NativeDragEvent distirbuidos durante la secuencia de arrastrar y colocar.
dragImage	(Opcional) Un objeto BitmapData para mostrar durante la operación de arrastrar. La imagen puede especificar un valor `alpha`. (Nota: Microsoft Windows aplica siempre un difuminado alfa fijo a las imágenes arrastradas).
offset	(Opcional) Un objeto Point que especifica el desplazamiento de la imagen arrastrada desde la zona interactiva del ratón. Para mover la imagen hacia arriba y hacia la izquierda en relación con el cursor del ratón, utilice coordenadas negativas. Si no se indica ningún desplazamiento, la esquina superior izquierda de la imagen arrastrada se coloca en la zona interactiva del ratón.
actionsAllowed	(opcional) Un objeto NativeDragOptions que especifica qué acciones (copiar, mover o vincular) son válidas para la operación de arrastrar. Si no se proporciona ningún argumento, se admiten todas las acciones. Se hace referencia al objeto DragOptions en los objetos NativeDragEvent para que un destino de arrastre potencial pueda comprobar que las acciones admitidas sean compatibles con la finalidad del componente de destino. Por ejemplo, quizá un componente "trash" (papelera) acepte solamente gestos de arrastrar que permitan la acción de mover.

El ejemplo siguiente muestra cómo se inicia una operación de arrastrar para un objeto de mapa de bits cargado desde un archivo. En el ejemplo se carga una imagen y, al darse un evento `mouseDown`, se inicia la operación de arrastrar.

```
 package
{
import flash.desktop.NativeDragManager;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.display.Loader;
import flash.system.LoaderContext;
import flash.net.URLRequest;
import flash.geom.Point;
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
import flash.events.Event;
import flash.events.MouseEvent;

public class DragOutExample extends UIComponent Sprite {
    protected var fileURL:String = "app:/image.jpg";
    protected var display:Bitmap;

    private function init():void {
        loadImage();
    }
    private function onMouseDown(event:MouseEvent):void {
        var bitmapFile:File = new File(fileURL);
        var transferObject:Clipboard = createClipboard(display, bitmapFile);
        NativeDragManager.doDrag(this,
                            transferObject,
                            display.bitmapData,
                            new Point(-mouseX,-mouseY));
    }
    public function createClipboard(image:Bitmap, sourceFile:File):Clipboard {
        var transfer:Clipboard = new Clipboard();
        transfer.setData("bitmap",
                            image,
                            true);
                            // ActionScript 3 Bitmap object by value and by reference
        transfer.setData(ClipboardFormats.BITMAP_FORMAT,
                            image.bitmapData,
                            false);
                            // Standard BitmapData format
        transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
```

```
                            new Array(sourceFile),
                            false);
                            // Standard file list format
        return transfer;
    }
    private function loadImage():void {
        var url:URLRequest = new URLRequest(fileURL);
        var loader:Loader = new Loader();
        loader.load(url,new LoaderContext());
        loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onLoadComplete);
    }
    private function onLoadComplete(event:Event):void {
        display = event.target.loader.content;
        var flexWrapper:UIComponent = new UIComponent();
        flexWrapper.addChild(event.target.loader.content);
        addChild(flexWrapper);
        flexWrapper.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    }
}
}
```

Finalización de una transferencia hacia fuera

Adobe AIR 1.0 y posterior

Cuando el usuario suelta el ratón para colocar el elemento arrastrado, el objeto iniciador distribuye un evento `nativeDragComplete`. Se puede comprobar la propiedad `dropAction` del objeto de evento y tomar las medidas que corresponda. Por ejemplo, si la acción es `NativeDragAction.MOVE`,, se podría eliminar el elemento de su lugar de origen. El usuario puede abandonar un gesto de arrastrar soltando el botón del ratón mientras el cursor se encuentra fuera de un destino idóneo. El gestor de la acción de arrastrar define la propiedad `dropAction` para un gesto abandonado en `NativeDragAction.NONE`.

Compatibilidad con el gesto de arrastrar hacia dentro

Adobe AIR 1.0 y posterior

Para tener compatibilidad con el gesto de arrastrar hacia dentro, la aplicación (o, lo que es más común, un componente visual de la aplicación) debe responder a los eventos `nativeDragEnter` o `nativeDragOver`.

Pasos de una operación típica de colocar

Adobe AIR 1.0 y posterior

La siguiente secuencia de eventos es típica de una operación de colocar:

1. El usuario arrastra un objeto de portapapeles sobre un componente.
2. El componente distribuye un evento `nativeDragEnter`.
3. El controlador de eventos `nativeDragEnter` examina el objeto de evento para comprobar los formatos de datos disponibles y las acciones admitidas. Si el componente puede controlar la operación de colocar, llama a `NativeDragManager.acceptDragDrop()`.

4 El NativeDragManager cambia el cursor del ratón para indicar que se puede colocar el objeto.

5 El usuario coloca el objeto sobre el componente.

6 El componente receptor distribuye un evento `nativeDragDrop`.

7 El componente receptor lee los datos en el formato deseado del objeto Clipboard dentro del objeto de evento.

8 Si el gesto de arrastrar originó en una aplicación de AIR, el objeto interactivo iniciador distribuye un evento `nativeDragComplete`. Si el gesto originó fuera de AIR, no se envía ninguna confirmación.

Reconocimiento de un gesto de arrastrar hacia dentro

Adobe AIR 1.0 y posterior

Cuando un usuario arrastra un elemento del portapapeles para introducirlo dentro de los límites de un componente visual, el componente distribuye eventos `nativeDragEnter` y `nativeDragOver` events. Para determinar si el componente puede aceptar el elemento del portapapeles, los controladores de estos eventos pueden comprobar las propiedades `clipboard` y `allowedActions` del objeto de evento. Para indicar que el componente puede aceptar que se le coloque el elemento, el controlador de eventos debe llamar al método `NativeDragManager.acceptDragDrop()`, pasando una referencia al componente receptor. Si hay más de un detector de eventos registrado que llama al método `acceptDragDrop()`, tiene preferencia el último controlador de la lista. La llamada al método `acceptDragDrop()` sigue siendo válida hasta que el ratón sale fuera del objeto receptor, activando el evento `nativeDragExit`.

Si se admite más de una acción en el parámetro `allowedActions` que se pasa a `doDrag()`, el usuario puede pulsar una tecla modificadora para indicar cuál de las acciones admitidas pretenden realizar. El gestor de la acción de arrastrar cambia la imagen del cursor para indicar al usuario qué acción se produce si coloca el objeto. La propiedad `dropAction` del objeto NativeDragEvent notifica la acción que se pretende realizar. La acción configurada para un gesto de arrastrar no es más que orientativa. Los componentes involucrados en la transferencia deben implementar el comportamiento que corresponda. Para finalizar una acción de mover, por ejemplo, el iniciador de la operación de arrastrar puede eliminar el elemento arrastrado y el destino puede añadirlo.

El destino de arrastre puede limitar la acción de colocar a una de tres acciones posibles mediante la definición de la propiedad `dropAction` de la clase NativeDragManager. Si un usuario intenta elegir otra acción con el teclado, NativeDragManager muestra el cursor de *indisposición*. Defina la propiedad `dropAction` de los controladores para ambos eventos, `nativeDragEnter` y `nativeDragOver`.

El ejemplo siguiente ilustra un controlador de eventos para un evento `nativeDragEnter` o `nativeDragOver`. Este controlador solo acepta un gesto de arrastrar hacia dentro si el portapapeles que se arrastra contiene datos en formato de texto.

```
import flash.desktop.NativeDragManager;
import flash.events.NativeDragEvent;

public function onDragIn(event:NativeDragEvent):void{
    NativeDragManager.dropAction = NativeDragActions.MOVE;
    if(event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
        NativeDragManager.acceptDragDrop(this); //'this' is the receiving component
    }
}
```

Finalización de la operación de colocar

Adobe AIR 1.0 y posterior

Cuando el usuario coloca un elemento arrastrado en un objeto interactivo que ha aceptado el gesto, el objeto interactivo distribuye un evento `nativeDragDrop`. El controlador de este evento puede extraer los datos de la propiedad `clipboard` del objeto de evento.

Si el portapapeles contiene un formato definido por la aplicación, el parámetro `transferMode` que se pasa al método `getData()` del objeto Clipboard determina si el gestor de la acción de arrastrar devuelve una referencia o una versión serializada del objeto.

El ejemplo siguiente ilustra un controlador de eventos para el evento `nativeDragDrop`.

```
 import flash.desktop.Clipboard;
import flash.events.NativeDragEvent;

public function onDrop(event:NativeDragEvent):void {
    if (event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)) {
    var text:String =
        String(event.clipboard.getData(ClipboardFormats.TEXT_FORMAT,
                                    ClipboardTransferMode.ORIGINAL_PREFERRED));
}
```

Una vez que sale el controlador de eventos, el objeto Clipboard ya no es válido. Todo intento de acceso al objeto o a sus datos genera un error.

Actualización del aspecto visual de un componente

Adobe AIR 1.0 y posterior

Un componente puede actualizar su aspecto visual con base en los eventos NativeDragEvent. En la tabla siguiente se describen los tipos de modificaciones que efectuaría un componente característico como respuesta a los distintos eventos:

Evento	Descripción
nativeDragStart	El objeto interactivo iniciador puede utilizar el evento `nativeDragStart` para presentar una confirmación visual de que el gesto de arrastrar originó en ese objeto interactivo.
nativeDragUpdate	El objeto interactivo iniciador puede utilizar el evento nativeDragUpdate para actualizar su estado durante el gesto. (Este evento no existe en AIR para Linux.)
nativeDragEnter	Un objeto interactivo receptor potencial puede utilizar este evento para recibir la selección de entrada, o para indicar visualmente que puede o no aceptar que se le coloque el elemento.
nativeDragOver	Un objeto interactivo receptor potencial puede utilizar este evento para responder al movimiento del ratón dentro del objeto interativo, como cuando el ratón pasa a una región “activa” de un componente complejo; por ejemplo, una visualización de un plano callejero.
nativeDragExit	Un objeto interactivo receptor potencial puede utilizar este evento para restaurar su estado cuando un gesto de arrastrar se sale de sus límites.
nativeDragComplete	El objeto interactivo iniciador puede utilizar este evento para actualizar su modelo de datos asociado - eliminando un elemento de una lista, por ejemplo- y restaurar su estado visual.

Seguimiento de la posición del ratón durante un gesto de arrastrar hacia dentro

Adobe AIR 1.0 y posterior

Mientras un gesto de arrastrar permanece sobre un componente, dicho componente distribuye eventos`nativeDragOver`. Estos eventos se distribuyen cada pocos milisegundos y también cada vez que se mueve el ratón. El objeto de evento `nativeDragOver` puede utilizarse para determinar la posición del ratón sobre el componente. Tener acceso a la posición del ratón puede resultar útil si el componente receptor es complejo pero no está compuesto por subcomponentes. Por ejemplo, si la aplicación presenta un mapa de bits que contiene un plano callejero y usted desea resaltar zonas del plano cuando el usuario arrastra información hasta ellas, puede utilizar las coordenadas del ratón notificadas en el evento `nativeDragOver` para realizar un seguimiento de la posición del ratón en el plano.

Operación de arrastrar y colocar en HTML

Adobe AIR 1.0 y posterior

Para arrastrar datos hacia dentro y hacia fuera de una aplicación basada en HTML (o hacia dentro y hacia fuera del HTML visualizado en un HTMLLoader), se pueden utilizar eventos de arrastrar y colocar en HTML. La API de arrastrar y colocar en HTML permite arrastrar hacia y desde elementos DOM en el contenido HTML.

Nota: *también se pueden utilizar las API NativeDragEvent y NativeDragManager de AIR mediante la detección de eventos en el objeto HTMLLoader que contiene el contenido HTML. Sin embargo, la API de HTML está mejor integrada con el DOM de HTML y le permite controlar el comportamiento predeterminado.*

Comportamiento predeterminado de arrastrar y colocar

Adobe AIR 1.0 y posterior

El entorno HTML proporciona comportamiento predeterminado para gestos de arrastrar y colocar que implican texto, imágenes y URL. Estos tipos de datos siempre pueden arrastrarse para sacarlos de un elemento si se utiliza el comportamiento predeterminado. Sin embargo, solo se puede arrastrar texto para introducirlo en un elemento y solo a elementos en una región editable de la página. Al arrastrar texto entre o dentro de regiones editables de la página, el comportamiento predeterminado realiza una acción de mover. Al arrastrar texto a una región editable desde una región no editable o desde fuera de la aplicación, el comportamiento predeterminado lleva a cabo una acción de copia.

El comportamiento predeterminado puede suprimirse si controla uno mismo los eventos de arrastrar y colocar. Para anular el comportamiento predeterminado hay que llamar a los métodos `preventDefault()` de los objetos distribuidos para los eventos de arrastrar y colocar. Luego se pueden introducir datos en el destino y eliminar datos del origen según haga falta para llevar a cabo la acción seleccionada.

La situación predeterminada es que el usuario puede seleccionar y arrastrar cualquier texto, así como arrastrar imágenes y vínculos. Se puede utilizar la propiedad de CSS WebKit `-webkit-user-select` para controlar cómo puede seleccionarse cualquier elemento HTML. Por ejemplo, si se define `-webkit-user-select` en `none`, el contenido del elemento no será seleccionable y, por lo tanto, no se podrá arrastrar. También se puede utilizar la propiedad de CSS `-webkit-user-drag` para controlar si se puede arrastrar un elemento en su totalidad. No obstante, el contenido del elemento se trata por separado. De todos modos, el usuario puede arrastrar una parte seleccionada del texto. Para obtener más información, consulte "CSS en AIR" en la página 1015.

Eventos de arrastrar y colocar en HTML

Adobe AIR 1.0 y posterior

Los eventos distribuidos por el elemento iniciador desde el que origina una operación de arrastrar son:

Evento	Descripción
dragstart	Se distribuye cuando el usuario inicia el gesto de arrastrar. El controlador de este evento puede impedir la operación de arrastrar, si es preciso, llamando al método preventDefault() del objeto de evento. Para controlar si se pueden copiar, vincular o mover los datos arrastrados, configure la propiedad effectAllowed. El texto, las imágenes y los vínculos seleccionados se colocan en el portapapeles como parte del comportamiento predeterminado, pero se pueden configurar otros datos para el gesto de arrastrar mediante la propiedad dataTransfer del objeto de evento.
drag	Se distribuye de forma continua durante el gesto de arrastrar.
dragend	Se distribuye cuando el usuario suelta el botón del ratón para terminar el gesto de arrastrar.

Los eventos distribuidos por un destino de arrastre son:

Evento	Descripción
dragover	Se distribuye de forma continua mientras el gesto de arrastrar permanece dentro de los límites del elemento. El controlador de este evento debe definir la propiedad dataTransfer.dropEffect para indicar si se producirá una acción de copiar, mover o vincular si el usuario suelta el ratón.
dragenter	Se distribuye cuando el gesto de arrastrar entra en el elemento. Si cambia alguna propiedad del objeto dataTransfer en un controlador de eventos dragenter, esas modificaciones se verán suprimidas en cuanto se produzca el siguiente evento dragover. Por otro lado, existe un breve retardo entre un evento dragenter y el primer evento dragover que puede hacer que el cursor parpadee si se han definido propiedades distintas. En muchos casos se puede utilizar el mismo controlador para ambos eventos.
dragleave	Se distribuye cuando el gesto de arrastrar sale de los límites del elemento.
drop	Se distribuye cuando el usuario coloca los datos en el elemento. Solo se tiene acceso a los datos arrastrados dentro del controlador de este evento.

El objeto de evento que se distribuye como respuesta a estos eventos es similar a un evento de ratón. Se pueden utilizar propiedades de evento de ratón tales como (`clientX`, `clientY`) y (`screenX`, `screenY`) para determinar la posición del ratón.

La propiedad más importante de un objeto de evento drag es `dataTransfer`, que contiene los datos arrastrados. El objeto `dataTransfer` en sí tiene los siguientes métodos y propiedades:

Método o propiedad	Descripción
effectAllowed	El efecto admitido por el origen de la operación de arrastrar. Lo habitual es que este valor lo defina el controlador del evento dragstart. Consulte "Efectos de arrastrar en HTML" en la página 644.
dropEffect	El efecto elegido por el destino o el usuario. Si se define la propiedad `dropEffect` en un controlador de eventos `dragover` o `dragenter`, AIR actualiza el cursor para mostrar el efecto que se produciría si el usuario soltara el ratón. Si la definición de `dropEffect` no coincide con uno de los efectos admitidos, no se permite colocar y se presenta el cursor de *indisposición*. Si usted no ha definido la propiedad `dropEffect` como respuesta al último evento `dragover` o `dragenter`, el usuario podrá elegir entre los efectos admitidos con las teclas modificadoras normales del sistema operativo. El efecto final lo notifica la propiedad `dropEffect` del objeto distribuido para `dragend`. Si el usuario abondona la operación de colocar, soltando el ratón fuera de un destino idóneo, `dropEffect` se define en `none`.
types	Un conjunto que contiene las cadenas de tipos MIME para cada formato de datos presente en el objeto`dataTransfer`.
getData(mimeType)	Obtiene los datos en el formato especificado por el parámetro `mimeType`. El método `getData()` solo puede llamarse como respuesta al evento `drop`.
setData(mimeType)	Añade datos a `dataTransfer` en el formato especificado por el parámetro mimeType. Para añadir datos en varios formatos, llame a `setData()` para cada tipo MIME. Los datos que el comportamiento predeterminado de arrastrar haya colocado en el objeto `dataTransfer` se eliminan. El método `setData()` solo puede llamarse como respuesta al evento `dragstart`.
clearData(mimeType)	Elimina datos en el formato especificado por el parámetro `mimeType`.
setDragImage(image, offsetX, offsetY)	Configura una imagen de arrastrar personalizada. El método `setDragImage()` solo se puede llamar como respuesta al evento dragstart y únicamente cuando un elemento HTML completo se arrastra estableciendo su estilo CSS `-webkit-user-drag` en `element`. El parámetro `image` puede ser un elemento JavaScript o un objeto Image.

Tipos MIME para la operación de arrastrar y colocar en HTML

Adobe AIR 1.0 y posterior

Los tipos MIME a utilizar con un objeto `dataTransfer` de un evento de arrastrar y colocar en HTML incluyen:

Formato de datos	Tipo MIME
Texto	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
Mapa de bits	"image/x-vnd.adobe.air.bitmap"
Lista de archivos	"application/x-vnd.adobe.air.file-list"

También se pueden utilizar otras cadenas MIME, incluidas las que define la aplicación. Sin embargo, cabe la posibilidad de que otras aplicaciones no reconozcan o no puedan utilizar los datos transferidos. Es su responsabilidad añadir datos al objeto `dataTransfer` en el formato previsto.

Importante: *solo el código que se ejecute en el entorno limitado de la aplicación tiene acceso a los archivos colocados tras una operación de arrastre. Si se intenta leer o definir una propiedad de un objeto File dentro de un entorno limitado ajeno a la aplicación, se produce un error de seguridad. Para obtener más información, consulte "Gestión de la colocación de archivos en entornos limitados de HTML ajenos a la aplicación" en la página 648.*

Efectos de arrastrar en HTML

Adobe AIR 1.0 y posterior

El iniciador del gesto de arrastrar puede limitar los efectos de arrastrar que se admiten mediante la definición de la propiedad `dataTransfer.effectAllowed` en el controlador del evento `dragstart`. Se pueden utilizar los siguientes valores de cadenas:

Valor de la cadena	Descripción
"none"	No se admite ninguna operación de arrastrar.
"copy"	Los datos se copiarán en el destino, dejando los originales en su lugar.
"link"	Los datos se compartirán con el destino utilizando un vínculo a los originales.
"move"	Los datos se copiarán en el destino y se eliminarán del lugar original.
"copyLink"	Los datos se pueden copiar o vincular.
"copyMove"	Los datos se pueden copiar o mover.
"linkMove"	Los datos se pueden vincular o mover.
"all"	Los datos se pueden copiar, mover o vincular. *All* es el efecto predeterminado cuando se impide el comportamiento predeterminado.

El destino del gesto de arrastrar puede definir la propiedad `dataTransfer.dropEffect` para indicar la acción que se realiza si el usuario finaliza la operación de colocar. Si el efecto de colocar es una de las acciones admitidas, el sistema muestra el cursor de copiar, mover o vincular, según proceda. De lo contrario, el sistema muestra el cursor de *indisposición*. Si el destino no define ningún efecto de colocar, el usuario puede elegir entre las acciones admitidas con las teclas modificadoras.

Defina el valor `dropEffect` de los controladores para ambos eventos, `dragover` y `dragenter`:

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

Nota: *si bien conviene siempre definir la propiedad `dropEffect` en el controlador de `dragenter`, tenga en cuenta que el siguiente evento `dragover` restaurará el valor predeterminado de la propiedad. Defina `dropEffect` como respuesta a ambos eventos.*

Arrastrar datos hacia fuera de un elemento en HTML

Adobe AIR 1.0 y posterior

El comportamiento predeterminado permite copiar la mayoría del contenido de una página HTML arrastrándolo. Se puede controlar el contenido que puede arrastrarse utilizando las propiedades de CSS `-webkit-user-select` y `-webkit-user-drag`.

Suprima el comportamiento predeterminado de arrastrar hacia fuera en el controlador del evento `dragstart`. Llame al método `setData()` de la propiedad `dataTransfer` del objeto de evento para incluir sus propios datos en el gesto de arrastrar.

Para indicar qué efectos de arrastrar admite un objeto de origen cuando no dependa del comportamiento predeterminado, defina la propiedad `dataTransfer.effectAllowed` del objeto de evento distribuido para el evento `dragstart`. Se puede escoger cualquier combinación de efectos. Por ejemplo, si un elemento de origen admite tanto el efecto *copiar* como el efecto *vincular*, defina la propiedad en `"copyLink"`.

Configuración de los datos arrastrados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Añada los datos para el gesto de arrastrar en el controlador del evento `dragstart` con la propiedad `dataTransfer`. Utilice el método `dataTransfer.setData()` para poner los datos en el portapapeles, pasando el tipo MIME y los datos a transferir.

Por ejemplo, si tiene un elemento de imagen en la aplicación con el ID *imageOfGeorge*, podría utilizar el siguiente controlador del evento dragstart. Este ejemplo añade representaciones de una imagen de George en varios formatos de datos, lo cual aumenta la probabilidad de que otras aplicaciones puedan utilizar los datos arrastrados.

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                               new Array(dragFile));
}
```

Nota: *cuando se llama al método `setData()` del objeto `dataTransfer`, el comportamiento predeterminado de arrastrar y colocar no añade datos.*

Arrastrar datos dentro de un elemento HTML

Adobe AIR 1.0 y posterior

El comportamiento predeterminado solo permite arrastrar texto a las regiones editables de la página. Para especificar que un elemento y sus elementos secundarios sean editables, incluya el atributo `contenteditable` en la etiqueta inicial del elemento. Un documento entero también puede transformarse en editable si se define la propiedad `designMode` del objeto de documento en `"on"`.

Se puede permitir otro comportamiento de arrastrar hacia dentro en una página si se controlan los eventos `dragenter`, `dragover` y`drop` para cualquier elemento que acepte datos arrastrados.

Activación de arrastrar hacia dentro

Adobe AIR 1.0 y posterior

Para controlar el gesto de arrastrar hacia dentro, primero hay que cancelar el comportamiento predeterminado. Detecte si hay eventos `dragenter` y `dragover` en cualquiera de los elementos HTML que desee utilizar como destino. En los controladores de estos eventos, llame al método `preventDefault()` del objeto de evento distribuido. La cancelación del comportamiento predeterminado permite colocar datos arrastrados en las regiones no editables.

Obtención de los datos colocados

Adobe AIR 1.0 y posterior

Se puede tener acceso a los datos colocados en el controlador del evento `ondrop`:

```
function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

Utilice el método `dataTransfer.getData()` para leer los datos y ponerlos en el portapapeles, pasando el tipo MIME del formato de datos a leerse. Para averiguar qué formatos de datos están disponibles, utilice la propiedad `types` del objeto `dataTransfer`. El conjunto `types` contiene la cadena de tipo MIME de cada formato disponible.

Cuando cancela el comportamiento predeterminado en los eventos dragenter o dragover, usted queda responsable de introducir los datos colocados en el lugar correspondiente del documento. No existe ninguna API que convierta una posición del ratón en posición de inserción en un elemento. Esta limitación puede dificultar la implementación de gestos de arrastrar tipo inserción.

Ejemplo: Supresión del comportamiento predeterminado de arrastrar hacia dentro en HTML

Adobe AIR 1.0 y posterior

Este ejemplo implementa un destino que presenta una tabla con todos los formatos de datos disponibles en el elemento colocado.

Se utiliza el comportamiento predeterminado para poder arrastrar texto, vínculos e imágenes dentro de la aplicación. El ejemplo suprime el comportamiento predeterminado de arrastrar hacia dentro para el elemento div que sirve como destino. El paso clave para permitir que el contenido no editable acepte un gesto de arrastrar hacia dentro es llamar al método `preventDefault()` del objeto de evento distribuido tanto para el evento `dragenter` como para el evento `dragover`. Como respuesta a un evento `drop`, el controlador convierte los datos transferidos en elemento de fila de HTML e inserta la fila en una tabla para visualizarla.

```
 <html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }

    function dragEnterOverHandler(event){
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -
1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-
vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
```

```
       <p>Items to drag:</p>
    <ul id="source">
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
        Uses "-webkit-user-drag:none" style.
           </li>
           <li style="-webkit-user-select:none;">
        Uses "-webkit-user-select:none" style.
           </li>

       </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
       <p>Drag items from the source list (or elsewhere).</p>
       <table id="displayTable" border="1">
        <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap
Data</th></tr>
           <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </
td></tr>
       </table>
       </div>
</div>
</body>
</html>
```

Gestión de la colocación de archivos en entornos limitados de HTML ajenos a la aplicación

Adobe AIR 1.0 y posterior

El contenido ajeno a la aplicación no tiene acceso a los objetos File que se producen al arrastrar archivos a una aplicación de AIR. Tampoco es posible pasar uno de estos objetos File al contenido de la aplicación a través de un puente de entorno limitado. (Se debe acceder a las propiedades del objeto durante la serialización). No obstante, se pueden colocar archivos en la aplicación detectando los eventos nativeDragDrop de AIR en el objeto HTMLLoader.

Normalmente, si un usuario coloca una archivo en un fotograma que aloja contenido ajeno a la aplicación, el evento drop no se propaga desde el elemento secundario al elemento principal. Sin embargo, dado que los eventos distribuidos por HTMLLoader (que contiene todo el contenido HTML de una aplicación de AIR) no forman parte del flujo de eventos de HTML, de todos modos se puede recibir el evento drop en el contenido de la aplicación.

Para recibir el evento para colocar un archivo, el documento principal añade un detector de eventos al objeto HTMLLoader utilizando la referencia provista por `window.htmlLoader`:

```
 window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

En el ejemplo siguiente un documento principal carga una página secundaria en un entorno limitado remoto (http://localhost/). El documento principal detecta el evento `nativeDragDrop` en el objeto HTMLLoader y reseña la URL del archivo.

```
<html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

El documento secundario debe presentar un destino válido, llamando al método `preventDefault()` del objeto Event en los controladores de eventos en HTML `dragenter` y `dragover`. De lo contrario, el evento drop nunca se llevará a cabo.

```
<html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

Colocación de promesas de archivo

Adobe AIR 2 y posterior

Una promesa de archivo es un formato del portapapeles de arrastrar y colocar que permite al usuario arrastrar un archivo que aún no existe fuera de una aplicación de AIR. Por ejemplo, con el uso de promesas de archivo la aplicación puede permitir al usuario arrastrar un icono proxy a una carpeta de escritorio. El icono proxy representa un archivo o algunos datos que se sabe que están disponibles en una URL. Una vez que el usuario coloca el icono, el motor de ejecución carga los datos y escribe el archivo en la ubicación de destino.

Se puede utilizar la clase URLFilePromise en una aplicación de AIR para arrastrar y colocar archivos a los que se puede acceder en una URL. La implementación de URLFilePromise se proporciona en la biblioteca aircore como parte del AIR 2 SDK. Puede utilizar el archivo aircore.swc o aircore.swf del directorio frameworks/libs/air de SDK.

Otra alternativa es implementar su propia lógica de promesa de archivos con la interfaz IFilePromise (definida en el paquete flash.desktop del motor de ejecución).

Las promesas de archivo son similares en concepto a la representación aplazada utilizando una función de controlador de datos en el portapapeles. Utilice las promesas de archivo en lugar de la representación aplazada al arrastrar y colocar archivos. La técnica de representación aplazada puede dar lugar a pausas no deseadas en la operación de arrastrar cuando se generan o se descargan los datos. Utilice la representación aplazada para las operaciones de copiar y pegar (para las que no se admiten las promesas de archivo).

Limitaciones en el uso de promesas de archivo

Las promesas de archivo presentan las siguientes limitaciones si se comparan con otros formatos de datos que se pueden situar en un portapapeles de arrastrar y colocar:

- Las promesas de archivo solo se pueden arrastrar desde una aplicación de AIR; no se pueden colocar en una aplicación de AIR.
- Las promesas de archivo no son compatibles con todos los sistemas operativos. Utilice la propiedad `Clipboard.supportsFilePromise` para comprobar si las promesas de archivo se admiten en el sistema host. En los sistemas que no admiten promesas de archivo, se debe proporcionar un mecanismo alternativo para descargar o generar los datos de archivo.
- Las promesas de archivo no se pueden utilizar con el portapapeles de copiar y pegar (`Clipboard.generalClipboard`).

Más temas de ayuda

flash.desktop.IFilePromise

air.desktop.URLFilePromise

Colocación de archivos remotos

Adobe AIR 2 y posterior

Utilice la clase URLFilePromise para crear objetos de promesa de archivo que representen archivos o datos disponibles en una URL. Añada uno o varios objetos de promesa de archivo al portapapeles utilizando el formato del portapapeles `FILE_PROMISE_LIST`. En el siguiente ejemplo, un solo archivo, disponible en http://www.example.com/foo.txt, se descarga y se guarda en la ubicación de destino como bar.txt. (Los nombres de archivo local y remoto no tienen que coincidir.)

```
if( Clipboard.supportsFilePromise )
{
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/foo.txt");
    filePromise.relativePath = "bar.txt";

    var fileList:Array = new Array( filePromise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

Se puede permitir al usuario que arrastre varios archivos a la vez añadiendo más objetos de promesa de archivo al conjunto asignado al portapapeles. También se pueden especificar subdirectorios en la propiedad `relativePath` de modo que algunos o todos los archivos incluidos en la operación se coloquen en una subcarpeta relativa a la ubicación de destino.

En el siguiente ejemplo se muestra cómo iniciar una operación de arrastrar que incluye varias promesas de archivo. En este ejemplo, una página html, *article.html*, se sitúa en el portapapeles como promesa de archivo, junto con sus dos archivos de imagen vinculados. Las imágenes se copian en una subcarpeta *images* de modo que se mantengan los vínculos relativos.

```
if( Clipboard.supportsFilePromise )
{   //Create the promise objects
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/article.html");
    filePromise.relativePath = "article.html";

    var image1Promise:URLFilePromise = new URLFilePromise();
    image1Promise.request = new URLRequest("http://example.com/images/img_1.jpg");
    image1Promise.relativePath = "images/img_1.html";
    var image2Promise:URLFilePromise = new URLFilePromise();
    image2Promise.request = new URLRequest("http://example.com/images/img_2.jpg");
    image2Promise.relativePath = "images/img_2.jpg";


    //Put the promise objects onto the clipboard inside an array
    var fileList:Array = new Array( filePromise, image1Promise, image2Promise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    //Start the drag operation
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

Implementación de la interfaz IFilePromise

Adobe AIR 2 y posterior

Para proporcionar promesas de archivo a recursos a los que no se puede acceder utilizando un objeto URLFilePromise, es posible implementar la interfaz IFilePromise en una clase personalizada. La interfaz IFilePromise define los métodos y las propiedades utilizadas por el motor de ejecución de AIR para acceder a los datos que se escribirán en un archivo una vez colocada la promesa de archivo.

Una implementación de IFilePromise transmite otro objeto al motor de ejecución de AIR que proporciona los datos para la promesa de archivo. Este objeto debe implementar la interfaz IDataInput, que el motor de ejecución de AIR utiliza para leer los datos. Por ejemplo, la clase URLFilePromise, que implementa IFilePromise, usa un objeto URLStream como proveedor de datos.

AIR puede leer los datos de forma sincrónica o asíncrona. La implementación de IFilePromise indica qué modo de acceso se admite devolviendo el valor adecuado en la propiedad `isAsync`. Si se proporciona un acceso asíncrono a los datos, el objeto proveedor de datos debe implementar la interfaz IEventDispatcher y distribuir los eventos necesarios como, por ejemplo, `open`, `progress` y `complete`.

Se puede utilizar una clase personalizada o una de las siguientes clases incorporadas, como proveedor de datos para una promesa de archivo:

- ByteArray (sincrónica)

- FileStream (sincrónica o asíncrona)
- Socket (asíncrona)
- URLStream (asincrónica)

Para implementar la interfaz IFilePromise, es necesario proporcionar código para las siguientes funciones y propiedades:

- `open():IDataInput`: devuelve el objeto proveedor de datos en el que se leen los datos para el archivo prometido. El objeto debe implementar la interfaz IDataInput. Si los datos se proporciona de forma asíncrona, el objeto también debe implementar la interfaz IEventDispatcher y distribuir los eventos necesarios (consulte "Uso de un proveedor de datos asíncrono en una promesa de archivo" en la página 654).
- `get relativePath():String`: proporciona la ruta, incluyendo el nombre de archivo, para el archivo creado. La ruta se resuelve en función de la ubicación de destino seleccionada por el usuario en la operación de arrastrar y colocar. Para garantizar que la ruta utilice el carácter separador apropiado para el sistema operativo del ordenador host, utilice la constante `File.separator` al especificar las rutas que incluyen directorios. Se puede añadir una función captadora o emplear un parámetro constructor para permitir que la ruta se establezca en tiempo de ejecución.
- `get isAsync():Boolean`: informa al motor de ejecución de AIR si el objeto proveedor de datos proporciona su información de forma sincrónica o asíncrona.
- `close():void`: se llama mediante el motor de ejecución cuando los datos se han leído por completo (o un error detiene la lectura). Esta función se puede utilizar para limpiar recursos.
- `reportError( e:ErrorEvent ):void`: se llama mediante el motor de ejecución cuando se produce un error al leer los datos.

Todos los métodos IFilePromise se llaman mediante el motor de ejecución durante una operación de arrastrar y colocar que implica la promesa de archivo. Generalmente la lógica de la aplicación no debe llamar a ninguno de estos métodos directamente.

Uso de un proveedor de datos sincrónico en una promesa de archivo

Adobe AIR 2 y posterior

La forma más sencilla de implementar la interfaz IFilePromise consiste en utilizar un objeto proveedor de datos sincrónico como, por ejemplo, ByteArray o FileStream. En el siguiente ejemplo se crea un objeto ByteArray, relleno de datos, y se devuelve cuando se llama al método `open()`.

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

En la práctica, las promesas de archivo sincrónicas presentan una utilidad limitada. Si la cantidad de datos es pequeña, se puede crear fácilmente un archivo en un directorio temporal y añadir un conjunto de lista de archivos normal al portapapeles de arrastrar y colocar. Por otra parte, si la cantidad de información es grande o la generación de datos requiere mucho procesamiento, es necesario contar un proceso sincrónico largo. Los procesos sincrónicos largos pueden bloquear las actualizaciones de la IU durante una cantidad considerable de tiempo y hacer que la aplicación no responda. Para evitar este problema, se puede crear un proveedor de datos asíncrono controlado por un temporizador.

Uso de un proveedor de datos asíncrono en una promesa de archivo

Adobe AIR 2 y posterior

Cuando se utiliza un objeto proveedor de datos asíncrono, la propiedad `isAsync` de IFilePromise se debe establecer en `true` y el objeto devuelto por el método `open()` debe implementar la interfaz IEventDispatcher. El motor de ejecución detecta varios eventos alternativos de modo que los distintos objetos incorporados se puedan utilizar como proveedores de datos. Por ejemplo, los eventos `progress` se distribuyen mediante los objetos FileStream y URLStream, mientras que los eventos `socketData` se distribuyen a través de los objetos Socket. El motor de ejecución detecta los eventos apropiados a partir de todos estos objetos.

Los siguientes eventos controlan el proceso de lectura de datos desde el objeto proveedor:

- Event.OPEN: informa al motor de ejecución de que el origen de datos está listo.
- ProgressEvent.PROGRESS: indica al motor de ejecución que los datos están disponibles. El motor de ejecución leerá la cantidad de datos disponibles desde el objeto proveedor de datos.
- ProgressEvent.SOCKET_DATA: indica al motor de ejecución que los datos están disponibles. El evento `socketData` se distribuye mediante los objetos basados en socket. Para otros tipos de objetos, se debe distribuir un evento `progress`. (El motor de ejecución detecta ambos eventos para percibir el momento en que se pueden leer los datos.)
- Event.COMPLETE: indica al motor de ejecución que todos los datos se han leído.
- Event.CLOSE: indica al motor de ejecución que todos los datos se han leído. (El motor de ejecución detecta `close` y `complete` para este propósito.)
- IOErrorEvent.IOERROR: indica al motor de ejecución que se ha producido un error al leer los datos. El motor de ejecución anula la creación del archivo y llama al método `close()` de IFilePromise.
- SecurityErrorEvent.SECURITY_ERROR: indica al motor de ejecución que se ha producido un error de seguridad. El motor de ejecución anula la creación del archivo y llama al método `close()` de IFilePromise.
- HTTPStatusEvent.HTTP_STATUS: el motor de ejecución lo utiliza, junto con `httpResponseStatus`, para garantizar que los datos disponibles representan el contenido deseado, en lugar de un mensaje de error (por ejemplo, una página 404). Los objetos basados en el protocolo HTTP deben distribuir este evento.
- HTTPStatusEvent.HTTP_RESPONSE_STATUS: el motor de ejecución lo utiliza, junto con `httpStatus`, para garantizar que los datos disponibles representan el contenido deseado. Los objetos basados en el protocolo HTTP deben distribuir este evento.

El proveedor de datos debe distribuir estos eventos en la siguiente secuencia:

1. evento `open`
2. eventos `progress` o `socketData`
3. evento `complete` o `close`

Nota: *los objetos incorporados, FileStream, Socket y URLStream distribuyen los eventos adecuados de forma automática.*

En el siguiente ejemplo se crea una promesa de archivo utilizando un proveedor de datos asíncrono y personalizado. La clase del proveedor de datos amplía ByteArray (para la compatibilidad con IDataInput) e implementa la interfaz IEventDispatcher. En cada evento del temporizador, el objeto genera un campo de datos y distribuye un evento de progreso para indicar al motor de ejecución que los datos están disponibles. Una vez generados datos suficientes, el objeto distribuye un evento completo.

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }

    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable,
bytesAvailable + chunkSize);
            dispatchEvent( progress )
```

```
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function,
useCapture:Boolean=false, priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function,
useCapture:Boolean=false):void
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

Nota: *debido a que la clase AsyncDataProvider del ejemplo amplía ByteArray, no puede ampliar también EventDispatcher. Para implementar la interfaz IEventDispatcher, la clase utiliza un objeto interno EventDispatcher y reenvía las llamadas del método IEventDispatcher a ese objeto interno. También se puede ampliar EventDispatcher e implementar IDataInput (o implementar ambas interfaces).*

La implementación asíncrona de IFilePromise resulta casi idéntica a la implementación sincrónica. Las principales diferencias radican en que `isAsync` devuelve el valor `true` y que el método `open()` devuelve un objeto de datos asíncrono:

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }

        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

Capítulo 36: Trabajo con menús

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Utilice las clases de la API del menú contextual para modificar el menú contextual en las aplicaciones de Flex y Flash Player basadas en la web.

Se deben utilizar las clases en la API del menú nativo para definir los menús de aplicación, ventana, contextuales y emergentes en Adobe® AIR®.

Menú: conceptos básicos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para ver una explicación rápida y ejemplos de código de la creación de menús nativos en aplicaciones de AIR, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Adding native menus to an AIR application (Cómo añadir menús nativos a una aplicación de AIR, en inglés) (Flex)
- Adding native menus to an AIR application (Cómo añadir menús nativos a una aplicación de AIR, en inglés) (Flash)

Las clases de menú nativo permiten acceder a las funciones del menú nativo del sistema operativo donde se ejecuta la aplicación. Se pueden utilizar los objetos NativeMenu para menús de aplicación (disponibles en Mac OS X), menús de ventana (disponibles en Windows y Linux), menús contextuales y menús emergentes.

Fuera de AIR, se pueden utilizar las clases del menú contextual para modificar el menú contextual que muestra automáticamente Flash Player cuando un usuario hace clic con el botón derecho o presiona comando y hace clic en un objeto de la aplicación. (En las aplicaciones de AIR no se muestra ningún menú contextual automático.)

Clases de menú

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Entre las clases de menú se incluyen:

Paquete	Clases
flash.display	• NativeMenu • NativeMenuItem
flash.ui	• ContextMenu • ContextMenuItem
flash.events	• Event • ContextMenuEvent

Variedades de menú

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

AIR admite los siguientes tipos de menús:

Menús contextuales Los menús contextuales se abren en respuesta a un clic con el botón derecho del ratón o a un clic de un comando en un objeto interactivo en el contenido SWF o en un elemento de documento en un contenido HTML.

En el motor de ejecución de Flash Player, un menú contextual se muestra automáticamente. Las clases ContextMenu y ContextMenuItem se pueden emplear para añadir comandos propios al menú. También se pueden eliminar algunos, pero no todos, comandos incorporados.

En el motor de ejecución de AIR, se puede crear un menú contextual utilizando la clase NativeMenu o ContextMenu. En el contenido HTML en AIR, se puede utilizar el Webkit HTML y las API JavaScript para añadir menús contextuales a elementos HTML.

Menús de aplicación (solo AIR) Un menú de aplicación es un menú global que se aplica a toda la aplicación. Los menús de aplicación son compatibles en Mac OS X, pero no en Windows o Linux. En Mac OS X, el sistema operativo automáticamente crea un menú de aplicación. Se puede utilizar la API del menú AIR para añadir elementos y submenús secundarios a los menús estándar. Se pueden añadir detectores para gestionar los comandos de menú existentes. O bien se pueden quitar elementos existentes.

Menús de ventana (solo AIR) Un menú de ventana está asociado con una sola ventana y se muestra debajo de la barra de título. Se pueden añadir los menús a una ventana creando un objeto NativeMenu y asignándolo a la propiedad `menu` del objeto NativeWindow. Los menús de ventana son compatibles con los sistemas operativos Windows y Linux, pero no en Mac OS X. Los menús de ventana nativos solo se pueden utilizar con ventanas que tienen fondo cromático del sistema.

Menús del icono de acoplamiento y de bandeja del sistema (solo AIR) Estos menús de iconos son similares a los menús contextuales y se asignan a un icono de aplicación en el acoplamiento de Mac OS X o las áreas de notificación en la barra de tareas de Windows y Linux. Los menús del icono de acoplamiento y bandeja del sistema utilizan la clase NativeMenu. En Mac OS X, los elementos en el menú se añaden por encima de los elementos del sistema operativo estándar. En Windows o Linux, no hay un menú estándar.

Menús emergentes (solo AIR) Un menú emergente de AIR es como un menú contextual, pero no necesariamente está asociado con un determinado objeto o elemento de aplicación. Los menús emergentes se pueden mostrar en cualquier parte de una ventana llamando al método `display()` de cualquier objeto NativeMenu.

Menús personalizados El sistema operativo invoca los menús nativos y, como tal, existen fuera de los modelos de representación de Flash y HTML. En lugar de utilizar menús nativos, siempre puede crear sus propios menús no nativos y personalizados utilizando MXML, ActionScript o JavaScript (en AIR). Estos menús deben procesarse por completo dentro del contenido de la aplicación.

Menús de Flex La arquitectura Adobe® Flex™ proporciona una serie de componentes de menú Flex. El motor de ejecución y no el sistema operativo invoca los menús Flex y no son menús *nativos*. El componente del menú Flex se puede utilizar para ventanas Flex que no tienen un fondo cromático del sistema. Otra ventaja de utilizar el componente de menú Flex es que se pueden especificar menús de forma declarativa en el formato MXML. Si se utiliza la arquitectura de Flex, se utilizan las clases del menú Flex para los menús de ventana en lugar de las clases nativas.

Menús predeterminados (solo AIR)

El sistema operativo o una clase incorporada de AIR proporciona los siguientes menús predeterminados:

- Menú de aplicación en Mac OS X

- Menú de icono de acoplamiento en Mac OS X
- Menú contextual para texto e imágenes seleccionados en un contenido HTML
- Menú contextual para texto seleccionado en un objeto TextField (o en un objeto que amplía TextField)

Menús contextuales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el contenido SWF, a cualquier objeto que hereda de InteractiveObject se le puede asignar un menú contextual asignando un objeto de menú a la propiedad `contextMenu`. Se incluyen varios comandos de manera predeterminada, incluidos Forward (Avanzar), Back (Atrás), Print (Imprimir), Quality (Calidad) y Zoom. En el motor de ejecución de AIR, el objeto de menú asignado a `contextMenu` puede ser del tipo NativeMenu o ContextMenu. En el motor de ejecución de Flash Player, solo está disponible la clase ContextMenu.

Puede detectar eventos de menú nativo o eventos de menús contextuales con el uso de las clases ContextMenu y ContextMenuItem; ambas se distribuyen. Una ventaja que ofrecen las propiedades del objeto ContextMenuEvent es que contextMenuOwner identifica el objeto al que se asocia el menú y `mouseTarget` identifica el objeto en el que se hizo clic para abrir el menú. Esta información no está disponible en el objeto NativeMenuEvent.

El siguiente ejemplo crea un objeto Sprite y añade un simple menú contextual de edición:

```
 var sprite:Sprite = new Sprite();
sprite.contextMenu = createContextMenu()
private function createContextMenu():ContextMenu{
    var editContextMenu:ContextMenu = new ContextMenu();
    var cutItem:ContextMenuItem = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItem = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItem = new ContextMenuItem("Paste")
    pasteItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doPasteCommand);
    editContextMenu.customItems.push(pasteItem);

    return editContextMenu
}
private function doCutCommand(event:ContextMenuEvent):void{trace("cut");}
private function doCopyCommand(event:ContextMenuEvent):void{trace("copy");}
private function doPasteCommand(event:ContextMenuEvent):void{trace("paste");}
```

Nota: *a diferencia del contenido SWF que se muestra en un entorno de navegador, los menús contextuales en AIR no tienen comandos incorporados.*

Personalización de un menú contextual de Flash Player

En un navegador o proyector, los menús contextuales en el contenido SWF siempre disponen de elementos incorporados. Se pueden eliminar todos los comandos predeterminados del menú, excepto Configuración y Acerca de. Si se establece la propiedad `showDefaultContextMenu` del objeto Stage en `false`, se eliminan estos comandos del menú contextual.

Para crear un menú contextual personalizado para un objeto de visualización específico hay que crear una nueva instancia de la clase ContextMenu, llamar al método `hideBuiltInItems()` y asignar dicha instancia a la propiedad `contextMenu` de la instancia de DisplayObject. El siguiente ejemplo proporciona un cuadrado dibujado dinámicamente con un comando de menú contextual para aplicarle un color aleatorio:

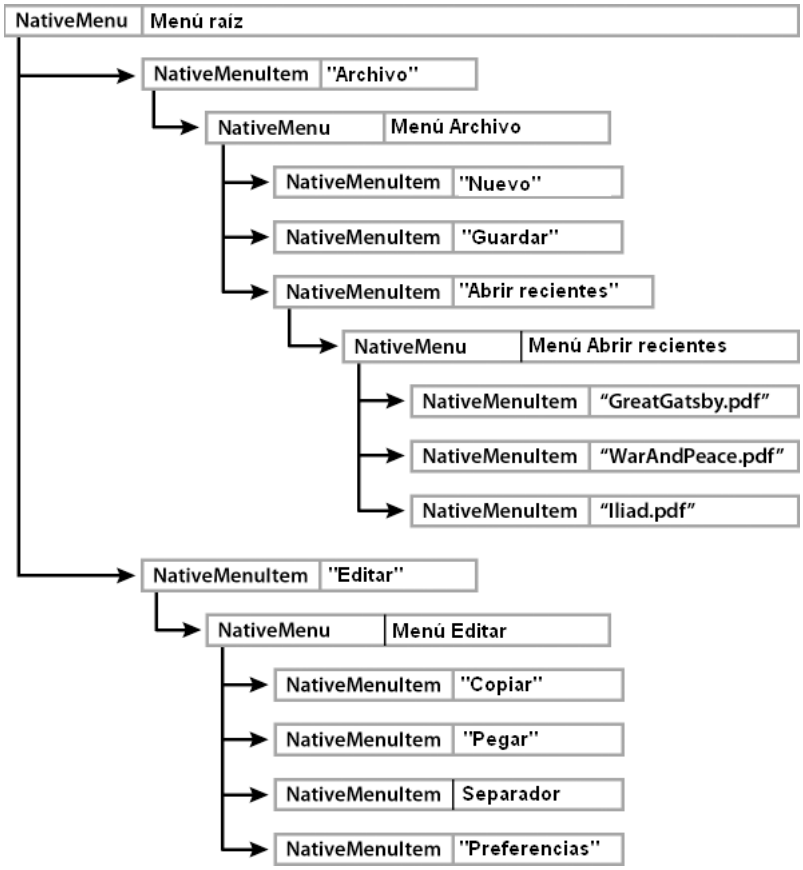
```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() *
512) - 255, (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

Estructura del menú nativo (AIR)

Adobe AIR 1.0 y posterior

Los menús nativos, por naturaleza, se crean en jerarquías. Los objetos NativeMenu contienen objetos secundarios NativeMenuItem. Los objetos NativeMenuItem que, a su vez, representan los submenús pueden tener objetos NativeMenu. El objeto de menú principal o raíz en la estructura representa la barra de menús para los menús de aplicación y de ventana. (Los menús contextuales, de icono y los menús emergentes no cuentan con una barra de menús).

El siguiente diagrama muestra la estructura de un menú típico. El menú raíz representa la barra de menús y contiene dos elementos de menú que hacen referencia a un submenú *Archivo* y a un submenú *Editar*. El submenú Archivo en esta estructura contiene dos elementos de comando y un elemento que hace referencia a un submenú *Menú Abrir reciente*, que contiene tres elementos. El submenú Editar contiene tres comandos y un separador.



La definición de un submenú requiere un objeto NativeMenu y NativeMenuItem. El objeto NativeMenuItem define la etiqueta que se muestra en el menú principal y permite al usuario abrir el submenú. El objeto NativeMenu actúa como un contenedor para los elementos en el submenú. El objeto NativeMenuItem hace referencia al objeto NativeMenu a través de la propiedad `submenu` de NativeMenuItem.

Para ver un ejemplo de código que crea este menú, consulte "Ejemplo de menú nativo: Menú de aplicación y de ventana (AIR)" en la página 670.

Eventos de menú

Adobe AIR 1.0 y posterior

Los objetos NativeMenu y NativeMenuItem distribuyen eventos `preparing`, `displaying` y `select`:

Preparing: siempre que el objeto está a punto de comenzar una interacción con el usuario, los elementos de menú distribuyen un evento `preparing` a cualquiera de los detectores registrados. La interacción incluye la apertura del menú o la selección de un elemento con un método abreviado de teclado.

Nota: *el evento* `preparing` *solo está disponible para Adobe AIR 2.6 y posterior.*

Displaying: Inmediatamente antes de visualizar un menú, el menú y los elementos de menú distribuyen un evento `displaying` a cualquier detector registrado.

Los eventos `preparing` y `displaying` proporcionan la oportunidad de actualizar los contenidos de menú o la apariencia de elementos antes de que se muestre al usuario. Por ejemplo, en el detector para el evento `displaying` de un menú "Abrir reciente", se pueden cambiar los elementos de menú para reflejar la lista actual de documentos recientemente visualizados.

Si se elimina el elemento de menú cuyo método abreviado de teclado activó un evento `preparing`, la interacción con el menú se cancela y no se distribuye ningún evento`select`.

Las propiedades `target` y `currentTarget` del evento son ambas el objeto en el que se registra el detector: el propio menú o uno de sus elementos.

El evento `preparing` se distribuye antes del evento `displaying`. Generalmente se detecta un evento o el otro, no ambos.

Select: Cuando el usuario elige un elemento de comando, el elemento distribuye un evento `select` a cualquier detector registrado. Los elementos de submenú y separador no se pueden seleccionar y por ende nunca distribuyen un evento `select`.

Un evento `select` se propaga desde un elemento de menú al menú que lo contienen hasta el menú raíz. Se pueden detectar eventos `select`directamente en un elemento y en niveles superiores en la estructura del menú. Cuando se detecta el evento `select` en un menú, se puede identificar el elemento seleccionado utilizando la propiedad de evento `target`. A medida que el evento se propaga a través de la jerarquía, la propiedad `currentTarget` del objeto de evento identifica el objeto de menú actual.

Nota: *los objetos ContextMenu y ContextMenuItem distribuyen eventos* `menuItemSelect` *y* `menuSelect`*, así como eventos* `select`*,* `preparing` *y* `displaying`*.*

Equivalentes de teclas para comandos de menú nativo (AIR)

Adobe AIR 1.0 y posterior

Se puede asignar un equivalente de tecla (a veces llamado acelerador) a un comando de menú. El elemento de menú distribuye un evento `select` a cualquier detector registrado cuando se presiona la tecla o combinación de teclas. El menú que contiene el elemento debe ser parte del menú de aplicación o de la ventana activa para que se invoque el comando.

Los equivalentes de teclas tienen dos partes, una cadena que representa la tecla principal y un conjunto de teclas modificadoras que también se deben presionar. Para asignar la tecla principal, establezca la propiedad `keyEquivalent` del elemento de menú a la cadena de un solo carácter para dicha tecla. Se utiliza una letra mayúscula, la tecla Mayús se añade automáticamente al conjunto modificador.

En Mac OS X, el modificador predeterminado es la tecla de comando (`Keyboard.COMMAND`). En Windows y Linux, es la tecla de control (`Keyboard.CONTROL`). Estas teclas predeterminadas se añaden automáticamente al conjunto modificador. Para asignar diferentes teclas modificadoras, asigne un nuevo conjunto que contenga los códigos de tecla deseados a la propiedad `keyEquivalentModifiers`. El conjunto predeterminado se sobrescribe. Si utiliza los modificadores predeterminados o si asigna su propio conjunto modificador, se añade la tecla Mayús si la cadena que asigna a la propiedad `keyEquivalent` es una letra mayúscula. Las constantes de los códigos de tecla para utilizar en las teclas modificadoras se definen en la clase Keyboard.

La cadena equivalente de la tecla asignada se muestra automáticamente junto al nombre del elemento de menú. El formato depende del sistema operativo del usuario y las preferencias del sistema.

Nota: *si asigna el valor* `Keyboard.COMMAND` *a un conjunto de modificadores de teclas en el sistema operativo Windows, no se muestra ningún equivalente de tecla en el menú. Sin embargo, se debe utilizar la tecla de control para activar el comando de menú.*

En el siguiente ejemplo se asigna `Ctrl+Mayús+G` como el equivalente de tecla para un elemento de menú:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

En este ejemplo se asigna `Ctrl+Mayús+G` como el equivalente de tecla configurando directamente el conjunto modificador:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [Keyboard.CONTROL];
```

Nota: *los equivalentes de teclas solo se activan para menús de ventana y de aplicación. Si se añade un equivalente de tecla a un menú contextual o emergente, el mismo se muestra en la etiqueta de menú, pero el comando de menú asociado nunca se invoca.*

Letras de selección (AIR)

Adobe AIR 1.0 y posterior

Las letras de selección son parte de la interfaz del teclado del sistema operativo a los menús. Linux, Mac OS X y Windows permiten a los usuarios abrir menús y seleccionar comandos con el teclado, pero hay pequeñas diferencias.

En Mac OS X, el usuario escribe las primeras letras del menú o comando y presiona la tecla de retorno. La propiedad `mnemonicIndex` se omite.

En Windows, solo una letra es significativa. Como valor predeterminado, la letra significativa es el primer carácter de la etiqueta, pero si se asigna una letra de selección al elemento de menú, entonces el carácter significativo se convierte en la letra designada. Si dos elementos en un menú tienen el mismo carácter significativo (independientemente si se ha asignado una letra de selección) entonces la interacción del teclado con el menú cambia ligeramente. En lugar de presionar una sola letra para seleccionar el menú o comando, el usuario debe presionar la letra tantas veces como sea necesario para resaltar el elemento deseado y luego presionar la tecla Intro para completar la selección. Para mantener un comportamiento coherente, debe asignar una letra de selección exclusiva para cada elemento en un menú para los menús de ventana.

en Linux, no se proporciona ninguna letra de selección predeterminada. Es necesario especificar un valor para la propiedad `mnemonicIndex` de un elemento de menú para proporcionar una letra de selección.

Se debe especificar la letra de selección como un índice en la cadena de la etiqueta. El índice del primer carácter en una etiqueta es 0. Por consiguiente, para utilizar “r” como la letra de selección para un elemento de menú denominado “Formato,” se debe establecer la propiedad `mnemonicIndex` igual a 2.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

Estado de los elementos de menú

Adobe AIR 1.0 y posterior

Los elementos de menú tienen dos propiedades de estado, `checked` y `enabled`:

checked Se debe establecer en `true` para mostrar una marca de verificación junto a la etiqueta del elemento.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

enabled Seleccione el valor `true` o `false` para controlar si el comando está activado o no. Los elementos desactivados aparecen "sombreados" y no distribuyen eventos `select`.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

Asociación de un objeto a un elemento de menú

Adobe AIR 1.0 y posterior

La propiedad `data` de la clase NativeMenuItem permite hacer referencia a un objeto arbitrario en cada elemento. Por ejemplo, en un menú "Abrir reciente", puede asignar el objeto File para cada documento a cada elemento de menú.

```
var file:File = File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

Creación de menús nativos (AIR)

Adobe AIR 1.0 y posterior

Este tema describe la manera de crear los diferentes tipos de menús nativos admitidos en AIR.

Creación de un objeto de menú raíz

Adobe AIR 1.0 y posterior

Para crear un objeto NativeMenu para que actúe como la raíz del menú, se debe utilizar el constructor NativeMenu:

```
var root:NativeMenu = new NativeMenu();
```

Para los menús de aplicación y de ventana, el menú raíz representa la barra de menús y solo debe tener los elementos que abren submenús. Los menús contextual y emergentes no tienen una barra de menús, por lo que el menú raíz puede tener líneas separadoras y comandos así como submenús.

Después de crear el menú, se pueden añadir elementos de menú. Los elementos aparecen en el menú en el orden en que se añaden, a menos que se añadan los elementos en un índice específico utilizando el método `addItemAt()` de un objeto de menú.

Se puede asignar el menú como un menú de aplicación, de ventana, de icono o contextual o mostrarlo como un menú emergente como se muestra en las siguientes secciones:

Definición del menú de aplicación o el menú de ventana

Es importante que el código se acomode a los menús de la aplicación (admitidos en Mac OS) y a los menús de ventana (admitidos en otros sistemas operativos)

```
var root:NativeMenu = new NativeMenu();
if (NativeApplication.supportsMenu)
{
    NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

Nota: *Mac OS define un menú que contiene elementos estándares para cada aplicación. Si se asigna un nuevo objeto NativeMenu a la propiedad* `menu` *del objeto NativeApplication se remplaza el menú estándar. También se puede utilizar el menú estándar en lugar de sustituirlo.*

Adobe Flex proporciona una clase FlexNativeMenu para crear fácilmente menús que funcionen en distintas plataformas. Si está utilizando Flex Framework, use las clases FlexNativeMenu en lugar de la clase NativeMenu.

Definición de un menú contextual en un objeto interactivo

```
interactiveObject.contextMenu = root;
```

Definición de un menú de icono de acoplamiento o un menú de icono de la bandeja del sistema

Es importante que el código se acomode a los menús de la aplicación (admitidos en Mac OS) y a los menús de ventana (admitidos en otros sistemas operativos)

```
if (NativeApplication.supportsSystemTrayIcon)
{
    SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
}
else if (NativeApplication.supportsDockIcon)
{
    DockIcon(NativeApplication.nativeApplication.icon).menu = root;
}
```

Nota: *Mac OS X define un menú estándar para el icono de acoplamiento de la aplicación. Cuando se asigna un nuevo NativeMenu a la propiedad de menú del objeto DockIcon, los elementos en ese menú se muestran arriba de los elementos estándar. No se pueden quitar, acceder ni modificar los elementos de menú estándar.*

Visualización de un menú como un menú emergente

```
root.display(stage, x, y);
```

Más temas de ayuda

Desarrollo de aplicaciones de AIR para varias plataformas

Creación de un submenú

Adobe AIR 1.0 y posterior

Para crear un submenú, se añade un objeto NativeMenuItem al menú principal y luego se asigna el objeto NativeMenu definiendo el submenú a la propiedad `submenu` del elemento. AIR proporciona dos modos de crear elementos de submenú y los objetos de menú asociados:

Se puede crear un elemento de menú y el objeto de menú relacionado en un paso con el método `addSubmenu()`:

```
var editMenuItem:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

También se puede crear el elemento de menú y asignar el objeto de menú a la propiedad `submenu` de forma separada:

```
var editMenuItem:NativeMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new NativeMenu();
```

Creación de un comando de menú

Adobe AIR 1.0 y posterior

Para crear un comando de menú, añada un objeto NativeMenuItem a un menú y añada un detector de evento que hace referencia a la función que implementa el comando de menú:

```
var copy:NativeMenuItem = new NativeMenuItem("Copy", false);
copy.addEventListener(Event.SELECT, onCopyCommand);
editMenu.addItem(copy);
```

Se puede detectar un evento `select` en el elemento de comando mismo (como se muestra en el ejemplo) o se puede detectar el evento `select` en un objeto de menú principal.

Nota: *los elementos de menú que representan submenús y líneas separadoras no distribuyen eventos `select` y por ende no se pueden utilizar como comandos.*

Creación de una línea separadora de menú

Adobe AIR 1.0 y posterior

Para crear una línea separadora, se debe crear un NativeMenuItem, definir el parámetro `isSeparator` en `true` en el constructor. Luego añadir el elemento separador al menú en la ubicación correcta:

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
editMenu.addItem(separatorA);
```

La etiqueta especificada para el separador, si hay una, no se muestra.

Menús contextuales en HTML (AIR)

Adobe AIR 1.0 y posterior

En HTML el contenido que se muestra utilizando el objeto HTMLLoader, el evento `contextmenu` se puede utilizar para mostrar un menú contextual. Como valor predeterminado, un menú contextual se muestra automáticamente cuando el usuario invoca el evento de menú contextual en el texto seleccionado (haciendo clic con el botón derecho del ratón o haciendo clic en el comando del texto) Para impedir que el menú predeterminado se abra, se debe detectar el evento `contextmenu`y llamar el método `preventDefault()` del objeto de evento:

```
function showContextMenu(event){
    event.preventDefault();
}
```

Luego se puede mostrar un menú contextual personalizado utilizando las técnicas DHTML o mostrando un menú contextual nativo de AIR. En el siguiente ejemplo se muestra un menú contextual nativo llamando el método `display()` del menú en respuesta al evento `contextmenu` HTML:

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context
menu.</p>
</body>
</html>
```

Visualización de menús nativos desplegables (AIR)

Adobe AIR 1.0 y posterior

Se puede mostrar cualquier objeto NativeMenu en cualquier momento y ubicación arriba de una ventana llamando al método `display()` del menú. El método requiere una referencia al escenario; y por consiguiente, solo el contenido en el entorno limitado la aplicación puede mostrar un menú como un menú emergente.

El siguiente método muestra el menú definido por un objeto NativeMenu denominado `popupMenu` en respuesta a un clic del ratón:

```
private function onMouseClick(event:MouseEvent):void {
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

Nota: *no se necesita mostrar el menú en respuesta directa a un evento Cualquier método puede llamar a la función* `display()`.

Gestión de eventos de menú

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un menú distribuye eventos cuando el usuario selecciona el menú o cuando el usuario selecciona un elemento de menú.

Resumen de eventos para clases de menús

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Añada detectores de eventos a los menús o elementos individuales para gestionar eventos de menús.

Objeto	Eventos detectados
NativeMenu (AIR)	Event.PREPARING (Adobe AIR 2.6 y posterior) Event.DISPLAYING Event.SELECT (propagado de elementos secundarios y submenús)
NativeMenuItem (AIR)	Event.PREPARING (Adobe AIR 2.6 y posterior) Event.SELECT Event.DISPLAYING (propagado del menú principal)
ContextMenu	ContextMenuEvent.MENU_SELECT
ContextMenuItem	ContextMenuEvent.MENU_ITEM_SELECT Event.SELECT (AIR)

Selección de eventos de menú

Adobe AIR 1.0 y posterior

Para gestionar un clic en un elemento de menú, añada un detector de evento para el evento `select` al objeto NativeMenuItem:

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommandX.addEventListener(Event.SELECT, doCommandX)
```

Debido a que los eventos `select` se propagan a los menús que los contienen, también puede detectar eventos select en un menú principal. Cuando se detecta a nivel de menú, se puede utilizar la propiedad `target` del objeto de evento para determinar el comando de menú seleccionado. En el siguiente ejemplo se rastrea la etiqueta del comando seleccionado:

```
 var colorMenuItem:NativeMenuItem = new NativeMenuItem("Choose a color");
var colorMenu:NativeMenu = new NativeMenu();
colorMenuItem.submenu = colorMenu;

var red:NativeMenuItem = new NativeMenuItem("Red");
var green:NativeMenuItem = new NativeMenuItem("Green");
var blue:NativeMenuItem = new NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(NativeApplication.supportsMenu){
    NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT, colorChoice);
} else if (NativeWindow.supportsMenu){
    var windowMenu:NativeMenu = new NativeMenu();
    this.stage.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(Event.SELECT, colorChoice);
}

function colorChoice(event:Event):void {
    var menuItem:NativeMenuItem = event.target as NativeMenuItem;
    trace(menuItem.label + " has been selected");
}
```

Si se utiliza la clase ContextMenuItem, se puede detectar el evento `select` o el evento `menuItemSelect`. El evento `menuItemSelect` proporciona información adicional sobre el objeto propietario del menú contextual, pero no se propaga a los menús que lo contienen.

Visualización de eventos de menú

Adobe AIR 1.0 y posterior

Para gestionar la apertura de un menú, se puede añadir un detector para el evento `displaying`, que se distribuye antes de visualizar un menú. Puede utilizar el evento displaying para actualizar el menú, por ejemplo añadiendo o quitando elementos o actualizando los estados activados o seleccionados de elementos individuales. También se puede detectar el evento `menuSelect` de un objeto ContextMenu.

En AIR 2.6 y posterior, se puede utilizar el evento `preparing` para actualizar un menú como respuesta a mostrar un menú o seleccionar un elemento con un método abreviado de teclado.

Ejemplo de menú nativo: Menú de aplicación y de ventana (AIR)

Adobe AIR 1.0 y posterior

El siguiente ejemplo crea el menú mostrado en “Estructura del menú nativo (AIR)” en la página 661.

El menú está diseñado para funcionar en Windows, en el que solo se admiten menús de ventana, y en Mac OS X, en el que solo se admiten menús de aplicación. Para hacer una distinción, el constructor de la clase MenuExample verifica las propiedades estáticas `supportsMenu` de las clases NativeWindow y NativeApplication. Si `NativeWindow.supportsMenu` es `true`, entonces el constructor crea un objeto NativeMenu para la ventana y luego crea y añade los submenús Archivo y Editar. Si `NativeApplication.supportsMenu` es `true`, entonces el constructor crea y añade los menús Archivo y Editar al menú existente proporcionado por el sistema operativo Mac OS X.

El ejemplo también muestra la gestión de eventos de menú. El evento `select` se gestiona a nivel del elemento y también a nivel del menú. Cada menú en la cadena desde el menú que contiene el elemento seleccionado hasta el menú raíz responde al evento `select`. El evento `displaying` se utiliza con el menú "Abrir reciente". Justo antes de que se abra el menú, los elementos en el menú se actualizan del conjunto de documentos recientes (que en realidad no cambia en este ejemplo). Aunque no se muestra en este ejemplo, también se pueden detectar eventos `displaying` en elementos individuales.

```
package {
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filesystem.File;
    import flash.desktop.NativeApplication;

    public class MenuExample extends Sprite
    {
        private var recentDocuments:Array =
            new Array(new File("app-storage:/GreatGatsby.pdf"),
                      new File("app-storage:/WarAndPeace.pdf"),
                      new File("app-storage:/Iliad.pdf"));

        public function MenuExample()
        {
            var fileMenu:NativeMenuItem;
            var editMenu:NativeMenuItem;

            if (NativeWindow.supportsMenu){
                stage.nativeWindow.menu = new NativeMenu();
                stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);
                fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }

            if (NativeApplication.supportsMenu){
                NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT,
selectCommandMenu);
                fileMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }
        }
```

```
public function createFileMenu():NativeMenu {
    var fileMenu:NativeMenu = new NativeMenu();
    fileMenu.addEventListener(Event.SELECT, selectCommandMenu);

    var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New"));
    newCommand.addEventListener(Event.SELECT, selectCommand);
    var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save"));
    saveCommand.addEventListener(Event.SELECT, selectCommand);
    var openRecentMenu:NativeMenuItem =
            fileMenu.addItem(new NativeMenuItem("Open Recent"));
    openRecentMenu.submenu = new NativeMenu();
    openRecentMenu.submenu.addEventListener(Event.DISPLAYING,
                                    updateRecentDocumentMenu);
    openRecentMenu.submenu.addEventListener(Event.SELECT, selectCommandMenu);

    return fileMenu;
}

public function createEditMenu():NativeMenu {
    var editMenu:NativeMenu = new NativeMenu();
    editMenu.addEventListener(Event.SELECT, selectCommandMenu);

    var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy"));
    copyCommand.addEventListener(Event.SELECT, selectCommand);
    copyCommand.keyEquivalent = "c";
    var pasteCommand:NativeMenuItem =
            editMenu.addItem(new NativeMenuItem("Paste"));
    pasteCommand.addEventListener(Event.SELECT, selectCommand);
    pasteCommand.keyEquivalent = "v";
    editMenu.addItem(new NativeMenuItem("", true));
    var preferencesCommand:NativeMenuItem =
            editMenu.addItem(new NativeMenuItem("Preferences"));
    preferencesCommand.addEventListener(Event.SELECT, selectCommand);

    return editMenu;
}

private function updateRecentDocumentMenu(event:Event):void {
    trace("Updating recent document menu.");
    var docMenu:NativeMenu = NativeMenu(event.target);

    for each (var item:NativeMenuItem in docMenu.items) {
        docMenu.removeItem(item);
    }

    for each (var file:File in recentDocuments) {
        var menuItem:NativeMenuItem =
                docMenu.addItem(new NativeMenuItem(file.name));
        menuItem.data = file;
        menuItem.addEventListener(Event.SELECT, selectRecentDocument);
    }
}

private function selectRecentDocument(event:Event):void {
    trace("Selected recent document: " + event.target.data.name);
}
```

```
        private function selectCommand(event:Event):void {
            trace("Selected command: " + event.target.label);
        }

        private function selectCommandMenu(event:Event):void {
            if (event.currentTarget.parent != null) {
                var menuItem:NativeMenuItem =
                        findItemForMenu(NativeMenu(event.currentTarget));
                if (menuItem != null) {
                    trace("Select event for \"" +
                            event.target.label +
                            "\" command handled by menu: " +
                            menuItem.label);
                }
            } else {
                trace("Select event for \"" +
                        event.target.label +
                        "\" command handled by root menu.");
            }
        }

        private function findItemForMenu(menu:NativeMenu):NativeMenuItem {
            for each (var item:NativeMenuItem in menu.parent.items) {
                if (item != null) {
                    if (item.submenu == menu) {
                        return item;
                    }
                }
            }
            return null;
        }
    }
}
```

Capítulo 37: Iconos de la barra de tareas en AIR

Adobe AIR 1.0 y posterior

Muchos sistemas operativos incluyen una barra de tareas (en Mac OS X, por ejemplo, el Dock) que pueden contener iconos para representar aplicaciones. Adobe® AIR® proporciona una interfaz para interactuar con los iconos de la barra de tareas de la aplicación mediante la propiedad `NativeApplication.nativeApplication.icon`.

- Using the system tray and dock icons (Iconos del Dock y de la bandeja del sistema, en inglés) (Flex)
- Using the system tray and dock icons (Iconos del dock y de la bandeja del sistema, en inglés) (Flash)

Más temas de ayuda

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

Iconos de la barra de tareas

Adobe AIR 1.0 y posterior

AIR crea el objeto `NativeApplication.nativeApplication.icon` automáticamente. En función del sistema operativo, el tipo de objeto es DockIcon o SystemTrayIcon. Es posible determinar cuál de estas subclases InteractiveIcon admitidas por AIR en el sistema operativo actual mediante las propiedades `NativeApplication.supportsDockIcon` y `NativeApplication.supportsSystemTrayIcon`. La clase base InteractiveIcon incluye las propiedades `width`, `height` y `bitmaps`. Estas propiedades se pueden utilizar para cambiar la imagen utilizada para el icono. Ahora bien, si accede a propiedades específicas de DockIcon o de SystemTrayIcon en el sistema operativo incorrecto se producirá un error de tiempo de ejecución.

Para establecer o cambiar la imagen utilizada para un icono, cree un conjunto que contenga una o varias imágenes y asígnelo a la propiedad `NativeApplication.nativeApplication.icon.bitmaps`. El tamaño de los iconos de la barra de tareas puede variar según el sistema operativo. Para que la imagen no se deteriore por el cambio de tamaño, puede añadir varios tamaños de imágenes al conjunto `bitmaps`. Si incluye más de una imagen, AIR selecciona el tamaño que más se acerque al tamaño real del icono en la barra de tareas y lo cambia de tamaño únicamente si es necesario. El siguiente ejemplo establece la imagen de un icono de la barra de tareas a partir de dos imágenes:

```
NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

Para cambiar la imagen del icono, asigne un conjunto con la nueva imagen o las nuevas imágenes a la propiedad `bitmaps`. Puede animar el icono haciendo que la imagen cambie en respuesta a un evento `enterFrame` o `timer`.

Para eliminar el icono del área de notificación de Windows y Linux o para restaurar el aspecto predeterminado del icono en Mac OS X, establezca `bitmaps` como un conjunto vacío:

```
NativeApplication.nativeApplication.icon.bitmaps = [];
```

Iconos del Dock

Adobe AIR 1.0 y posterior

AIR admite iconos del Dock si `NativeApplication.supportsDockIcon` es `true`. La propiedad `NativeApplication.nativeApplication.icon` representa el icono de la aplicación en el Dock (no el icono de la ventana).

Nota: *con AIR no es posible cambiar los iconos de ventanas en el Dock en Mac OS X. Asimismo, los cambios realizados en el icono del Dock de la aplicación solo se aplican cuando la aplicación se está ejecutando (el icono recupera su aspecto normal cuando se cierra la aplicación).*

Menús de iconos del Dock

Adobe AIR 1.0 y posterior

Es posible añadir comandos al menú estándar del Dock. Basta con crear un objeto NativeMenu que contenga los comandos y asignarlo a la propiedad `NativeApplication.nativeApplication.icon.menu`. Las opciones del menú se visualizarán encima de las opciones estándar del menú del icono del Dock.

Efecto de rebote de los iconos del Dock

Adobe AIR 1.0 y posterior

Puede hacer que el icono del Dock tenga efecto de rebote si llama al método `NativeApplication.nativeApplication.icon.bounce()`. Si establece el parámetro `bounce() priority` como informativo, el icono rebota una vez. Si se establece como crítico, el icono rebota hasta que el usuario activa la aplicación. Las constantes del parámetro `priority` se definen en la clase NotificationType.

Nota: *el icono no tiene efecto de rebote si la aplicación ya está activa.*

Eventos de iconos del Dock

Adobe AIR 1.0 y posterior

Cuando se hace clic en un icono del Dock, el objeto NativeApplication distribuye un evento `invoke`. Si la aplicación no está en ejecución, el sistema la inicia. En caso contrario, se entrega el evento `invoke` a la instancia de la aplicación en ejecución.

Iconos de la bandeja del sistema

Adobe AIR 1.0 y posterior

AIR admite iconos de bandeja del sistema si `NativeApplication.supportsSystemTrayIcon` es `true` (esto solo sucede en Windows y en la mayoría de distribuciones de Linux). En Windows y Linux, los iconos de la bandeja del sistema se visualizan en el área de notificación de la barra de tareas. De forma predeterminada, no aparece ningún icono. Para mostrar un icono, debe asignar un conjunto que contenga objetos BitmapData a la propiedad `bitmaps` del icono. Para cambiar la imagen del icono, asigne un conjunto con las nuevas imágenes a `bitmaps`. Para quitar el icono, establezca `bitmaps` como `null`.

Menús de los iconos de la bandeja del sistema

Adobe AIR 1.0 y posterior

Es posible añadir un menú al icono de la bandeja del sistema. Basta con crear un objeto NativeMenu y asignarlo a la propiedad `NativeApplication.nativeApplication.icon.menu` (el sistema operativo no proporciona ningún menú predeterminado). Para acceder al menú del icono de la bandeja del sistema, haga clic con el botón derecho sobre el icono.

Sugerencias de los iconos de la bandeja del sistema

Adobe AIR 1.0 y posterior

Puede añadir una sugerencia a un icono si establece la propiedad tooltip:

```
NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

Eventos de iconos de la bandeja del sistema

Adobe AIR 1.0 y posterior

El objeto SystemTrayIcon al que hace referencia la propiedad NativeApplication.nativeApplication.icon distribuye un evento ScreenMouseEvent en los eventos `click`, `mouseDown`, `mouseUp`, `rightClick`, `rightMouseDown` y `rightMouseUp`. Puede utilizar estos eventos, junto con el menú del icono, para permitir que los usuarios puedan interactuar con la aplicación cuando esta no tenga ninguna ventana visible.

Ejemplo: Creación de una aplicación sin ventanas

Adobe AIR 1.0 y posterior

El siguiente ejemplo crea una aplicación de AIR con un icono de bandeja del sistema pero sin ventanas visibles. (La propiedad `visible` de la aplicación no se debe establecer en `true` en el descriptor de la aplicación o la ventana estará visible cuando se inicie la aplicación.)

```
package
{
    import flash.display.Loader;
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.desktop.DockIcon;
    import flash.desktop.SystemTrayIcon;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.desktop.NativeApplication;

    public class SysTrayApp extends Sprite
    {
        public function SysTrayApp():void{
            NativeApplication.nativeApplication.autoExit = false;
            var icon:Loader = new Loader();
            var iconMenu:NativeMenu = new NativeMenu();
            var exitCommand:NativeMenuItem = iconMenu.addItem(new NativeMenuItem("Exit"));
```

```
                exitCommand.addEventListener(Event.SELECT, function(event:Event):void {
                    NativeApplication.nativeApplication.icon.bitmaps = [];
                    NativeApplication.nativeApplication.exit();
                });

            if (NativeApplication.supportsSystemTrayIcon) {
                NativeApplication.nativeApplication.autoExit = false;
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE, iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_16.png"));

                var systray:SystemTrayIcon =
                    NativeApplication.nativeApplication.icon as SystemTrayIcon;
                systray.tooltip = "AIR application";
                systray.menu = iconMenu;
            }

            if (NativeApplication.supportsDockIcon){
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE,iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_128.png"));
                var dock:DockIcon = NativeApplication.nativeApplication.icon as DockIcon;
                dock.menu = iconMenu;
            }
        }

        private function iconLoadComplete(event:Event):void
        {
            NativeApplication.nativeApplication.icon.bitmaps =
                [event.target.content.bitmapData];
        }
    }
}
```

Nota: *al utilizar el componente WindowedApplication de Flex, el atributo* `visible` *de la etiqueta WindowedApplication debe definirse como* `false`*. Este atributo sustituye a la configuración en el descriptor de la aplicación.*

Nota: *en el ejemplo se asume que existen dos archivos de imagen llamados* `AIRApp_16.png` *y* `AIRApp_128.png` *en un subdirectorio de la aplicación denominado* `icons`*. (El SDK de AIR contiene archivos de icono de ejemplos que puede copiar en su carpeta del proyecto.)*

Botones e iconos de la barra de tareas de la ventana

Adobe AIR 1.0 y posterior

Los iconos de las ventanas se suelen visualizar en un área de la ventana, denominada barra de tareas o Dock, para que los usuarios puedan acceder fácilmente a las ventanas en segundo plano o minimizadas. El Dock de Mac OS X muestra un icono de las aplicaciones así como uno para cada ventana minimizada. Las barras de tareas de Microsoft Windows y Linux muestran un botón con el icono del programa y el título de cada ventana de tipo normal de la aplicación.

Resaltar el botón de ventana de la barra de tareas

Adobe AIR 1.0 y posterior

Si una ventana está en segundo plano, puede avisar al usuario de la existencia de un evento de interés relacionado con la ventana. En Mac OS X, puede avisar al usuario con un efecto de rebote del icono en el Dock (tal como se describe en la sección "Efecto de rebote de los iconos del Dock" en la página 675). En Windows y Linux, puede resaltar el botón de la barra de tareas de la ventana llamando al método `notifyUser()` de la instancia de NativeWindow. El parámetro `type` transferido al método determina la urgencia de la notificación:

- `NotificationType.CRITICAL`: el icono de la ventana parpadea hasta que el usuario la pone en primer plano.
- `NotificationType.INFORMATIONAL`: el icono de la ventana se resalta cambiando de color.

  Nota: *en Linux, solo se admite el tipo informativo de notificación. La transmisión de cualquier valor de tipo a la función* `notifyUser()` *creará el mismo efecto.*

  La siguiente sentencia resalta el botón de la barra de tareas de una ventana:

```
stage.nativeWindow.notifyUser(NotificationType.CRITICAL);
```

  Si llama al método `NativeWindow.notifyUser()` en un sistema operativo no compatible con avisos de nivel de ventana, no producirá ningún efecto. Utilice la propiedad `NativeWindow.supportsNotification` para determinar si es compatible con la notificación de ventanas.

Creación de ventanas sin botones ni iconos de barra de tareas

Adobe AIR 1.0 y posterior

En el sistema operativo Windows, las ventanas creadas con los tipos *utility* o *lightweight* no aparecen en la barra de tareas. Las ventanas invisibles tampoco aparecen en la barra de tareas.

Dado que la ventana inicial siempre es de tipo *normal* para poder crear una aplicación sin ventanas en la barra de tareas, debe cerrar la ventana inicial o hacerla invisible. Para cerrar todas las ventanas de la aplicación sin salir del programa, establezca la propiedad `autoExit` del objeto NativeApplication como `false` antes de cerrar la última ventana. Si simplemente quiere evitar que la ventana principal sea visible, añada `<visible>false</visible>` al elemento `<initalWindow>` del archivo descriptor de la aplicación (y no establezca la propiedad `visible` como `true` ni llame al método `activate()` de la ventana).

En la ventanas nuevas abiertas por la aplicación, establezca la propiedad `type` del objeto NativeWindowInitOption transferido al constructor de la ventana como `NativeWindowType.UTILITY` o `NativeWindowType.LIGHTWEIGHT`.

En Mac OS X, las ventanas minimizadas se visualizan en el Dock. Si quiere evitar que el icono minimizado se visualice, oculte la ventana en vez de minimizarla. El siguiente ejemplo detecta un evento de cambio `nativeWindowDisplayState` y lo cancela y la ventana se está minimizando. En su lugar, el controlador establece la propiedad `visible` de la ventana como `false`:

```
private function preventMinimize(event:NativeWindowDisplayStateEvent):void{
    if(event.afterDisplayState == NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

Si una ventana está minimizada en el Dock de Mac OS X al establecer la propiedad `visible` como `false`, el icono del Dock no desaparece. El usuario puede seguir haciendo clic en el icono para volver a mostrar la ventana.

Capítulo 38: Trabajo con el sistema de archivos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash® Player ofrece capacidades básicas de lectura y escritura de archivos, a través de la clase FileReference. Por motivos de seguridad, cuando la ejecución se realice en Flash Player, el usuario siempre debe conceder permiso antes de que se pueda leer o escribir un archivo.

Adobe® AIR® proporciona un acceso más completo al sistema de archivos del equipo host que el que se ofrece en Flash Player. Al utilizar la API del sistema de archivos de AIR, se puede tener acceso a los directorios y archivos y gestionarlos, crear directorios y archivos, escribir datos en los archivos, etc.

Más temas de ayuda

flash.net.FileReference

flash.net.FileReferenceList

flash.filesystem.File

flash.filesystem.FileStream

Uso de la clase FileReference

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto FileReference representa un archivo de datos en un equipo cliente o servidor. Los métodos de la clase FileReference permiten que la aplicación cargue y guarde archivos localmente, y que pueda transferir datos de archivo en servidores remotos.

La clase FileReference proporciona dos enfoques distintos para cargar, transferir y guardar archivos de datos. Desde su introducción, la clase FileReference ha incluido los métodos `browse()`, `upload()` y`download()`. Utilice el método `browse()` para permitir que el usuario seleccione un archivo. Use el método `upload()` para transferir los datos de archivo a un servidor remoto. Emplee el método `download()` para recuperar esos datos del servidor y guardarlos en un archivo local. Al comenzar con Flash Player 10 y Adobe AIR 1.5, la clase FileReference incluye los métodos `load()` y `save()`. Asimismo, los métodos `load()` y `save()` permiten acceder y almacenar archivos locales directamente. El uso de dichos métodos es similar a los métodos del mismo nombre incluidos en las clases URLLoader y Loader.

Nota: *la clase File, que amplía la clase FileReference, junto con la clase FileStream proporcionan funciones adicionales para trabajar con archivos y el sistema de archivos local. Las clases File y FileStream solo se admiten en AIR y no en Flash Player.*

Clase FileReference

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cada objeto FileReference representa un único archivo de datos del equipo. Las propiedades de la clase FileReference contienen la siguiente información referente al archivo: tamaño, tipo, nombre, extensión, creador, fecha de creación y fecha de modificación.

Nota: *la propiedad* `creator` *solo se utiliza en Mac OS. En todas las demás plataformas se devuelve* `null`*.*

Nota: *la propiedad* `extension` *solo se admite en Adobe AIR.*

Se puede crear una instancia de la clase FileReference de una de estas dos formas:

- Con el operador `new`, como se puede ver en el siguiente código:

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- Llame al método `FileReferenceList.browse()`, que abre un cuadro de diálogo y solicita al usuario que seleccione uno o varios archivos para cargar. Seguidamente, crea un conjunto de objetos FileReference si el usuario selecciona correctamente uno o varios archivos.

Una vez creado el objeto FileReference, puede realizar lo siguiente:

- Llamar al método `FileReference.browse()`, que abre un cuadro de diálogo y solicita al usuario que seleccione un solo archivo del sistema de archivos locales. Esto se suele realizar antes de una llamada posterior al método `FileReference.upload()` o `FileReference.load()`. Llame al método `FileReference.upload()` para cargar el archivo en un servidor remoto. Llame al método `FileReference.load()` para abrir un archivo local.
- Llamada al método `FileReference.download()`. El método `download()` abre un cuadro de diálogo donde el usuario puede seleccionar una ubicación para guardar el nuevo archivo. Posteriormente, descarga datos del servidor y los guarda en el nuevo archivo.
- Llamada al método `FileReference.load()`. Este método comienza a cargar datos desde un archivo seleccionado previamente mediante el método `browse()`. No es posible llamar al método `load()` hasta que no finaliza la operación de `browse()` (el usuario selecciona un archivo).
- Llamada al método `FileReference.save()`. Este método abre un cuadro de diálogo y pide al usuario que seleccione una ubicación de archivo única en el sistema de archivos local. Seguidamente, guarda los datos en la ubicación especificada.

Nota: *solo se puede realizar una acción* `browse()`*,* `download()` *o* `save()` *a la vez, ya que no se pueden abrir varios cuadros de diálogo de forma simultánea.*

Las propiedades del objeto FileReference, como `name`, `size` o `modificationDate` no se definen hasta que se produce una de las situaciones siguientes:

- Se ha llamado al método `FileReference.browse()` o `FileReferenceList.browse()` y el usuario ha seleccionado un archivo desde el cuadro de diálogo.
- Se ha llamado al método `FileReference.download()` y el usuario ha especificado una nueva ubicación del archivo desde el cuadro de diálogo.

Nota: *cuando se realiza una descarga, solo la propiedad* `FileReference.name` *se llena antes de finalizar la descarga. Una vez finalizada la descarga, todas las propiedades están disponibles.*

Durante la llamada a los métodos `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` o `FileReference.save()`, la mayoría de reproductores continúan ejecutando el archivo SWF, incluidas la distribución de eventos y la ejecución del código.

Para las operaciones de carga y descarga, un archivo SWF solo puede acceder a archivos de su propio dominio, incluidos los dominios especificados en un archivo de política. Es preciso colocar un archivo de política en el servidor que contiene el archivo si dicho servidor no se encuentra en el mismo dominio que el archivo SWF que inicia la carga o la descarga.

Consulte FileReference.

Carga de datos desde archivos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `FileReference.load()` permite cargar datos en la memoria local desde un archivo local.

Nota: *el código debe llamar primero al método `FileReference.browse()` para que el usuario pueda seleccionar un archivo para cargarlo. Esta restricción no se aplica al contenido que se ejecuta en Adobe AIR en el entorno limitado de seguridad de la aplicación.*

El método `FileReference.load()` devuelve un valor inmediatamente después de la llamada, pero los datos que se cargan no están disponibles hasta un tiempo después. El objeto FileReference distribuye eventos para invocar métodos de detectores en cada paso del proceso de carga.

El objeto FileReference distribuye los siguientes eventos durante el proceso de carga.

- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de carga.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que se leen bytes de datos del archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de carga finaliza correctamente.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si falla el proceso de carga debido a un error de entrada/salida durante la apertura o la lectura de los datos en el archivo.

Cuando el objeto FileReference distribuye el evento complete, es posible acceder a los datos cargados como un elemento ByteArray en la propiedad `data` del objeto FileReference.

El siguiente ejemplo muestra cómo solicitar al usuario que seleccione un archivo y cargue en la memoria los datos de dicho archivo:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

El código de ejemplo primero crea el objeto FileReference llamado `fileRef` y, a continuación, llama a su método `browse()`. El método `browse()` abre un cuadro de diálogo que solicita al usuario que seleccione un archivo. Cuando un archivo está seleccionado, el código invoca al método `onFileSelected()`. Este método añade detectores para los eventos `progress` y `complete` y, seguidamente, llama al método `load()` del objeto FileReference. Los otros métodos de controladores del ejemplo simplemente producen mensajes que informan sobre el progreso de la operación de carga. Cuando finaliza la carga, la aplicación muestra el contenido del archivo cargado mediante el método `trace()`.

En Adobe AIR, la clase FileStream proporciona una funcionalidad adicional para leer datos desde un archivo local. Consulte "Lectura y escritura de archivos" en la página 719.

Guardar datos en archivos locales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `FileReference.save()` permite guardar datos en un archivo local. Comienza abriendo un cuadro de diálogo para que el usuario pueda introducir un nuevo nombre de archivo y una ubicación en la que guardarlo. Una vez seleccionado el nombre de archivo y la ubicación, los datos se escriben en el nuevo archivo. Una vez guardado correctamente el archivo, las propiedades del objeto FileReference se llenan con las propiedades del archivo local.

Nota: *el código solo debe llamar al método `FileReference.save()` como respuesta a un evento iniciado por el usuario (por ejemplo, un clic del ratón o la pulsación de una tecla). En caso contrario, se emite un error. Esta restricción no se aplica a contenido que se ejecuta en Adobe AIR en el entorno limitado de seguridad de la aplicación.*

El método `FileReference.save()` devuelve un valor inmediatamente después de recibir la llamada. Seguidamente, el objeto FileReference distribuye eventos para llamar a métodos de detección en cada paso del proceso grabación del archivo.

El objeto FileReference distribuye los siguientes eventos durante el proceso de grabación del archivo:

- Evento `select` (`Event.SELECT`): se distribuye cuando el usuario especifica la ubicación y el nombre de archivo del nuevo archivo que se va a guardar.
- Evento `cancel` (`Event.CANCEL`): se distribuye cuando el usuario hace clic en el botón Cancelar del cuadro de diálogo.
- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de grabación.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que los bytes de datos se guardan en el archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de grabación finaliza correctamente.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si el proceso de grabación falla debido a un error de entrada/salida al intentar guardar los datos en el archivo.

El tipo de objeto transferido en el parámetro `data` del método `FileReference.save()` determina el modo en el que se escriben los datos en el archivo:

- Si es un valor String, se guarda como un archivo de texto con codificación UTF-8.
- Si es un objeto XML, se escribe en un archivo en formato XML (conservando todo el formato).
- Si es un objeto Array, su contenido se escribe directamente en el archivo sin conversión alguna.
- Si es cualquier otro objeto, el método `FileReference.save()` llama al método `toString()` del objeto y guarda el valor String resultante en un archivo de texto UTF-8. Si no se puede llamar al método `toString()` del objeto, se emite un error.

Si el valor del parámetro `data` es `null`, se genera un error.

El siguiente código amplía el ejemplo anterior para el método `FileReference.load()`. Una vez leídos los datos del archivo, este ejemplo solicita al usuario que especifique un nombre de archivo y, a continuación, guarda los datos en un nuevo archivo:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }
        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.save(fileRef.data,"NewFileName.txt");
        }
```

```
        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

Si todos los datos se cargan desde el archivo, el código llama al método `onComplete()`. El método `onComplete()` elimina los detectores de los eventos de carga y seguidamente llama al método `saveFile()`. El método `saveFile()` llama a `FileReference.save()`. El método `FileReference.save()` abre un nuevo cuadro de diálogo para que el usuario pueda indicar un nuevo nombre de archivo y ubicación para guardarlo. Los métodos de detectores de eventos restantes realizan el seguimiento del progreso de grabación del archivo hasta que finaliza.

En Adobe AIR, la clase FileStream proporciona funcionalidad adicional para escribir datos en un archivo local. Consulte “Lectura y escritura de archivos” en la página 719.

Cargar archivos en un servidor

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para cargar archivos en un servidor, hay que llamar primero al método `browse()` para permitir que un usuario seleccione uno o varios archivos. A continuación, cuando se llama al método `FileReference.upload()`, el archivo seleccionado se transfiere al servidor. Si el usuario ha seleccionado varios archivos con el método `FileReferenceList.browse()`, Flash Player crea un conjunto de archivos seleccionados denominado `FileReferenceList.fileList`. A continuación, se puede utilizar el método `FileReference.upload()` para cargar cada archivo de forma individual.

Nota: *la utilización del método `FileReference.browse()` permite cargar únicamente archivos individuales. Para permitir que un usuario cargue varios archivos, es necesario utilizar el método `FileReferenceList.browse()`.*

De forma predeterminada, el cuadro de diálogo del selector de archivos del sistema permite a los usuarios seleccionar cualquier tipo de archivo del equipo local, aunque los desarrolladores pueden especificar uno o varios filtros personalizados de tipo de archivo. Para ello, deben utilizar la clase FileFilter y pasar un conjunto de instancias de filtro de archivos al método `browse()`:

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg;
*.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

Cuando el usuario ha seleccionado los archivos y ha hecho clic en el botón Abrir en el selector de archivos del sistema, se distribuye el evento `Event.SELECT`. Si se ha utilizado el método `FileReference.browse()` para seleccionar un archivo para cargarlo, es necesario utilizar el siguiente código para enviar el archivo a un servidor web:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Para enviar datos al servidor con el método `FileReference.upload()`, se pueden utilizar las propiedades `URLRequest.method` y `URLRequest.data` para enviar variables con los métodos `POST` o `GET`.

Cuando se intenta cargar un archivo con el método `FileReference.upload()`, se pueden distribuir los siguientes eventos:

- Evento `open` (`Event.OPEN`): se distribuye cuando se inicia la operación de carga.
- Evento `progress` (`ProgressEvent.PROGRESS`): se distribuye periódicamente a medida que se cargan los bytes de datos del archivo.
- Evento `complete` (`Event.COMPLETE`): se distribuye cuando la operación de carga finaliza correctamente.
- Evento `httpStatus` (`HTTPStatusEvent.HTTP_STATUS`): se distribuye cuando el proceso de carga falla debido a un error HTTP.
- Evento `httpResponseStatus` (`HTTPStatusEvent.HTTP_RESPONSE_STATUS`): se distribuye si una llamada a los métodos `upload()` o `uploadUnencoded()` intenta acceder a los datos mediante HTTP y Adobe AIR puede detectar y devolver el código de estado de la petición.
- Evento `securityError` (`SecurityErrorEvent.SECURITY_ERROR`): se distribuye cuando se produce un error en la carga debido a una infracción de la seguridad.
- Evento `uploadCompleteData` (`DataEvent.UPLOAD_COMPLETE_DATA`): se distribuye cuando se han recibido datos del servidor tras una carga correcta.
- Evento `ioError` (`IOErrorEvent.IO_ERROR`): se distribuye si falla el proceso de carga por cualquiera de los siguientes motivos:
  - Se produce un error de entrada/salida mientras Flash Player lee, escribe o transmite el archivo.
  - El archivo SWF intenta cargar un archivo en un servidor que requiere autenticación, por ejemplo, un nombre de usuario y una contraseña. Durante la carga, Flash Player no proporciona un medio para que los usuarios introduzcan contraseñas.
  - El parámetro `url` contiene un protocolo no válido. El método `FileReference.upload()` debe utilizar HTTP o HTTPS.

Flash Player no ofrece compatibilidad total con los servidores que requieren autenticación. Únicamente los archivos SWF que se ejecutan en un navegador y emplean el plugin de navegador o un control Microsoft ActiveX®, pueden mostrar un cuadro de diálogo que indique al usuario que introduzca un nombre de usuario y una contraseña para la autenticación y, además, solo para descargas. La transferencia de archivos no se producirá correctamente en las cargas que utilicen el plug-in o el control ActiveX, o en las cargas y descargas que utilicen el reproductor autónomo o externo.

Si se crea un script de servidor en ColdFusion para aceptar una carga de archivos desde Flash Player, se puede utilizar un código similar al siguiente:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Este código de ColdFusion carga el archivo enviado por Flash Player y lo guarda en el mismo directorio que la plantilla de ColdFusion, sobrescribiendo cualquier archivo con el mismo nombre. El código anterior muestra la cantidad mínima de código necesario para aceptar una carga de archivos; este script no debería utilizarse en un entorno de producción. Lo ideal sería añadir validación de datos para garantizar que los usuarios solo cargan tipos de archivos aceptados como, por ejemplo, una imagen, en lugar de un script de servidor potencialmente peligroso.

El siguiente código muestra cargas de archivos mediante PHP e incluye validación de datos. El script limita en 10 el número de archivos cargados en el directorio de carga, garantiza que el tamaño de archivo es inferior a 200 KB, y solo permite cargar y guardar archivos JPEG, GIF o PNG en el sistema de archivos.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
    move_uploaded_file($_FILES['Filedata']['tmp_name'],
"./temporary/".$_FILES['Filedata']['name']);
    $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
    if ($type == 1 || $type == 2 || $type == 3)
    {
        rename("./temporary/".$_FILES['Filedata']['name'],
"./images/".$_FILES['Filedata']['name']);
    }
    else
    {
        unlink("./temporary/".$_FILES['Filedata']['name']);
    }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
    array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
    $files_to_delete = array_splice($files, 0, count($files) - $MAXIMUM_FILE_COUNT);
    for ($i = 0; $i < count($files_to_delete); $i++)
    {
        unlink($files_to_delete[$i][0]);
    }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
    if ($a[1] == $b[1])
    {
        return 0;
    }
    else
    {
        return ($a[1] < $b[1]) ? -1 : 1;
    }
}
?>
```

Se pueden pasar variables adicionales al script de carga mediante el método de petición `POST` o `GET`. Para enviar variables adicionales de `POST` al script de carga, se puede utilizar el siguiente código:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

El ejemplo anterior crea un objeto URLVariables que se transmite al script del servidor remoto. En versiones anteriores de ActionScript, se podían pasar variables al script de carga del servidor pasando valores en la cadena de consulta. ActionScript 3.0 permite pasar variables al script remoto con un objeto URLRequest, que permite pasar datos mediante el método `POST` o `GET` que, a su vez, hace más sencillo pasar grandes conjuntos de datos. Para especificar si las variables se pasan con el método de petición `GET` o `POST`, se puede establecer la propiedad `URLRequest.method` en `URLRequestMethod.GET` o `URLRequestMethod.POST`, respectivamente.

ActionScript 3.0 también permite sustituir el nombre del campo de archivo de carga predeterminado, `Filedata`, proporcionando un segundo parámetro al método `upload()`, como se muestra en el ejemplo anterior (que sustituía el valor predeterminado `Filedata` por `Custom1`).

De forma predeterminada, Flash Player no intentará enviar una carga de prueba, aunque este valor predeterminado se puede omitir pasando un valor de `true` como tercer parámetro al método `upload()`. El objetivo de la carga de prueba es comprobar si la carga de archivos real se realizará correctamente, así como la autenticación del servidor, en caso de ser necesaria.

Nota: *actualmente, la carga de prueba solo puede realizarse en Flash Player basado en Windows.*

El script de servidor que gestiona la carga de archivos espera una petición HTTP `POST` con los siguientes elementos:

- `Content-Type` con un valor `multipart/form-data`.
- `Content-Disposition` con un atributo `name` establecido en "`Filedata`" y un atributo `filename` establecido en el nombre del archivo original. Se puede especificar un atributo `name` personalizado pasando un valor para el parámetro `uploadDataFieldName` en el método `FileReference.upload()`.
- El contenido binario del archivo.

A continuación se muestra un ejemplo de petición HTTP `POST`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

La siguiente petición de ejemplo HTTP `POST` envía tres variables `POST`: `api_sig`, `api_key` y `auth_token` y utiliza un valor de nombre de campo de datos de carga personalizado de `"photo"`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

Descarga de archivos de un servidor

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para permitir que los usuarios descarguen archivos de un servidor, se utiliza el método `FileReference.download()`, que utiliza dos parámetros `request` y `defaultFileName`. El primer parámetro es el objeto URLRequest que contiene el URL del archivo para descargar. El segundo parámetro es opcional y permite especificar un nombre de archivo predeterminado que aparece en el cuadro de diálogo del archivo descargado. Si se omite el segundo parámetro, `defaultFileName`, se utiliza el nombre de archivo del URL especificado.

El siguiente código descarga un archivo denominado index.xml desde el mismo directorio que el archivo SWF:

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Para establecer currentnews.xml como nombre predeterminado, en lugar de index.xml, es necesario especificar el parámetro `defaultFileName`, como se muestra en el siguiente fragmento de código:

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Cambiar el nombre de un archivo puede ser muy útil si el nombre de archivo del servidor no es intuitivo o ha sido generado por el servidor. También se recomienda especificar de forma explícita el parámetro `defaultFileName` cuando se descarga un archivo mediante un script de servidor, en lugar de descargarlo directamente. Por ejemplo, es necesario especificar el parámetro `defaultFileName` si se dispone de un script de servidor que descarga archivos específicos en función de las variables de URL que se le pasan. De lo contrario, el nombre predeterminado del archivo descargado es el nombre del script de servidor.

Es posible enviar datos al servidor mediante el método `download()` añadiendo parámetros al URL, para que los analice el script de servidor. El siguiente fragmento de código ActionScript 3.0 descarga un documento en función de los parámetros pasados a un script de ColdFusion:

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

El siguiente código muestra el script ColdFusion, download.cfm, que descarga uno de los dos archivos del servidor, dependiendo del valor de una variable de URL:

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

Clase FileReferenceList

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase FileReferenceList permite al usuario seleccionar uno o varios archivos para cargarlos en un script de servidor. La carga de archivos se controla mediante el método `FileReference.upload()`, al que es necesario llamar en cada archivo que selecciona el usuario.

El siguiente código crea dos objetos FileFilter (`imageFilter` y `textFilter`) y los pasa en un conjunto al método `FileReferenceList.browse()`. Como consecuencia, el cuadro de diálogo del archivo del sistema operativo muestra dos filtros de tipos de archivo posibles.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)",
"*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

Permitir al usuario seleccionar y cargar uno o varios archivos mediante la clase FileReferenceList equivale a utilizar `FileReference.browse()` para seleccionar archivos, aunque FileReferenceList permite seleccionar más de un archivo. Para cargar varios archivos es necesario actualizar cada uno de los archivos seleccionados mediante `FileReference.upload()`, como se muestra en el siguiente código:

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Dado que el evento `Event.COMPLETE` se añade a cada objeto FileReference individual del conjunto, Flash Player llama al método `completeHandler()` cuando finaliza la carga de cada uno de los archivos.

Uso de la API del sistema de archivos de AIR

Adobe AIR 1.0 y posterior

La API del sistema de archivos de Adobe AIR incluye las siguiente clases:

- Archivo
- FileMode
- FileStream

La API del sistema de archivos permite realizar las siguientes operaciones (entre otras):

- Copiar, crear, eliminar y mover archivos y directorios.
- Obtener información sobre archivos y directorios
- Leer y escribir archivos.

Aspectos básicos de los archivos de AIR

Adobe AIR 1.0 y posterior

Para obtener una explicación rápida y ejemplos de código del trabajo con el sistema de archivos en AIR, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Building a text-file editor (Creación de un editor de archivos de texto, en inglés - Flash)
- Building a text-file editor (Creación de un editor de archivos de texto, en inglés) (Flex)
- Building a directory search application (Creación de una aplicación de búsqueda de directorios, en inglés) (Flex)
- Reading and writing from an XML preferences file (Lectura y escritura desde un archivo XML de preferencias, en inglés) (Flex)
- Compressing files and data (Compresión de archivos y datos, en inglés) (Flex)

Adobe AIR pone a disposición clases que sirven para tener acceso a archivos y carpetas, así como para crearlos y gestionarlos. Estas clases, que están incluidas en el paquete flash.filesystem, se utilizan de la forma siguiente:

Clases de archivos	Descripción
File	El objeto File representa una ruta a un archivo o directorio. Los objetos de archivo sirven de puntero a un archivo o carpeta, iniciando la interacción con el archivo o la carpeta.
FileMode	La clase FileMode define las constantes de cadenas que se utilizan en el parámetro `fileMode` de los métodos `open()` y `openAsync()` de la clase FileStream. El parámetro `fileMode` de estos métodos determina las capacidades que están disponibles para el objeto FileStream una vez abierto el archivo, entre los cuales se encuentran la escritura, la lectura, el anexado y la actualización.
FileStream	El objeto FileStream se utiliza para abrir archivos para su lectura y escritura. Una vez creado un objeto File que señala un archivo nuevo o existente, se pasa ese puntero al objeto FileStream para poder abrir y leer o escribir datos.

Algunos métodos de la clase File tienen versiones tanto sincrónicas como asíncronas:

- `File.copyTo()` y `File.copyToAsync()`
- `File.deleteDirectory()` y `File.deleteDirectoryAsync()`
- `File.deleteFile()` y `File.deleteFileAsync()`
- `File.getDirectoryListing()` y `File.getDirectoryListingAsync()`
- `File.moveTo()` y `File.moveToAsync()`
- `File.moveToTrash()` y `File.moveToTrashAsync()`

Además, las opciones de FileStream funcionan de modo sincrónico o asíncrono, dependiendo de cómo abre el archivo el objeto FileStream: si llama al método `open()` o llama al método `openAsync()`.

Las versiones asíncronas permiten iniciar procesos que se ejecutan en segundo plano y distribuyen eventos cuando se han finalizado (o cuando se produce un evento de error). Puede ejecutarse otro tipo de código mientras tienen lugar estos procesos asíncronos en segundo plano. Con las versiones asíncronas de las operaciones, hay que configurar las funciones de detección de eventos empleando el método `addEventListener()` del objeto File o FileStream que llama a la función.

Las versiones sincrónicas permiten escribir código más sencillo que no depende de la configuración de funciones de detección de eventos. Sin embargo, dado que no puede ejecutarse otro código mientras se ejecuta el método sincrónico, pueden retrasar procesos importantes como la animación o la representación de objetos de visualización.

Trabajo con objetos File en AIR

Adobe AIR 1.0 y posterior

Un objeto File es un puntero a un archivo o directorio del sistema de archivos.

La clase File amplía la clase FileReference. La clase FileReference, que está disponible en Adobe® Flash® Player además de en AIR, representa un puntero a un archivo. La clase File añade propiedades y métodos que no se exponen en Flash Player (en un archivo SWF que se ejecuta en un navegador) por motivos de seguridad.

Clase File

Adobe AIR 1.0 y posterior

La clase File se emplea para lo siguiente:

- obtener la ruta a directorios especiales, entre ellos el directorio del usuario, el directorio de documentos del usuario, el directorio desde el cual se inició la aplicación, y el directorio de la aplicación;
- copiar archivos y directorios;
- mover archivos y directorios;
- eliminar archivos y directorios (o pasarlos a la papelera);
- enumerar los archivos y directorios que contiene un directorio;
- crear archivos y directorios temporales.

Una vez que un objeto File apunta a una ruta de archivo, se puede utilizar para leer y escribir datos de archivo usando la clase FileStream.

Un objeto File puede apuntar a la ruta de un archivo o directorio que aún no existe. Puede utilizarse un objeto File de este tipo al crear un archivo o directorio.

Rutas a objetos File

Adobe AIR 1.0 y posterior

Cada objeto File tiene dos propiedades que definen su ruta:

Propiedad	Descripción
`nativePath`	Especifica la ruta a un archivo en una plataforma determinada. Por ejemplo, una ruta en Windows podría ser "c:\Directorio de muestras\test.txt" mientras que en Mac OS sería "/Directorio de muestras/test.txt". En Windows, una propiedad `nativePath` utiliza la barra diagonal inversa (\) como carácter separador de directorios, mientras que en Mac OS y Linux utilizan la barra diagonal (/).
`url`	Esto puede utilizar el esquema de URL de archivo para apuntar a un archivo. Por ejemplo, una ruta en Windows podría ser "file:///c:/Directorio%20de%20muestras/test.txt" mientras que en Mac OS sería "file:///Directorio%20de%20muestras/test.txt". El motor de ejecución incluye otros esquemas de URL especiales además de `file` que se describen en "Esquemas de URL compatibles de AIR" en la página 705.

La clase File incluye propiedades estáticas para señalar a directorios estándar tanto en Mac OS, Windows y Linux. Entre estas propiedades se incluyen:

- `File.applicationStorageDirectory`: directorio de almacenamiento exclusivo para cada aplicación de AIR instalada.. Este directorio es un lugar adecuado para almacenar preferencias de usuario y recursos de la aplicación dinámicos. Considere en almacenamiento de grandes cantidades de datos en otro lugar.

  En Android e iOS, el directorio de almacenamiento de la aplicación se elimina cuando la aplicación se desinstala o el usuario opta por borrar los datos de la aplicación, pero este no es el caso en otras plataformas.

- `File.applicationDirectory`; directorio donde se instala la aplicación (junto con cualquier activo instalado). En algunos sistemas operativos, la aplicación se almacena en un solo archivo de paquete en lugar de en un directorio físico. En este caso, es posible que no se pueda acceder al contenido utilizando la ruta nativa. El directorio de la aplicación es de solo lectura.
- `File.desktopDirectory`: directorio del escritorio del usuario. Si una plataforma no define un directorio de escritorio, se utiliza otra ubicación del sistema de archivos.
- `File.documentsDirectory`: directorio de documentos del usuario. Si una plataforma no define un directorio de documentos, se utiliza otra ubicación del sistema de archivos.
- `File.userDirectory`: el directorio del usuario. Si una plataforma no define un directorio de usuario, se utiliza otra ubicación del sistema de archivos.

Nota: *si una plataforma no define ubicaciones estándar para los directorios de usuario, documentos o escritorio,* `File.documentsDirectory`*,* `File.desktopDirectory` *y* `File.userDirectory` *pueden hacer referencia al mismo directorio.*

Estas propiedades tienen valores diferentes en los distintos sistemas operativos. Por ejemplo, Mac y Windows cuentan con rutas nativas distintas en el directorio del escritorio del usuario. Sin embargo, la propiedad `File.desktopDirectory` señala a una ruta de directorio adecuada en todas las plataformas. Para escribir aplicaciones que funcionen bien en distintas plataformas, utilice estas propiedades como base para hacer referencia a otros directorios y archivos utilizados por la aplicación. Utilice el método `resolvePath()` para mejorar la ruta. Por ejemplo, el siguiente código señala al archivo preferences.xml en el directorio de almacenamiento de la aplicación:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

Aunque la clase File permite señalar a una ruta de archivo específica, esto puede implicar que las aplicaciones no funcionen en distintas plataformas. Por ejemplo, la ruta C:\Documents and Settings\joe\ solo funciona en Windows. Por estos motivos, es mejor utilizar las propiedades estáticas de la clase File como, por ejemplo, `File.documentsDirectory`.

Ubicaciones de directorio comunes

Plataforma	Tipo de directorio	Ubicación típica del sistema de archivos
Android	Aplicación	`/data/data/`
	Almacenamiento de la aplicación	`/data/data/air.applicationID/nombredearchivo/Local Store`
	Caché	/data/data/applicationID/cache
	Escritorio	`/mnt/sdcard`
	Documentos	`/mnt/sdcard`
	Temporal	`/data/data/applicationID/cache/FlashTmp.randomString`
	Usuario	`/mnt/sdcard`
iOS	Aplicación	`/var/mobile/Applications/uid/filename.app`
	Almacenamiento de la aplicación	`/var/mobile/Applications/uid/Library/Application Support/applicationID/Local Store`
	Caché	/var/mobile/Applications/uid/Library/Caches
	Escritorio	no accesible
	Documentos	`/var/mobile/Applications/uid/Documents`
	Temporal	`/private/var/mobile/Applications/uid/tmp/FlashTmpNNN`
	Usuario	no accesible
Linux	Aplicación	`/opt/filename/share`
	Almacenamiento de la aplicación	`/home/userName/.appdata/applicationID/Local Store`
	Escritorio	`/home/userName/Desktop`
	Documentos	`/home/userName/Documents`
	Temporal	`/tmp/FlashTmp.randomString`
	Usuario	`/home/userName`
Mac	Aplicación	`/Applications/filename.app/Contents/Resources`
	Almacenamiento de la aplicación	`/Users/`***`userName`***`/Library/Preferences/`***`applicationid`***`/Local Store` (AIR 3.2 y versiones anteriores) ***`ruta de acceso`***`/Library/Application Support/`***`applicationid`***`/Local Store` (AIR 3.3 y versiones posteriores), donde la ruta es `/Users/`***`userName`***`/Library/Containers/`***`bundle-id`***`/Data` (dentro de entorno limitado de seguridad) o `/Users/userName` (fuera de entorno limitado de seguridad)
	Caché	/Users/Nombre de usuario/Library/Caches
	Escritorio	`/Users/`***`userName`***`/Desktop`
	Documentos	`/Users/`***`userName`***`/Documents`
	Temporal	`/private/var/folders/JY/randomString/TemporaryItems/FlashTmp`
	Usuario	`/Users/`***`userName`***

Plataforma	Tipo de directorio	Ubicación típica del sistema de archivos
Windows	Aplicación	`C:\Program Files\filename`
	Almacenamiento de la aplicación	`C:\Documents and settings\userName\ApplicationData\applicationID\Local Store`
	Caché	C:\Documents and settings\nombredeusuario\Configuración local\Temp
	Escritorio	`C:\Documents and settings\userName\Desktop`
	Documentos	`C:\Documents and settings\userName\My Documents`
	Temporal	`C:\Documents and settings\userName\Local Settings\Temp\randomString.tmp`
	Usuario	`C:\Documents and settings\userName`

Las rutas nativas reales para estos directorios varían en función del sistema operativo y la configuración del equipo. Las rutas que se muestran en esta tabla representan ejemplos típicos. Siempre se deben utilizar las propiedades estáticas adecuadas de la clase File para hacer referencia a estos directorios de forma que la aplicación funcione correctamente en cualquier plataforma. En una aplicación real de AIR, los valores para `applicationID` y `filename` mostrados en la tabla proceden del descriptor de la aplicación. Si se especifica un ID de editor en el descriptor de la aplicación, este ID se añade al ID de la aplicación en estas rutas. El valor de `userName` es el nombre de la cuenta del usuario que realiza la instalación.

Configuración de un objeto File para que apunte a un directorio

Adobe AIR 1.0 y posterior

Existen distintas formas de configurar un objeto File para que apunte a un directorio.

Apuntar al directorio de inicio del usuario

Adobe AIR 1.0 y posterior

Un objeto File puede apuntar al directorio de inicio del usuario. El siguiente código configura un objeto File para que apunte a un subdirectorio AIR Test del directorio de inicio:

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

Apuntar al directorio de documentos del usuario

Adobe AIR 1.0 y posterior

Un objeto File puede apuntar al directorio de documentos del usuario. El siguiente código configura un objeto File para que apunte a un subdirectorio AIR Test del directorio de documentos:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test");
```

Apuntar al directorio del escritorio

Adobe AIR 1.0 y posterior

Un objeto File puede apuntar al escritorio. El siguiente código configura un objeto File para que apunte a un subdirectorio AIR Test del escritorio:

```
var file:File = File.desktopDirectory.resolvePath("AIR Test");
```

Apuntar al directorio de almacenamiento de la aplicación

Adobe AIR 1.0 y posterior

Un objeto File puede apuntar al directorio de almacenamiento de la aplicación. Para cada aplicación de AIR hay una ruta asociada exclusiva que define el directorio de almacenamiento de la aplicación. Este directorio es exclusivo para cada aplicación y usuario. Se puede utilizar este directorio para guardar datos que son específicos del usuario y la aplicación (como los archivos de datos de usuario o de preferencias). Por ejemplo, el siguiente código configura un objeto File para que apunte a un archivo de preferencias, prefs.xml, que se encuentra en el directorio de almacenamiento de la aplicación:

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

La ubicación del directorio de almacenamiento de la aplicación se suele basar en el nombre de usuario y el ID de la aplicación. Las siguientes ubicaciones del sistema de archivos se incluyen aquí para ayudar a depurar la aplicación. Siempre se debe usar la propiedad `File.applicationStorage` o `app-storage:` esquema de URI para resolver archivos en este directorio:

- En Mac OS: varía según la versión de AIR.

  AIR 3.2 y versiones anteriores: `/Users/nombre de usuario/Library/Preferences/`*`applicationID`*`/Local Store/`

  AIR 3.3 y versiones posteriores: *`ruta de acceso`*`/Library/Application Support/`*`applicationID`*`/Local Store`, donde *`ruta de acceso`* es `/Users/`*`username`*`/Library/Containers/`*`bundle-id`*`/Data` (dentro de entorno limitado de seguridad) o `/Users/`*`username`* (fuera de entorno limitado de seguridad)

  Por ejemplo (AIR 3.2):

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- En Windows, en el directorio Documents and Settings en:

  C:\Documents and Settings\user name`\Application Data\`*applicationID*`\Local Store\`

  Por ejemplo:

  ```
  C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
  ```

- En Linux—En:

  `/home/`*nombre de usuario*`/.appdata/`*applicationID*`/Local Store/`

  Por ejemplo:

  ```
  /home/babbage/.appdata/com.example.TestApp/Local Store
  ```

- En Android—en:

  /data/data/`androidPackageID`/`applicationID/Local Store`

  Por ejemplo:

  ```
  /data/data/air.com.example.TestApp/com.example.TestApp/Local Store
  ```

Nota: *si una aplicación dispone de ID de edición, este también se utiliza como parte de la ruta al directorio de almacenamiento de la aplicación.*

La URL (y la propiedad `url`) para un objeto File creado con `File.applicationStorageDirectory` utiliza el esquema de URL `app-storage` (consulte "Esquemas de URL compatibles de AIR" en la página 705), como en el ejemplo siguiente:

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
```

Apuntar al directorio de la aplicación

Adobe AIR 1.0 y posterior

Un objeto File puede apuntar al directorio en el que se instaló la aplicación, al que se describe como directorio de la aplicación. Para ello se utiliza propiedad `File.applicationDirectory`. Este directorio puede utilizarse para examinar el archivo descriptor de la aplicación u otros recursos que haya instalados con la aplicación. Por ejemplo, el siguiente código configura un objeto File para que apunte a un directorio llamado *images* que se encuentra en el directorio de la aplicación:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
```

La URL (y la propiedad `url`) para un objeto File creado con `File.applicationDirectory` utiliza el esquema de URL `app-storage` (consulte "Esquemas de URL compatibles de AIR" en la página 705), como en el ejemplo siguiente:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

Nota: *en Android, no se puede acceder a los archivos del paquete de la aplicación a través de* `nativePath`*. La propiedad* `nativePath` *es una cadena vacía. Utilice siempre la dirección URL para acceder a los archivos en el directorio de la aplicación en lugar de una ruta nativa.*

Apuntar al directorio de memoria caché

Adobe AIR 3.6 y posterior

Puede hacer que un objeto File apunte temporalmente al sistema operativo o al directorio de memoria caché mediante la propiedad `File.cacheDirectory`. Este directorio contiene archivos temporales que no son esenciales para ejecutar la aplicación, por lo que no causarán problemas ni pérdida de datos para el usuario si se eliminan.

En la mayoría de sistemas operativos, el directorio de memoria caché es temporal. En iOS, este directorio corresponde al directorio Caches de la biblioteca de aplicación. Si el espacio de almacenamiento disponible en el dispositivo es escaso, el sistema operativo podría eliminar los archivos de este directorio, de los que no se hacen copias de seguridad en el espacio de almacenamiento en línea. Para obtener más información, consulte "Control de copias de seguridad y almacenamiento en caché" en la página 706.

Apuntar a la raíz del sistema de archivos

Adobe AIR 1.0 y posterior

El método `File.getRootDirectories()` enumera todos los volúmenes raíz -como C: y los volúmenes montados- de un ordenador con Windows. En Mac OS y Linux, este método siempre produce como resultado el directorio raíz exclusivo del ordenador (el directorio "/"). El método `StorageVolumeInfo.getStorageVolumes()` proporciona más información detallada sobre los volúmenes de almacenamiento montados (consulte "Trabajo con volúmenes de almacenamiento" en la página 717).

Nota: *la raíz del sistema de archivos no se puede leer en Android. Se devuelve un objeto File que hace referencia al directorio con la ruta nativa, "/", pero las propiedades de ese objeto no tienen valores precisos. Por ejemplo,* `spaceAvailable` *siempre es 0.*

Apuntar a un directorio explícito

Adobe AIR 1.0 y posterior

Para que un objeto File apunte a un directorio explícito, se configura la propiedad `nativePath` del objeto File, como en el ejemplo siguiente (en Windows):

```
var file:File = new File();
file.nativePath = "C:\\AIR Test";
```

Importante: señalar a una ruta explícita de este modo puede implicar que el código no funcione en distintas plataformas. Por ejemplo, el ejemplo anterior solo funciona en Windows. Se pueden utilizar las propiedades estáticas del objeto File como, por ejemplo, `File.applicationStorageDirectory`, para localizar un directorio que funcione en varias plataformas. Utilice el método `resolvePath()` (consulte la sección siguiente) para buscar una ruta relativa.

Desplazarse a rutas relativas

Adobe AIR 1.0 y posterior

El método `resolvePath()` sirve para obtener una ruta relativa a otra ruta determinada. En el ejemplo siguiente, el código configura un objeto File para que apunte a un subdirectorio "AIR Test" del directorio de inicio del usuario:

```
var file:File = File.userDirectory;
file = file.resolvePath("AIR Test");
```

También se puede utilizar la propiedad `url` de un objeto File para que apunte a un directorio con base en una cadena URL, como en el ejemplo siguiente:

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

Para ver más información, consulte “Modificación de rutas de archivos” en la página 705.

Dejar que el usuario utilice la función Examinar para seleccionar un directorio

Adobe AIR 1.0 y posterior

La clase File incluye el método `browseForDirectory()`, que presenta un cuadro de diálogo del sistema que permite al usuario seleccionar un directorio para asignarlo al objeto. El método `browseForDirectory()` es asíncrono. El objeto File distribuye un evento `select` si el usuario selecciona un directorio y hace clic en el botón Abrir, o distribuye un evento `cancel` si el usuario hace clic en el botón Cancelar.

Por ejemplo, con el siguiente código el usuario puede seleccionar un directorio y se produce una ruta a ese directorio al seleccionarlo:

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(e:Event):void {
    trace(file.nativePath);
}
```

Nota: *en Android, no se admite el método `browseForDirectory()`. La llamada a este método no tiene efecto; un evento cancel se distribuye de forma inmediata. Para que los usuarios puedan seleccionar un directorio, utilice un cuadro de diálogo personalizado y definido por la aplicación.*

Apuntar al directorio desde el que se invocó la aplicación

Adobe AIR 1.0 y posterior

La ubicación del directorio desde el cual se invoca una aplicación puede obtenerse comprobando la propiedad `currentDirectory` del objeto InvokeEvent que se distribuye al invocar la aplicación. Para obtener más información, consulte "Captura de argumentos de la línea de comandos" en la página 913.

Configuración de un objeto File para que apunte a un archivo

Adobe AIR 1.0 y posterior

Existen distintas formas de configurar el archivo al que apunta un objeto File.

Apuntar a una ruta de archivo explícita

Adobe AIR 1.0 y posterior

Importante: señalar a una ruta explícita puede implicar que el código no funcione en distintas plataformas. Por ejemplo, la ruta C:/foo.txt solo funciona en Windows. Se pueden utilizar las propiedades estáticas del objeto File como, por ejemplo, `File.applicationStorageDirectory`, para localizar un directorio que funcione en varias plataformas. Utilice el método `resolvePath()` (consulte "Modificación de rutas de archivos" en la página 705) para buscar una ruta relativa.

Se puede utilizar la propiedad `url` de un objeto File para que apunte a un archivo o un directorio con base en una cadena URL, como en el ejemplo siguiente:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

También se puede pasar la URL a la función constructora `File()`, como en el ejemplo siguiente:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

La propiedad `url` siempre produce la versión codificada en formato URI de la URL (por ejemplo, los espacios en blanco se sustituyen con `"%20"`):

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

Se puede utilizar también la propiedad `nativePath` de un objeto File para definir una ruta explícita. Por ejemplo, cuando se ejecuta en un ordenador con Windows, el código que aparece a continuación configura un objeto File para que apunte al archivo test.txt en el subdirectorio AIR Test de la unidad C:.

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

También se puede pasar esta ruta a la función constructora `File()`, como en el ejemplo siguiente:

```
var file:File = new File("C:/AIR Test/test.txt");
```

Utilice el carácter de barra diagonal (/) como delimitador para la propiedad `nativePath`. En Windows, también puede utilizar el carácter de barra diagonal inversa (\), pero esto implica que las aplicaciones no funcionen en distintas plataformas.

Para ver más información, consulte "Modificación de rutas de archivos" en la página 705.

Enumeración de los archivos de un directorio

Adobe AIR 1.0 y posterior

El método `getDirectoryListing()` de un objeto File sirve para obtener un conjunto de objetos File que apuntan a archivos y subdirectorios en el nivel raíz de un directorio. Para obtener más información, consulte "Enumeración de directorios" en la página 712.

Dejar que el usuario utilice la función Examinar para seleccionar un archivo

Adobe AIR 1.0 y posterior

La clase File incluye los siguientes métodos que presentan un cuadro de diálogo del sistema que permite al usuario seleccionar un archivo para asignarlo al objeto:

- `browseForOpen()`
- `browseForSave()`
- `browseForOpenMultiple()`

Todos estos métodos son asíncronos. Los métodos `browseForOpen()` y `browseForSave()` distribuyen el evento "select" cuando el usuario selecciona un archivo (o una ruta de destino, en el caso de browseForSave()). Con los métodos `browseForOpen()` y `browseForSave()`, al seleccionarlo el objeto File de destino apunta a los archivos seleccionados. El método `browseForOpenMultiple()` distribuye un evento `selectMultiple` cuando el usuario selecciona varios archivos. El evento `selectMultiple` es del tipo FileListEvent, el cual tiene una propiedad `files` que es un conjunto de objetos File (que apuntan a los archivos seleccionados).

Por ejemplo, el siguiente código presenta al usuario un cuadro de diálogo "Abrir" que le permite seleccionar un archivo:

```
 var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

Si en la aplicación ya estaba abierto otro cuadro de diálogo con la función examinar cuando se llama a un método de examinar, el motor de ejecución emite una excepción de Error.

Nota: *en Android, solo se pueden seleccionar los archivos de audio, vídeo e imagen con los métodos* `browseForOpen()` *y* `browseForOpenMultiple()`*. El cuadro de diálogo browseForSave() también muestra únicamente los archivos de medios aunque el usuario pueda introducir un nombre de archivo arbitrario. Para abrir y guardar los archivos que no son de medios, se debe tener en cuenta el uso de cuadros de diálogo personalizados en lugar de estos métodos.*

Modificación de rutas de archivos

Adobe AIR 1.0 y posterior

Se puede también modificar la ruta de un objeto File existente llamando al método `resolvePath()` o modificando la propiedad `nativePath` o `url` del objeto, como en los ejemplos siguientes (en Windows):

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("..");
trace(file2.nativePath); // C:\Documents and Settings\userName
var file3:File = File.documentsDirectory;
file3.nativePath += "/subdirectory";
trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4:File = new File();
file4.url = "file:///c:/AIR Test/test.txt";
trace(file4.nativePath); // C:\AIR Test\test.txt
```

Al usar la propiedad `nativePath`, utilice el carácter de barra diagonal (/) como carácter separador de directorios. En Windows, también se puede emplear el carácter de barra diagonal inversa (\), pero no es recomendable, ya que implica que el código no funcione en distintas plataformas.

Esquemas de URL compatibles de AIR

Adobe AIR 1.0 y posterior

En AIR, Al definir la propiedad `url` de un objeto File se puede utilizar cualquiera de los esquemas de URL siguientes:

Esquema de URL	Descripción
file	Se utiliza para especificar una ruta relativa a la raíz del sistema de archivos. Por ejemplo: `file:///c:/AIR Test/test.txt` La norma para URL especifica que la URL tipo "file" debe tener el formato `file:///<host>/<ruta>`. Como caso especial, `<host>` puede ser la cadena vacía, que se interpreta como "la máquina desde la cual se interpreta la URL". Esta es la razón por la cual las URL del tipo "file" a menudo tienen tres barras diagonales (///).
app	Se utiliza para especificar una ruta relativa al directorio raíz de la aplicación instalada (el directorio que contiene el archivo aplicacion.xml para la aplicación instalada). Por ejemplo, la ruta siguiente apunta a un subdirectorio de imágenes del directorio de la aplicación instalada: `app:/images`
app-storage	Se utiliza para especificar una ruta relativa al directorio de almacenamiento de la aplicación. AIR define un directorio de almacenamiento exclusivo para cada una de las aplicaciones instaladas, lo cual proporciona un lugar útil para guardar datos que son específicos para esa aplicación. Por ejemplo, la siguiente ruta apunta al archivo "prefs.xml" en el subdirectorio "settings" del directorio de almacenamiento de la aplicación: `app-storage:/settings/prefs.xml`

Control de copias de seguridad y almacenamiento en caché

Adobe AIR 3.6 y posterior, iOS y OS X únicamente

En ciertos sistemas operativos, especialmente iOS y Mac OS X, se pueden crear automáticamente copias de seguridad de archivos de aplicación en un almacenamiento remoto. Además, en iOS existen restricciones de copia de seguridad aplicables de algunos archivos, así como restricciones sobre la ubicación en la que guardar archivos según su finalidad.

A continuación se resumen las directrices de Apple sobre copia de seguridad y almacenamiento. En secciones posteriores encontrará más información al respecto.

- Para especificar que no se requiere una copia de seguridad de un archivo y (solo iOS) que el sistema operativo puede eliminarlo si hace falta liberar espacio de almacenamiento en el dispositivo, guarde el archivo en el directorio de memoria caché (`File.cacheDirectory`). Esta es la ubicación preferida en iOS y debería utilizarse para la mayoría de archivos que pueden volver a generarse o cargarse.
- Para especificar que no se requiere una copia de seguridad de un archivo pero el sistema operativo no debería eliminarlo, guárdelo en uno de los directorios de bibliotecas de la aplicación (por ejemplo, `File.applicationStorageDirectory`) o en el directorio de documentos (`File.documentsDirectory`). Establezca la propiedad de objeto File `preventBackup` en `true`. Apple lo exige así para contenido que puede volver a generarse o cargarse y es necesario para el funcionamiento correcto de la aplicación cuando se usa sin conexión.

Archivos que requieren copia de seguridad

A fin de ahorrar espacio de copia de seguridad y reducir el uso de ancho de banda de red, las directrices de Apple para aplicaciones de iOS y Mac especifican que solo se deberían hacer copias de archivos que contengan datos introducidos por usuarios o información que no se pueda volver a generar o cargar.

De forma predeterminada se crean copias de seguridad de todos los archivos que haya en las carpetas de bibliotecas de la aplicación. En Mac OS X se encuentran en el directorio de almacenamiento de la aplicación. En iOS, esto incluye el directorio de almacenamiento de la aplicación, el directorio de la aplicación, el directorio del escritorio y el del usuario (puesto que dichos directorios están asignados a las carpetas de bibliotecas de aplicación en iOS). Por lo tanto, se hacen copias de seguridad de todos los archivos contenidos en esos directorios, y se guardan en el almacenamiento del servidor de forma predeterminada.

Si guarda en una de estas ubicaciones un archivo que puede volver a generarse en la aplicación, debería marcarlo para indicar al sistema operativo que no haga una copia de seguridad del mismo. Para indicar que un archivo no debe tener una copia de seguridad, establezca la propiedad de objeto File `preventBackup` en `true`.

En iOS, aunque un archivo en una de las carpetas de bibliotecas de aplicación tenga la propiedad `preventBackup` establecida en `true`, se marca como archivo persistente que no debería ser eliminado por el sistema operativo.

Control de eliminación de archivos y almacenamiento en memoria caché

Las directrices de Apple para aplicaciones de iOS especifican que, de ser posible, el contenido regenerable debería poder ser eliminado por el sistema operativo si hiciera falta liberar espacio de almacenamiento en el dispositivo.

En iOS, los archivos ubicados en las carpetas de bibliotecas de aplicación (por ejemplo, en el directorio de almacenamiento de la aplicación o en el de documentos) se marcan como permanentes y el sistema operativo no los elimina.

Guarde en el directorio de memoria caché de la aplicación archivos que la aplicación puede volver a crear y pueden eliminarse en caso de faltar espacio de almacenamiento. Para acceder al directorio de memoria caché se utiliza la propiedad estática `File.cacheDirectory`.

En iOS, el directorio correspondiente es el directorio de caché de la aplicación (<Application Home>/Library/Caches). En otros sistemas operativos, este directorio se asigna a uno equivalente. Por ejemplo, en Mac OS X también se asigna al directorio Caches en la biblioteca de aplicación. En Android, el directorio de memoria caché se asigna al directorio de caché de la aplicación. En Windows, el directorio de memoria caché se asigna al directorio temp del sistema operativo. Tanto en Android como en Windows, el directorio es accesible mediante una llamada a los métodos de clase File `createTempDirectory()` y `createTempFile()`.

Búsqueda de la ruta relativa entre dos archivos

Adobe AIR 1.0 y posterior

El método `getRelativePath()` sirve para buscar la ruta relativa entre dos archivos:

```
var file1:File = File.documentsDirectory.resolvePath("AIR Test");
var file2:File = File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt
```

El segundo parámetro del método `getRelativePath()`, el parámetro `useDotDot`, permite que se incluya la sintaxis `..` en los resultados para indicar directorios superiores:

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3:File = File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..
trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

Versiones canónicas de nombres de archivo

Adobe AIR 1.0 y posterior

En los nombres de ruta y archivo no se distingue entre mayúsculas y minúsculas en Windows y Mac OS. En el siguiente ejemplo, dos objetos File apuntan al mismo archivo:

```
File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

No obstante, los documentos y nombres de directorio sí incluyen mayúsculas. En los siguientes ejemplos se da por sentado que existe una carpeta llamada AIR Test en el directorio de documentos:

```
var file:File = File.documentsDirectory.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
trace(file.nativePath); // ... AIR Test
```

El método `canonicalize()` convierte el objeto `nativePath` para que figuren correctamente las mayúsculas en el nombre del archivo o directorio. En los sistemas de archivos que distinguen entre mayúsculas y minúsculas (como Linux), cuando existen varios archivos con nombres que difieren únicamente en las mayúsculas y minúsculas, el método `canonicalize()` ajusta la ruta para que coincida con el primer archivo encontrado (en un orden que determina el sistema de archivos).

El método `canonicalize()` también sirve para convertir nombres de archivo cortos (nombres "8.3") en nombres de archivo largos en Windows, como en los ejemplos siguientes:

```
var path:File = new File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
trace(path.nativePath); // C:\AIR Test
```

Trabajo con paquetes y vínculos simbólicos

Adobe AIR 1.0 y posterior

Diversos sistemas operativos admiten archivos de paquetes y archivos de vínculos simbólicos:

Paquetes En Mac OS, los directorios pueden designarse como paquetes y aparecer en el Finder de Mac OS archivo sencillo en lugar de como directorio.

Vínculos simbólicos Mac OS, Linux y Windows Vista admiten este tipo de vínculos. Los vínculos simbólicos permiten que un archivo apunte a otro archivo o directorio del disco. Si bien son similares, los vínculos simbólicos no son lo mismo que los alias. Un alias siempre se notifica como archivo (y no como directorio) y la lectura o escritura en un alias o acceso directo no afecta nunca el archivo o directorio original al que apunta. Por otro lado, un vínculo simbólico se comporta exactamente igual que el archivo o directorio al que apunta. Se puede notificar como archivo o directorio, y la lectura o escritura en un vínculo simbólico afecta al archivo o directorio al que apunta y no al propio vínculo simbólico. Asimismo, en Windows la propiedad `isSymbolicLink` para un objeto File que hace referencia a un punto de unión (utilizado en el sistema de archivos NTFS) se establece en `true`.

La clase File incluye las propiedades `isPackage` y `isSymbolicLink` para comprobar si un objeto File remite a un paquete o a un vínculo simbólico.

El código siguiente itera a través del directorio del escritorio del usuario, enumerando los subdirectorios que *no* son paquetes:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !desktopNodes[i].isPackage)
    {
        trace(desktopNodes[i].name);
    }
}
```

El código siguiente itera a través del directorio del escritorio del usuario, enumerando los archivos y directorios que *no* son vínculos simbólicos:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        trace(desktopNodes[i].name);
    }
}
```

El método `canonicalize()` modifica la ruta de un vínculo simbólico para apuntar al archivo o directorio al que se refiere el vínculo. El código siguiente itera a través del directorio del escritorio del usuario y notifica las rutas referenciadas por archivos que son vínculos simbólicos:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode:File = desktopNodes[i] as File;
        linkNode.canonicalize();
        trace(linkNode.nativePath);
    }
}
```

Determinación del espacio disponible en un volumen

Adobe AIR 1.0 y posterior

La propiedad `spaceAvailable` de un objeto File es el espacio (expresado en bytes) que está disponible para ser utilizado en el lugar donde se encuentra el archivo. Por ejemplo, el código siguiente comprueba el espacio disponible en el directorio de almacenamiento de la aplicación:

```
trace(File.applicationStorageDirectory.spaceAvailable);
```

Si el objeto File remite a un directorio, la propiedad `spaceAvailable` indica el espacio en el directorio que utilizan los archivos. Si el objeto File remite a un archivo, la propiedad `spaceAvailable` indica el espacio en el que podría ampliarse el archivo. Si no existe la ubicación de archivo, la propiedad `spaceAvailable` se define en 0. Si el objeto File remite a un vínculo simbólico, la propiedad `spaceAvailable` se define en el espacio disponible en el lugar al que apunta el vínculo simbólico.

El espacio disponible para un directorio o archivo suele ser el mismo que el que está disponible en el volumen que contiene el directorio o el archivo. No obstante, el espacio disponible puede ajustarse teniendo en cuenta los cupos y límites por directorio.

Para añadir un archivo o directorio a un volumen se suele necesitar más espacio que el tamaño mismo del archivo o del contenido del directorio. Por ejemplo, el sistema operativo puede necesitar más espacio para guardar la información del índice. O los sectores del disco que se requieren pueden usar espacio adicional. Además, el espacio que hay disponible cambia constantemente. Por todo ello es que no se puede asignar todo el espacio notificado a guardar los archivos. Para ver más información sobre la escritura en el sistema de archivos, consulte "Lectura y escritura de archivos" en la página 719.

El método `StorageVolumeInfo.getStorageVolumes()` proporciona más información detallada sobre los volúmenes de almacenamiento montados (consulte "Trabajo con volúmenes de almacenamiento" en la página 717).

Apertura de archivos con la aplicación del sistema predeterminada

Adobe AIR 2 y posterior

En AIR 2, es posible abrir un archivo utilizando la aplicación registrada por el sistema operativo para abrirla. Por ejemplo, una aplicación de AIR puede abrir un archivo DOC con la aplicación resgitrada para abrirlo. Utilice el método `openWithDefaultApplication()` de un objeto File para abrir el archivo. Por ejemplo, el siguiente código abre un archivo denominado test.doc en el escritorio del usuario y lo abre con la aplicación predeterminada para los archivos DOC:

```
var file:File = File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

Nota: *en Linux, el tipo MIME del archivo (no la extensión del nombre de archivo) determina la aplicación predeterminada para un archivo.*

El siguiente código permite al usuario buscar un archivo mp3 y abrirlo en la aplicación predeterminada para reproducir archivos mp3:

```
var file:File = File.documentsDirectory;
var mp3Filter:FileFilter = new FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);

function fileSelected(e:Event):void
{
    file.openWithDefaultApplication();
}
```

El método `openWithDefaultApplication()` no se puede utilizar con archivos ubicados en el directorio de la aplicación.

AIR impide el uso del método `openWithDefaultApplication()` para abrir determinados archivos. En Windows, AIR impide la apertura de archivos de determinados tipos como, por ejemplo, EXE o BAT. En Mac OS y en Linux, AIR impide abrir archivos que inician una aplicación específica. (Estos incluyen Terminal y AppletLauncher en Mac OS; y csh, bash o ruby en Linux.) Si se intenta abrir uno de estos archivos con el método `openWithDefaultApplication()`, se generará una excepción. Para obtener una lista de los tipos de archivo con limitaciones, consulte la entrada de la referencia del lenguaje del método `File.openWithDefaultApplication()`.

Nota: *esta restricción no existe en una aplicación de AIR instalada con un programa de instalación nativo (una aplicación de escritorio ampliada).*

Obtención de información sobre el sistema de archivos

Adobe AIR 1.0 y posterior

La clase File incluye las siguientes propiedades fijas que proporcionan información útil sobre el sistema de archivos:

Propiedad	Descripción
`File.lineEnding`	La secuencia de caracteres de final de línea que utiliza el sistema operativo del ordenador host. En Mac OS y Linux es el carácter de salto de línea. En Windows es el carácter de retorno de carro seguido del carácter de salto de línea.
`File.separator`	El carácter separador de componentes de rutas en el sistema operativo del ordenador host. En Mac OS y Linux es el carácter de barra diagonal (/). En Windows es el carácter de barra diagonal inversa (\).
`File.systemCharset`	El código predeterminado que utiliza para los archivos el sistema operativo del ordenador host. Esto corresponde al lenguaje del juego de caracteres que utiliza el sistema operativo.

La clase `Capabilities` incluye también información del sistema que puede resultar de utilidad al trabajar con archivos:

Propiedad	Descripción
Capabilities.hasIME	Especifica si el reproductor se ejecuta en un sistema que tiene (`true`) o no tiene (`false`) un editor de método de entrada (IME) instalado.
Capabilities.language	Especifica el código de idioma del sistema en el que se está ejecutando el reproductor.
Capabilities.os	Especifica el sistema operativo actual.

Nota: *tenga cuidado al utilizar `Capabilities.os` para determinar características del sistema. Si existe una propiedad más específica para determinar una característica del sistema, utilícela. De lo contrario, corre el riesgo de escribir código que no funcione correctamente en todas las plataformas. Por ejemplo, considérese el fragmento de código siguiente:*

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

Este código implica problemas en Linux. Es mejor utilizar simplemente la propiedad `File.separator`.

Trabajo con directorios

Adobe AIR 1.0 y posterior

El motor de ejecución ofrece funciones para trabajar con directorios en el sistema de archivos local.

Para obtener más información sobre la creación de objetos File que apuntan a directorios, consulte "Configuración de un objeto File para que apunte a un directorio" en la página 699.

Creación de directorios

Adobe AIR 1.0 y posterior

El método `File.createDirectory()` sirve para crear un directorio. En el ejemplo siguiente, el código crea un directorio llamado "AIR Test" como subdirectorio del directorio de inicio del usuario:

```
var dir:File = File.userDirectory.resolvePath("AIR Test");
dir.createDirectory();
```

Si el directorio ya existe, el método `createDirectory()` no hace nada.

Además, en algunos modos un objeto FileStream crea un directorio cuando se abre un archivo. Se crean directorios inexistentes cuando se concreta una instancia de FileStream con el parámetro `fileMode` del constructor `FileStream()` definido en `FileMode.APPEND` o `FileMode.WRITE`. Para ver más información, consulte "Flujo de trabajo de lectura y escritura de archivos" en la página 719.

Creación de directorios temporales

Adobe AIR 1.0 y posterior

La clase File incluye un método `createTempDirectory()` que crea un directorio en la carpeta de directorios temporales para el sistema, como en el ejemplo siguiente:

```
var temp:File = File.createTempDirectory();
```

El método `createTempDirectory()` crea automáticamente un directorio temporal exclusivo (ahorrándole el trabajo de determinar un nuevo lugar exclusivo).

Se puede utilizar un directorio temporal para guardar de forma provisional los archivos temporales que se utilizan para una sesión de la aplicación. Cabe observar que existe un método `createTempFile()` para crear nuevos archivos temporales exclusivos en el directorio temporal del sistema.

Puede que desee eliminar el directorio temporal antes de cerrar la aplicación, dado que *no* se elimina de forma automática en todos los dispositivos.

Enumeración de directorios

Adobe AIR 1.0 y posterior

El método `getDirectoryListing()` o el método `getDirectoryListingAsync()` de un objeto File sirve para obtener un conjunto de objetos File que apuntan a archivos y subdirectorios de un directorio.

Por ejemplo, el siguiente código enumera el contenido del directorio de documentos del usuario (sin examinar los subdirectorios):

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
    trace(contents[i].name, contents[i].size);
}
```

Al utilizar la versión asíncrona del método, el objeto de evento `directoryListing` tiene una propiedad `files` que es el conjunto de objetos File que corresponden a los directorios:

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

Copia y movimiento de directorios

Adobe AIR 1.0 y posterior

Los directorios se copian o mueven de la misma forma que los archivos. Por ejemplo, el siguiente código copia un directorio de modo sincrónico:

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

Si se especifica "true" para el parámetro `overwrite` del método `copyTo()`, se eliminan todos los archivos y las carpetas de un directorio de destino existente, sustituyéndose por los archivos y carpetas del directorio de origen (aunque el archivo de destino no exista en el directorio de origen).

El directorio que se especifique como parámetro `newLocation` del método `copyTo()` especifica la ruta al directorio resultante; *no* especifica el directorio *superior* que contendrá el directorio resultante.

Para obtener más información, consulte "Copia y movimiento de directorios" en la página 715.

Eliminación del contenido de los directorios

Adobe AIR 1.0 y posterior

La clase File incluye un método `deleteDirectory()` y un método `deleteDirectoryAsync()`. Estos métodos eliminan directorios, el primero de forma sincrónica y el segundo de forma asíncrona (consulte "Aspectos básicos de los archivos de AIR" en la página 695). Ambos métodos incluyen un parámetro `deleteDirectoryContents` (que tiene un valor booleano); cuando dicho parámetro se define en `true` (el valor predeterminado es `false`), al llamar al método se eliminan los directorios que no estén vacíos; de lo contrario, solo se eliminan los directorios vacíos.

En el ejemplo siguiente, el código elimina de modo sincrónico el subdirectorio AIR Test del directorio de documentos del usuario:

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

El código siguiente elimina de modo asíncrono el subdirectorio AIR Test del directorio de documentos del usuario:

```
 var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

También se incluyen los métodos `moveToTrash()` y `moveToTrashAsync()`, que sirven para trasladar un directorio a la papelera del sistema. Para obtener más información, consulte "Traslado de archivos a la papelera" en la página 716.

Trabajo con archivos

Adobe AIR 1.0 y posterior

La API de archivos de AIR permite añadir capacidades básicas de interacción de archivos a las aplicaciones. Por ejemplo, se pueden leer y escribir archivos, copiar y eliminar archivos, etc. Dado que las aplicaciones tienen acceso al sistema de archivos local, consulte "Seguridad en AIR" en la página 1122 si aún no lo ha hecho.

Nota: *se puede asociar un tipo de archivo con una aplicación de AIR (de modo que, al hacerle doble clic, se abra la aplicación). Para obtener más información, consulte "Gestión de asociaciones con archivos" en la página 921.*

Obtención de información sobre los archivos

Adobe AIR 1.0 y posterior

La clase File incluye las siguientes propiedades que brindan información sobre un archivo o directorio al que apunta un objeto File:

Propiedad File	Descripción
creationDate	La fecha de creación del archivo en el disco local.
creator	Obsoleto: utilice la propiedad `extension`. (Esta propiedad notifica el tipo de creador Macintosh del archivo, que solo se utiliza en las versiones de Mac OS anteriores a Mac OS X).
downloaded	(AIR 2 y posterior) Indica si el archivo o directorio al que se hace referencia se ha descargado (de Internet) o no. La propiedad solo es significativa en sistemas operativos en los que se pueden identificar archivos descargados: • Windows XP Service Pack 2 y posterior, y en Windows Vista • Mac OS 10.5 y versiones posteriores
exists	Si existe o no el archivo o directorio al que se remite.
extension	La extensión del archivo, que es la parte del nombre después del punto final ("."), sin incluir este. Si el nombre de archivo no contiene un punto, la extensión es `null`.
icono	Un objeto Icon que contiene los iconos definidos para el archivo.
isDirectory	Si el objeto File remite o no a un directorio.
modificationDate	La fecha de la última modificación del archivo o directorio del disco local.
name	El nombre del archivo o directorio (incluida la extensión, si la hay) en el disco duro,
nativePath	El trayecto completo en la representación del sistema operativo del ordenador host. Consulte "Rutas a objetos File" en la página 696.

Propiedad File	Descripción
parent	La carpeta que contiene la carpeta o el archivo que representa el objeto File. Esta propiedad es `null` si el objeto File remite a un archivo o directorio que se encuentra en la raíz del sistema de archivos.
size	El tamaño del archivo en el disco local, expresado en bytes.
type	Obsoleto: utilice la propiedad `extension`. (En un ordenador Macintosh esta propiedad es el tipo de archivo de cuatro caracteres, que solo se utiliza en las versiones de Mac OS anteriores a Mac OS X).
url	La URL del archivo o directorio. Consulte "Rutas a objetos File" en la página 696.

Para obtener más información sobre estas propiedades, consulte la entrada de la clase File en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Copia y movimiento de directorios

Adobe AIR 1.0 y posterior

La clase File incluye dos métodos de copiar archivos o directorios: `copyTo()` y `copyToAsync()`. La clase File incluye dos métodos de mover archivos o directorios: `moveTo()` y `moveToAsync()`. Los métodos `copyTo()` y `moveTo()` funcionan de modo sincrónico y los métodos `copyToAsync()` y `moveToAsync()` son asíncronos (consulte "Aspectos básicos de los archivos de AIR" en la página 695).

Para copiar o mover un archivo se configuran dos objetos File. Uno de ellos apunta al archivo a copiar o mover y es el objeto que llama al método de copiar o mover, mientras que el otro apunta a la ruta de destino (el resultado).

Lo que sigue copia un archivo test.txt desde el subdirectorio AIR Test del directorio de documentos del usuario en un archivo llamado copy.txt en el mismo directorio:

```
var original:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile:File = File.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

En este ejemplo el valor del parámetro `overwrite` del método `copyTo()` (el segundo parámetro) está definido en `true`. Al definir `overwrite` como `true`, si el archivo de destino ya existe se sobrescribe. Este parámetro es opcional. Si está definido en `false` (el valor predeterminado), la operación distribuye un evento IOErrorEvent en el caso de que el archivo de destino ya exista (el archivo no se copia).

Las versiones "Async" de los métodos de copiar y mover funcionan de modo asíncrono. Utilice el método `addEventListener()` para controlar si se concluye la tarea o se produce un estado de error, como en el código siguiente:

```
var original = File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination:File = File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event:Event):void {
    trace(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event:IOErrorEvent):void {
    trace("I/O Error.");
}
```

La clase File también incluye los métodos `File.moveToTrash()` y `File.moveToTrashAsync()`, que sirven para trasladar un archivo o directorio a la papelera del sistema.

Eliminación de archivos

Adobe AIR 1.0 y posterior

La clase File incluye un método `deleteFile()` y un método `deleteFileAsync()`. Estos métodos eliminan archivos, el primero de forma sincrónica y el segundo de forma asíncrona (consulte "Aspectos básicos de los archivos de AIR" en la página 695).

En el ejemplo siguiente, el código elimina de modo sincrónico el archivo test.txt del directorio de documentos del usuario:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

El código siguiente elimina de modo asíncrono el archivo test.txt del directorio de documentos del usuario:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

También se incluyen los métodos `moveToTrash()` y `moveToTrashAsync()`, que sirven para trasladar un archivo o directorio a la papelera del sistema. Para obtener más información, consulte "Traslado de archivos a la papelera" en la página 716.

Traslado de archivos a la papelera

Adobe AIR 1.0 y posterior

La clase File incluye un método `moveToTrash()` y un método `moveToTrashAsync()`. Estos métodos envían un archivo o directorio a la papelera del sistema, el primero de forma sincrónica y el segundo de forma asíncrona (consulte "Aspectos básicos de los archivos de AIR" en la página 695).

En el ejemplo siguiente, el código traslada de modo sincrónico el archivo test.txt del directorio de documentos del usuario a la papelera del sistema:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

Nota: *en sistemas operativos que no admiten el concepto de carpeta de papelera recuperable, los archivos se eliminan inmediatamente.*

Creación de archivos temporales

Adobe AIR 1.0 y posterior

La clase File incluye un método `createTempFile()` que crea un archivo en la carpeta de directorios temporales para el sistema, como en el ejemplo siguiente:

```
var temp:File = File.createTempFile();
```

El método `createTempFile()` crea automáticamente un archivo temporal exclusivo (ahorrándole el trabajo de determinar un nuevo lugar exclusivo).

Se puede utilizar un archivo temporal para guardar de forma provisional la información que se utiliza en una sesión de la aplicación. Obsérvese que existe además un método `createTempDirectory()` para crear nuevos directorios temporales exclusivos en el directorio temporal System.

Puede que desee eliminar el archivo temporal antes de cerrar la aplicación, dado que *no* se elimina de forma automática en todos los dispositivos.

Trabajo con volúmenes de almacenamiento

Adobe AIR 2 y posterior

En AIR 2, es posible detectar el momento en que los volúmenes de almacenamiento masivo se montan o se desmontan. La clase StorageVolumeInfo define un objeto singleton `storageVolumeInfo`. El objeto `StorageVolumeInfo.storageVolumeInfo` distribuye un evento `storageVolumeMount` cuando se monta el volumen de almacenamiento. Asimismo, se distribuye un evento `storageVolumeUnmount` si el volumen se desmonta. La clase StorageVolumeChangeEvent define estos eventos.

Nota: *en distribuciones modernas de Linux, el objeto StorageVolumeInfo solo distribuye eventos `storageVolumeMount` y `storageVolumeUnmount` para dispositivos físicos y unidades de red montadas en ubicaciones determinadas.*

La propiedad `storageVolume` de la clase StorageVolumeChangeEvent es un objeto StorageVolume. La clase StorageVolume define las propiedades básicas del volumen de almacenamiento:

- `drive`: letra de la unidad del volumen en Windows (`null` en otros sistemas operativos)
- `fileSystemType`: tipo del sistema de archivos en el volumen de almacenamiento (por ejemplo, "FAT", "NTFS", "HFS" o "UFS").
- `isRemoveable`: indica si se puede eliminar un volumen (`true`) o no (`false`).
- `isWritable`: indica si se puede modificar un volumen (`true`) o no (`false`)
- `name`: nombre del volumen.
- `rootDirectory`: objeto File que se corresponde con el directorio raíz del volumen.

La clase StorageVolumeChangeEvent también incluye una propiedad `rootDirectory`. La propiedad `rootDirectory` es un objeto File que hace referencia al directorio raíz del volumen de almacenamiento que se ha montado o desmontado.

La propiedad `storageVolume` del objeto StorageVolumeChangeEvent es undefined (`null`) en volúmenes sin montar. Sin embargo, se puede acceder a la propiedad `rootDirectory` del evento.

El siguiente código genera el nombre y la ruta de archivo de un volumen de almacenamiento cuando se monta:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_MOUNT, onVolumeMount);
function onVolumeMount(event:StorageVolumeChangeEvent):void
{
    trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

El siguiente código genera la ruta de archivo de un volumen de almacenamiento cuando se desmonta:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_UNMOUNT, onVolumeUnmount);
function onVolumeUnmount(event:StorageVolumeChangeEvent):void
{
    trace(event.rootDirectory.nativePath);
}
```

El objeto `StorageVolumeInfo.storageVolumeInfo` incluye un método `getStorageVolumes()`. Este método devuelve un vector de los objetos StorageVolume que se corresponde con los volúmenes de almacenamiento montados actualmente. El siguiente código muestra cómo agrupar los nombres y los directorios raíz de todos los volúmenes de almacenamiento montados:

```
var volumes:Vector.<StorageVolume> = new Vector.<StorageVolume>;
volumes = StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (var i:int = 0; i < volumes.length; i++)
{
    trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

Nota: *en distribuciones modernas de Linux, el método `getStorageVolumes()` devuelve objetos correspondientes a dispositivos físicos y unidades de red montadas en ubicaciones determinadas.*

El método `File.getRootDirectories()` incluye los directorios raíz (consulte "Apuntar a la raíz del sistema de archivos" en la página 701. No obstante, los objetos StorageVolume (enumerados mediante el método `StorageVolumeInfo.getStorageVolumes()`) ofrecen más información sobre los volúmenes de almacenamiento.

La propiedad `spaceAvailable` de la propiedad `rootDirectory` de un objeto StorageVolume se puede utilizar para obtener el espacio disponible en un volumen de almacenamiento. (Consulte "Determinación del espacio disponible en un volumen" en la página 709.)

Más temas de ayuda

StorageVolume

StorageVolumeInfo

Lectura y escritura de archivos

Adobe AIR 1.0 y posterior

La clase FileStream permite que las aplicaciones de AIR puedan leer y escribir en el sistema de archivos.

Flujo de trabajo de lectura y escritura de archivos

Adobe AIR 1.0 y posterior

El flujo de trabajo para la lectura y escritura de archivos es el siguiente.

Inicialice un objeto File que apunte a la ruta.

El objeto File representa la ruta del archivo con la que se desea trabajar (o un archivo que se va a crear).

```
var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

En este ejemplo se utiliza la propiedad `File.documentsDirectory` y el método `resolvePath()` de un objeto File para inicializar el objeto File. Existen también varias formas más de configurar un objeto File para que apunte a un archivo. Para obtener más información, consulte "Configuración de un objeto File para que apunte a un archivo" en la página 703.

Inicialice un objeto FileStream.

Llame al método open() u openAsync() del objeto FileStream.

El método que se llame dependerá de si se desea abrir el archivo para realizar operaciones sincrónicas o asíncronas. Utilice el objeto File como parámetro `file` del método open. Para el parámetro `fileMode`, indique una constante de la clase FileMode que especifique la forma en que se utilizará el archivo.

En el ejemplo siguiente, el código inicializa un objeto FileStream que se utiliza para crear un archivo y sobrescribir los datos existentes:

```
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);
```

Para obtener más información, consulte "Inicialización de un objeto FileStream, y apertura y cierre de archivos" en la página 721 y "Modos de apertura de FileStream" en la página 720.

Si abrió el archivo de forma asíncrona (con el método openAsync()), añada y configure detectores de eventos para el objeto FileStream.

Estos métodos de detección de eventos responden a eventos distribuidos por el objeto FileStream en distintas situaciones. Entre estas situaciones se incluyen el momento en que los datos se leen desde el archivo, cuando se detectan errores de E/S o cuando se ha escrito la cantidad total de datos que se debe escribir.

Para obtener más información, consulte "Programación asíncrona y eventos generados por un objeto FileStream abierto de forma asíncrona" en la página 725.

Incluya código para leer y escribir datos, según proceda.

Existen varios métodos de la clase FileStream relacionados con la lectura y la escritura. (Cada uno empieza con "read" o "write"). El método que se elija para leer o escribir datos depende del formato de los datos en el archivo de destino.

Por ejemplo, si el contenido del archivo de destino es texto codificado en UTF, se pueden utilizar los métodos `readUTFBytes()` y `writeUTFBytes()`. Si se desea tratar los datos como conjuntos de bytes, se pueden utilizar los métodos `readByte()`, `readBytes()`, `writeByte()` y `writeBytes()`. Para obtener más información, consulte "Formatos de datos y elección de los métodos de lectura y escritura" en la página 726.

Si abrió el archivo de forma asíncrona, asegúrese de que disponga de suficientes datos antes de llamar a un método de lectura. Para obtener más información, consulte "Búfer de lectura y propiedad bytesAvailable de un objeto FileStream" en la página 723.

Antes de escribir en un archivo, si se desea comprobar la cantidad de espacio disponible en el disco se puede comprobar la propiedad spaceAvailable del objeto File. Para obtener más información, consulte "Determinación del espacio disponible en un volumen" en la página 709.

Llame al método close() del objeto FileStream cuando haya terminado de trabajar con el archivo.
La llamada al método close() hace que el archivo esté disponible para las demás aplicaciones.

Para obtener más información, consulte "Inicialización de un objeto FileStream, y apertura y cierre de archivos" en la página 721.

Para ver una aplicación de muestra que utiliza la clase FileStream para leer y escribir archivos, consulte los artículos siguientes en el centro de desarrollo de Adobe AIR:

- Building a text-file editor (Creación de un editor de archivos de texto, en inglés)
- Building a text-file editor (Creación de un editor de archivos de texto, en inglés)
- Reading and writing from an XML preferences file (Lectura y escritura desde un archivo XML de preferencias, en inglés)

Trabajo con objetos FileStream

Adobe AIR 1.0 y posterior

La clase FileStream define métodos para abrir, leer y escribir archivos.

Modos de apertura de FileStream

Adobe AIR 1.0 y posterior

Los métodos `open()` y `openAsync()` de un objeto FileStream incluyen cada uno un parámetro `fileMode` que define algunas propiedades para una secuencia de archivo, entre ellas:

- La capacidad de leer del archivo
- La capacidad de escribir en el archivo
- Si los datos se anexarán siempre al final del archivo (al escribir)
- Qué hacer si el archivo no existe (y cuando no existen los directorios superiores)

A continuación se enumeran los distintos modos de archivo (que se pueden especificar como parámetro `fileMode` de los métodos `open()` y `openAsync()`):

Modo de archivo	Descripción
FileMode.READ	Especifica que el archivo está abierto solo para fines de lectura.
FileMode.WRITE	Especifica que el archivo está abierto con posibilidad de escritura. Si el archivo no existe, se crea al abrirse el objeto FileStream. Si el archivo existe, se eliminan los datos existentes.
FileMode.APPEND	Especifica que el archivo está abierto con posibilidad de anexarle datos. Si el archivo no existe, se crea el mismo. Si el archivo existe, no se sobrescriben los datos existentes y la escritura comienza al final del archivo.
FileMode.UPDATE	Especifica que el archivo está abierto con posibilidad de lectura y escritura. Si el archivo no existe, se crea el mismo. Especifique este modo para tener acceso directo de lectura/escritura al archivo. Se puede leer desde cualquier posición del archivo. Al escribir en el archivo, solo los bytes que se escriben nuevamente sobreescriben los bytes existentes (todos los demás permanecen sin modificar).

Inicialización de un objeto FileStream, y apertura y cierre de archivos

Adobe AIR 1.0 y posterior

Al abrir un objeto FileStream, este queda a disposición para leer y escribir datos en un archivo. Para abrir un objeto FileStream se pasa un objeto File al método `open()` o `openAsync()` del objeto FileStream:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

El parámetro `fileMode` (el segundo parámetro de los métodos `open()` y `openAsync()`) especifica el modo en que ha de abrirse el archivo: para lectura, escritura, anexado o actualización. Para ver más información, consulte la sección anterior, "Modos de apertura de FileStream" en la página 720.

Si se utiliza el método `openAsync()` de abrir el archivo para operaciones asíncronas, se debe configurar los detectores de eventos para que controlen eventos asíncronos:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(IOErrorEvent.IO_Error, errorHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function completeHandler(event:Event):void {
    // ...
}

function progressHandler(event:ProgressEvent):void {
    // ...
}

function errorHandler(event:IOErrorEvent):void {
    // ...
}
```

El archivo se abre para realizar operaciones sincrónicas o asíncronas, según se utilice el método `open()` o `openAsync()`. Para obtener más información, consulte "Aspectos básicos de los archivos de AIR" en la página 695.

Si se define el parámetro `fileMode` en `FileMode.READ` o `FileMode.UPDATE` en el método open del objeto FileStream, los datos se leerán en el búfer de lectura en cuanto se abra el objeto FileStream. Para obtener más información, consulte "Búfer de lectura y propiedad bytesAvailable de un objeto FileStream" en la página 723.

Se puede llamar al método `close()` de un objeto FileStream para cerrar el archivo asociado, dejándolo a disposición de otras aplicaciones.

Propiedad "position" de un objeto FileStream

Adobe AIR 1.0 y posterior

La propiedad `position` de un objeto FileStream determina dónde se leerán o escribirán los datos en el siguiente método de lectura o escritura.

Antes de una operación de lectura o escritura, configure la propiedad `position` con cualquier posición válida del archivo.

En el ejemplo siguiente, el código escribe la cadena `"hello"` (con código UTF) en la posición 8 del archivo:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

Cuando recién se abre un objeto FileStream, la propiedad `position` es 0.

Antes de una operación de lectura, el valor de `position` debe ser 0 como mínimo y menos que la cantidad de bytes del archivo (que son las posiciones existentes en el archivo).

El valor de la propiedad `position` solo se modifica en las siguientes situaciones:

- cuando se configura explícitamente la propiedad `position`;
- cuando se llama a un método de lectura;
- cuando se llama a un método de escritura.

Cuando se llama a un método de lectura o escritura de un objeto FileStream, la propiedad `position` se incrementa inmediatamente en la cantidad de bytes que se leen o escriben. Dependiendo del método de lectura que se utilice, la propiedad `position` se incrementa en la cantidad de bytes que se especifican para leer o en la cantidad de bytes que hay disponibles. Posteriormente, al llamar a un método de lectura o escritura se leerá o escribirá empezando en la nueva posición.

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
trace(myFileStream.position); // 4200
```

Hay una excepción a ello: si el FileStream se abrió en modo de anexado, la propiedad `position` no se modifica tras llamar a un método de escritura. (En el modo de anexado, los datos siempre se escriben al final del archivo, independientemente del valor de la propiedad `position`).

Si es un archivo que se abrió para realizar operaciones asíncronas, la operación de escritura no se finaliza antes de ejecutarse la siguiente línea del código. Sin embargo, se puede llamar a varios métodos asíncronos de forma secuencial y la rutina los ejecuta en la secuencia prevista:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(Event.CLOSE, closeHandler);
myFileStream.close();
trace("started.");

closeHandler(event:Event):void
{
    trace("finished.");
}
```

La salida de la sentencia trace para este código es:

```
started.
finished.
```

Sí se puede especificar el valor de `position` inmediatamente después de llamar a un método de lectura o escritura (o en cualquier momento); la siguiente operación de lectura o escritura se llevará a cabo empezando en esa posición. Por ejemplo, observe que el siguiente código define la propiedad `position` inmediatamente después de llamar a la operación `writeBytes()`, y `position` se define en ese valor (300) incluso después de haberse concluido la operación de escritura:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
trace(myFileStream.position); // 300
```

Búfer de lectura y propiedad bytesAvailable de un objeto FileStream

Adobe AIR 1.0 y posterior

Cuando se abre un objeto FileStream con capacidad de lectura (uno en que el parámetro `fileMode` del método `open()` o `openAsync()` se definió en `READ` o `UPDATE`), el motor de ejecución guarda los datos en un búfer interno. El objeto FileStream empieza a leer datos en el búfer en cuanto se abre el archivo (llamando al método `open()` o `openAsync()` del objeto FileStream).

Si es un archivo que se abrió para realizar operaciones sincrónicas (con el método `open()`), siempre se puede configurar el puntero `position` en cualquier posición (dentro de los límites del archivo) y empezar a leer cualquier cantidad de datos (dentro de los límites del archivo), como se ilustra en el siguiente código, que presupone que el archivo contiene por lo menos 100 bytes):

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

Se abra el archivo para operaciones sincrónicas o para operaciones asíncronas, los métodos de lectura siempre leen desde los bytes "disponibles", representados por la propiedad `bytesAvailable`. Al leer de forma sincrónica, todos los bytes del archivo están disponibles todo el tiempo. Al leer de forma asíncrona, los bytes quedan disponibles a partir de la posición especificada por la propiedad `position` en una serie de llenados de búfer asíncronos señalados por los eventos `progress`.

Para archivos que se abren para realizar operaciones *sincrónicas*, la propiedad `bytesAvailable` se define siempre de modo que represente la cantidad de bytes desde la propiedad `position` hasta el final del archivo (para fines de lectura, siempre están disponibles todos los bytes del archivo).

En el caso de archivos abiertos para realizar operaciones *asíncronas*, hay que asegurarse de que el búfer de lectura haya consumido suficientes datos antes de llamar a un método de lectura. Para un archivo abierto de forma asíncrona, a medida que avanza la operación de lectura se van añadiendo al búfer los datos del archivo -empezando por el valor especificado para `position` cuando se inició la operación de lectura- y la propiedad `bytesAvailable` se incrementa con cada byte que se lee. La propiedad `bytesAvailable` indica la cantidad de bytes que hay disponibles desde el byte de la posición especificada por la propiedad `position` y el final del búfer. El objeto FileStream envía periódicamente un evento `progress`.

Para un archivo abierto de forma asíncrona, a medida que los datos quedan disponibles en el búfer de lectura el objeto FileStream distribuye periódicamente el evento `progress`. Por ejemplo, el siguiente código lee datos en un objeto ByteArray, `bytes`, a medida que se van leyendo los mismos para ponerlos en el búfer.

```
 var bytes:ByteArray = new ByteArray();
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function progressHandler(event:ProgressEvent):void
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

Para un archivo abierto de forma asíncrona, solo se pueden leer los datos que se encuentran en el búfer de lectura. Además, a medida que el usuario lee los datos, estos se eliminan del búfer de lectura. Para las operaciones de lectura hay que asegurarse de que los datos existan en el búfer de lectura antes de llamar a la operación de lectura. En el siguiente ejemplo el código lee 8.000 bytes de datos empezando por la posición 4.000 del archivo:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
myFileStream.position = 4000;

var str:String = "";

function progressHandler(event:Event):void
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

Durante una operación de escritura, el objeto FileStream no lee datos para ponerlos en el búfer de lectura. Cuando finaliza una operación de escritura (todos los datos en el búfer de escritura se han escrito en el archivo), el objeto FileStream inicia un nuevo búfer de lectura (suponiendo que el objeto FileStream asociado se abrió con capacidad de lectura) y empieza a leer datos para ponerlos en el búfer de lectura, comenzando por la posición especificada por la propiedad `position`. La propiedad `position` puede ser la posición del último byte escrito, o puede ser otra posición si el usuario especifica otro valor para el objeto `position` después de la operación de escritura.

Programación asíncrona y eventos generados por un objeto FileStream abierto de forma asíncrona

Adobe AIR 1.0 y posterior

Cuando se abre un archivo de forma asíncrona (con el método `openAsync()`), la lectura y escritura de los archivos se realiza de modo asíncrono. Puede ejecutarse otros códigos ActionScript a medida que se leen datos para ponerlos en el búfer de lectura y se escriben los datos de salida.

Ello significa que hay que registrarse para los eventos generados por el objeto FileStream que se abren de forma asíncrona.

Al registrarse para el evento `progress`, se puede notificar al usuario a medida que se disponen de nuevos datos para la lectura, como en el código siguiente:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progressHandler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Para leer la totalidad de los datos, regístrese para el evento `complete`, como en el código siguiente:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";
function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

De modo muy similar a lo que sucede con los datos de entrada que pasan al búfer para permitir la lectura asíncrona, los datos que se escriben en una secuencia asíncrona pasan a un búfer para escribirse de forma asíncrona en el archivo. A medida que se escriben los datos en un archivo, el objeto FileStream distribuye periódicamente un objeto `OutputProgressEvent`. Un objeto `OutputProgressEvent` incluye una propiedad `bytesPending` que se ajusta a la cantidad de bytes que quedan por escribir. Puede registrarse para el evento `outputProgress` para que se le notifique el momento en que se escriba el contenido del búfer en el archivo, lo que permitirá presentar un cuadro de diálogo del progreso, por ejemplo, pero en general no es necesario. En especial, puede llamar al método `close()` sin preocuparse por los bytes sin escribir. El objeto FileStream seguirá escribiendo datos y el evento `close` se entregará cuando se haya escrito el byte final en el archivo y se haya cerrado el archivo subyacente.

Formatos de datos y elección de los métodos de lectura y escritura

Adobe AIR 1.0 y posterior

Cada archivo es un grupo de bytes en un disco. En ActionScript, los datos de un archivo pueden siempre representarse como un conjunto de bytes (ByteArray). En el siguiente ejemplo el código lee los datos de un archivo para ponerlos en un objeto ByteArray denominado `bytes`:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var bytes:ByteArray = new ByteArray();

function completeHandler(event:Event):void
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

Asimismo, el código siguiente escribe los datos de ByteArray denominado `bytes` en un archivo:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

Sin embargo, con frecuencia no se desea guardar los datos en un objeto ByteArray de ActionScript. Y sucede con frecuencia que el archivo de datos tiene un formato de archivo especificado.

Por ejemplo, los datos del archivo pueden ser en formato de archivo de texto, y tal vez no le interese representar estos datos en un objeto String.

Por este motivo la clase FileStream incluye métodos de lectura y escritura para leer y escribir datos en (y de) tipos de objetos que no sean ByteArray. Por ejemplo, el método `readMultiByte()` permite leer datos de un archivo y guardarlos en una cadena, como en el siguiente código:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

El segundo parámetro del método `readMultiByte()` especifica el formato de texto que utiliza ActionScript para interpretar los datos ("iso-8859-1" en el ejemplo). Adobe AIR admite las codificaciones de juegos de caracteres comunes (consulte Juegos de caracteres admitidos).

La clase FileStream incluye también el método `readUTFBytes()`, que lee datos del búfer de lectura y los pone en una cadena utilizando el juego de caracteres UTF-8. Dada la longitud variable de los caracteres del juego de caracteres UTF-8, no utilice `readUTFBytes()` en un método que responda al evento `progress`, puesto que los datos al final del búfer de lectura pueden representar un carácter incompleto. (Este es también el caso cuando se emplea el método `readMultiByte()` con codificación de caracteres de longitud variable). Por este motivo conviene leer la totalidad de los datos cuando el objeto FileStream distribuye el evento `complete`.

También hay métodos similares de escritura, `writeMultiByte()` y `writeUTFBytes()`, para trabajar con objetos String y archivos de texto.

Los métodos `readUTF()` y `writeUTF()` (no deben confundirse con `readUTFBytes()` y `writeUTFBytes()`) también leen y escriben los datos de texto en un archivo, pero presuponen que los datos de texto vienen precedidos de datos que especifican la longitud de los datos de texto, lo cual no es común con los archivos de texto estándar.

Algunos archivos de texto en código UTF empiezan con un carácter "UTF-BOM" (marca de orden de bytes) que define la propiedad "endian" además del formato de codificación (como UTF-16 o UTF-32).

Para ver un ejemplo de lectura y escritura en un archivo de texto, consulte "Ejemplo: Lectura de un archivo XML para ponerlo en un objeto XML" en la página 728.

`readObject()` y `writeObject()` son formas convenientes de guardar y recuperar datos para objetos ActionScript complejos. Los datos se codifican en AMF (formato de mensaje de acción). Adobe AIR, Flash Player, Flash Media Server y Flex Data Services incluyen APIs para trabajar con datos en este formato.

Existen algunos otros métodos de lectura y escritura (como `readDouble()` y `writeDouble()`). Si se utiliza alguno de estos, asegúrese de que el formato del archivo se corresponda con los formatos de los datos definidos por estos métodos.

Los formatos de archivo son a menudo más complejos que formatos de texto sencillo. Por ejemplo, un archivo MP3 incluye datos comprimidos que solo pueden interpretarse con algoritmos de descompresión y descodificación que son exclusivos para archivos MP3. Los archivos MP3 pueden incluir también etiquetas ID3 que contienen información acerca del archivo en forma de metaetiquetas (como el título y el artista de una canción). Existen varias versiones del formato ID3, pero la más sencilla de ellas (ID3 versión 1) se trata en la sección "Ejemplo: Lectura y escritura de datos con acceso directo" en la página 729.

Otros formatos de archivos (para imágenes, bases de datos, documentos de aplicaciones, etc.) tienen distintas estructuras; para trabajar con estos datos en ActionScript hay que entender cómo están estructurados los mismos.

Uso de los métodos load() y save()

Flash Player 10 y posterior, Adobe AIR 1.5 y posterior

Flash Player 10 añadió los métodos `load()` y `save()` a la clase FileReference. Estos métodos también se encuentran en AIR 1.5; la clase File hereda los métodos de FileReference. Estos métodos se diseñaron para proporcionar un medio seguro para cargar y guardar datos de archivos en Flash Player. Sin embargo, las aplicaciones de AIR también pueden utilizar estos métodos como un modo sencillo de cargar y guardar archivos de forma asincrónica.

Por ejemplo, el siguiente código guarda un cadena en un archivo de texto:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
var str:String = "Hello.";
file.addEventListener(Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event:Event):void
{
    trace("Done.");
}
```

El parámetro `data` del método `save()` puede adoptar un valor String, XML o ByteArray. Cuando el argumento es un valor String o XML, el método guarda el archivo como archivo de texto codificado como UTF-8.

Cuando se ejecuta este ejemplo de código, la aplicación muestra un cuadro de diálogo en el que el usuario selecciona el destino del archivo guardado.

El siguiente código carga una cadena de un archivo de texto codificado como UTF-8:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, loaded);
file.load();
var str:String;
function loaded(event:Event):void
{
    var bytes:ByteArray = file.data;
    str = bytes.readUTFBytes(bytes.length);
    trace(str);
}
```

La clase FileStream proporciona más funcionalidad que la que ofrecen los métodos `load()` y `save()`:

- Con el uso de la clase FileStream, puede leer y escribir datos de forma sincrónica y asíncrona.
- El uso de la clase FileStream permite escribir de forma incremental en un archivo.
- La utilización de la clase FileStream permite abrir un archivo para acceso aleatorio (pudiendo escribir y leer en cualquier sección del archivo).
- La clase FileStream permite especificar el tipo de acceso al archivo del que se dispone, estableciendo el parámetro `fileMode` del método `open()` u `openAsync()`.
- La clase FileStream permite guardar datos en archivos sin tener que mostrar al usuario un cuadro de diálogo para abrir o guardar.
- Puede utilizar directamente tipos distintos a los conjuntos de bytes al leer datos con la clase FileStream.

Ejemplo: Lectura de un archivo XML para ponerlo en un objeto XML

Adobe AIR 1.0 y posterior

Los siguientes ejemplos muestran cómo leer y escribir en un archivo de texto que contiene datos XML.

Para leer del archivo, inicialice los objetos File y FileStream, llame al método `readUTFBytes()` del objeto FileStream y convierta la cadena en objeto XML:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.READ);
var prefsXML:XML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
fileStream.close();
```

Asimismo, para escribir los datos en el archivo basta con configurar objetos File y FileStream apropiados y luego llamar a un método de escritura del objeto FileStream. Pase la versión en cadena de los datos XML al método de escritura, como en el código siguiente:

```
var prefsXML:XML = <prefs><autoSave>true</autoSave></prefs>;
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);

var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += prefsXML.toXMLString();

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

En estos ejemplos se utilizan los métodos `readUTFBytes()` y `writeUTFBytes()` porque presuponen que los archivos están en formato UTF-8. En caso contrario, puede resultar necesario usar otro método (consulte "Formatos de datos y elección de los métodos de lectura y escritura" en la página 726).

Los ejemplos anteriores utilizan objetos FileStream abiertos para una operación sincrónica. También se pueden abrir archivos para operaciones asíncronas (que dependen de las funciones de detección de eventos para responder a los eventos). Por ejemplo, el siguiente código muestra cómo leer un archivo XML de forma asíncrona:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.addEventListener(Event.COMPLETE, processXMLData);
fileStream.openAsync(file, FileMode.READ);
var prefsXML:XML;

function processXMLData(event:Event):void
{
    prefsXML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
    fileStream.close();
}
```

El método `processXMLData()` se invoca cuando se lee la totalidad del archivo para ponerlo en el búfer del lectura (cuando el objeto FileStream distribuye el evento `complete`). Llama al método `readUTFBytes()` para obtener una versión en cadena de los datos leídos, y crea un objeto XML, `prefsXML`, con base en esa cadena.

Para ver un ejemplo de aplicación que muestra estas capacidades, consulte Reading and writing from an XML preferences file (Lectura y escritura desde un archivo XML de preferencias, en inglés).

Ejemplo: Lectura y escritura de datos con acceso directo

Adobe AIR 1.0 y posterior

Los archivos MP3 pueden incluir etiquetas ID3, que son secciones al principio o al final del archivo que contienen metadatos para identificar la grabación. El formato mismo de la etiqueta ID3 tiene distintas revisiones. Este ejemplo describe cómo leer y escribir de un archivo MP3 que contiene el formato ID3 más sencillo (ID3 versión 1.0) usando "datos de acceso directo al archivo", que significa que lee y escribe en lugares arbitrarios del archivo.

Un archivo MP3 que contiene una etiqueta ID3 versión 1 incluye los datos ID3 al final del archivo, en los últimos 128 bytes.

Al acceder a un archivo para fines de lectura/escritura directa, es importante especificar `FileMode.UPDATE` como parámetro `fileMode` para el método `open()` o `openAsync()`:

```
var file:File = File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr:FileStream = new FileStream();
fileStr.open(file, FileMode.UPDATE);
```

Esto permite tanto leer como escribir en el archivo.

Al abrir el archivo se puede definir el puntero `position` en la posición de 128 bytes antes del final del archivo:

```
fileStr.position = file.size - 128;
```

Este código define la propiedad `position` en esta ubicación del archivo porque el formato ID3 v1.0 especifica que los datos de la etiqueta ID3 se guardan en los últimos 128 bytes del archivo. La especificación también estipula lo siguiente:

- Los 3 primeros bytes de la etiqueta contienen la cadena `"TAG"`.

- Los 30 caracteres siguientes contienen el título del tema MP3 en forma de cadena.
- Los 30 caracteres siguientes contienen el nombre del artista en forma de cadena.
- Los 30 caracteres siguientes contienen el nombre del álbum en forma de cadena.
- Los 4 caracteres siguientes contienen el año en forma de cadena.
- Los 30 caracteres siguientes contienen el comentario en forma de cadena.
- El siguiente byte contiene un código que indica el género del tema.
- Todos los datos de texto están en formato ISO 8859-1.

El método `id3TagRead()` comprueba los datos después de leídos (en el momento del evento `complete`):

```
 function id3TagRead():void
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year:String = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode:String =  fileStr.readByte().toString(10);
    }
}
```

También se puede realizar una escritura de acceso directo en el archivo. Por ejemplo, se podría analizar la variable `id3Title` para asegurarse de que las mayúsculas sean correctas (empleando los métodos de la clase String) y después escribir en el archivo una cadena modificada, denominada `newTitle`, como en el caso siguiente:

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

Para cumplir la norma ID3 versión 1, la longitud de la cadena `newTitle` debe ser de 30 caracteres, con el carácter 0 de relleno al final (`String.fromCharCode(0)`).

Capítulo 39: Almacenamiento de datos locales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede utilizar la clase SharedObject para almacenar pequeñas cantidades de datos en el equipo clientes. En Adobe AIR, también puede utilizar la clase EncryptedLocalStore para almacenar pequeñas cantidades de datos privados en el equipo local en una aplicación de AIR.

También se pueden leer y escribir archivos en el sistema de archivos y (en Adobe AIR) acceder a archivos de la base de datos local. Para obtener más información, consulte "Trabajo con el sistema de archivos" en la página 679 y "Trabajo con bases de datos SQL locales en AIR" en la página 744.

Existe una serie de factores de seguridad relacionados con los objetos compartidos. Para obtener más información, consulte "Objetos compartidos" en la página 1120 en "Seguridad" en la página 1085.

Objetos compartidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un objeto compartido, a menudo referido como "cookie de Flash", es un archivo de datos que se puede crear en el equipo a partir de los sitios que se visitan. Los objetos compartidos se utilizan sobre todo para mejorar la navegación en web ya que, por ejemplo, permiten personalizar el aspecto de un sitio web que se visita con frecuencia.

Acerca de los objetos compartidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los objetos compartidos funcionan como las cookies del navegador. Puede utilizar la clase SharedObject para almacenar datos en el disco duro local del usuario y llamar a los datos durante la misma sesión o en una posterior. Las aplicaciones pueden acceder únicamente a sus propios datos SharedObject y solo si se están ejecutando en el mismo dominio. Los datos no se envían al servidor y no se puede acceder a ellos mediante otras aplicaciones que se ejecuten en otros dominios, pero sí pueden acceder a ellos las aplicaciones del mismo dominio.

Objetos compartidos comparados con cookies

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las cookies y los objetos compartidos son muy similares. La mayor parte de los programadores web están familiarizados con el modo de funcionamiento de las cookies, por lo que puede resultar útil compararlas con los objetos SharedObject locales.

Las cookies que se adhieren al estándar RFC 2109 suelen disponer de las siguientes propiedades:

- Pueden caducar y a menudo lo hacen al final de una sesión y de forma predeterminada.
- Pueden desactivarse mediante el cliente en función de los criterios específicos de un sitio.

- Existe un límite de 300 cookies en total y 20 cookies como máximo por sitio.
- Se suelen limitar a un tamaño de 4 KB cada una.
- En ocasiones se consideran una amenaza para la seguridad y, como resultado, se deshabilitan en el cliente.
- Se almacenan en una ubicación especificada por el navegador cliente.
- Se transmiten del cliente al servidor mediante HTTP.

Por el contrario, los objetos compartidos cuentan con las propiedades siguientes:

- No caducan de forma predeterminada.
- De forma predeterminada, se limitan a un tamaño de 100 KB cada uno.
- Pueden almacenar tipos de datos sencillos (por ejemplo, String, Array y Date).
- Se almacenan en una ubicación que especifica la aplicación (en el directorio de inicio del usuario).
- Nunca se transmiten entre el cliente y el servidor.

Acerca de la clase SharedObject

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si utiliza la clase SharedObject, podrá crear y eliminar objetos compartidos, además de detectar el tamaño actual de un objeto SharedObject que esté utilizando.

Creación de un objeto compartido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear un objeto SharedObject, utilice el método `SharedObject.getLocal()`, que tiene la siguiente sintaxis:

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

El siguiente ejemplo crea un objeto compartido denominado mySO:

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

Con ello se crea un archivo en equipo del cliente llamado preferences.sol.

El término *local* hace referencia a la ubicación del objeto compartido. En este caso, Adobe® Flash® Player almacena el archivo SharedObject localmente en el directorio de inicio del cliente.

Cuando se crea un objeto compartido, Flash Player crea un nuevo directorio para la aplicación y el dominio dentro de su entorno limitado. También crea un archivo *.sol que almacena los datos de SharedObject. La ubicación predeterminada de este archivo es un subdirectorio del directorio de inicio del usuario. La siguiente tabla muestra las ubicaciones predeterminadas de este directorio:

Sistema operativo	Ubicación
Windows 95/98/ME/2000/XP	`c:/Documents and Settings/`*`username`*`/Application Data/Macromedia/Flash Player/#SharedObjects`

Sistema operativo	Ubicación
Windows Vista/Windows 7	`c:/Users/`*`username`*`/AppData/Roaming/Macromedia/Flash Player/#SharedObjects`
Macintosh OS X	`/Users/`*`username`*`/Library/Preferences/Macromedia/Flash Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol`
Linux/Unix	`/home/username/.macromedia/Flash_Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol`

Bajo el directorio #SharedObjects existe un directorio nombrado al azar. Debajo hay un directorio que coincide con el nombre de host y, a continuación, la ruta a la aplicación y finalmente el archivo *.sol.

Por ejemplo, si solicita una aplicación denominada MyApp.swf en el host local, y en un subdirectorio llamado /sos, Flash Player almacena el archivo *.sol en la siguiente ubicación en Windows XP:

```
c:/Documents and Settings/fred/Application Data/Macromedia/Flash
Player/#SharedObjects/KROKWXRK/#localhost/sos/MyApp.swf/data.sol
```

Nota: *si no proporcionan ningún nombre en el método* `SharedObject.getLocal()`*, Flash Player asigna un nombre al archivo undefined.sol.*

De forma predeterminada, Flash puede guardar objetos SharedObject persistentes localmente de hasta 100 KB por dominio. Este valor puede configurarlo el usuario. Si la aplicación intenta guardar datos en un objeto compartido que podrían hacerlo superior a 100 KB, Flash Player muestra el cuadro de diálogo Almacenamiento local, que permite al usuario permitir o rechazar más almacenamiento local para el dominio que está solicitando acceso.

Cómo especificar una ruta

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede utilizar el parámetro opcional *pathname* para especificar una ubicación para el archivo SharedObject. Este archivo debe ser un subdirectorio del directorio SharedObject de ese dominio. Por ejemplo, si se solicita una aplicación en el host local y se especifica lo siguiente:

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Flash Player escribe el archivo SharedObject en el directorio /#localhost (o /localhost si la aplicación está fuera de línea). Esto resulta útil si se desea que varias aplicaciones en el cliente puedan acceder al mismo objeto compartido. En este caso, el cliente podría ejecutar dos aplicaciones de Flex, que especifican una ruta al objeto compartido que es la raíz del dominio; el cliente podría acceder al mismo objeto compartido desde ambas aplicaciones. Para compartir datos entre varias aplicaciones sin persistencia, puede utilizar el objeto LocalConnection.

Si especifica un directorio que no existe, Flash Player no crea ningún archivo SharedObject.

Cómo añadir datos a un objeto compartido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede añadir datos al archivo *.sol de objeto compartido SharedObject mediante la propiedad `data` del objeto SharedObject. Para añadir nuevos datos al objeto compartido, utilice la siguiente sintaxis:

```
sharedObject_name.data.variable = value;
```

En el siguiente ejemplo se añaden las propiedades `userName`, `itemNumbers` y `adminPrivileges` y sus valores a SharedObject:

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

Tras asignar valores a la propiedad `data`, debe indicar a Flash Player que escriba estos valores en el archivo de SharedObject. Para que Flash Player escriba los valores en el archivo de SharedObject, utilice el método `flush()` de`SharedObject`, del siguiente modo:

```
mySO.flush();
```

Si no se llama al método `SharedObject.flush()`, Flash Player escribe los valores en el archivo cuando se cierra la aplicación. No obstante, esto no proporciona al usuario ninguna oportunidad de aumentar el espacio disponible que Flash Player tiene para almacenar los datos si estos sobrepasan la configuración predeterminada. Por lo tanto, se recomienda llamar a `SharedObject.flush()`.

Cuando se utiliza el método `flush()` para escribir objetos compartidos en el disco duro de un usuario, es necesario comprobar si el usuario ha desactivado de forma explícita el almacenamiento local a través del Administrador de configuración de Flash Player (www.macromedia.com/support/documentation/es/flashplayer/help/settings_manager07.html), tal y como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

Almacenamiento de objetos en objetos compartidos

Es posible almacenar objetos sencillos, como Arrays o Strings, en una propiedad `data` de SharedObject.

El siguiente ejemplo muestra una clase de ActionScript que define métodos que controlan la interacción con el objeto compartido. Estos métodos permiten al usuario añadir y eliminar objetos del objeto compartido. Esta clase almacena una ArrayCollection que contiene objetos simples.

```
package {
    import mx.collections.ArrayCollection;
    import flash.net.SharedObject;

    public class LSOHandler {

        private var mySO:SharedObject;
        private var ac:ArrayCollection;
        private var lsoType:String;

        // The parameter is "feeds" or "sites".
        public function LSOHandler(s:String) {
            init(s);
        }

        private function init(s:String):void {
            ac = new ArrayCollection();
            lsoType = s;
            mySO = SharedObject.getLocal(lsoType);
            if (getObjects()) {
```

```
                ac = getObjects();
            }
        }

        public function getObjects():ArrayCollection {
            return mySO.data[lsoType];
        }

        public function addObject(o:Object):void {
            ac.addItem(o);
            updateSharedObjects();
        }

        private function updateSharedObjects():void {
            mySO.data[lsoType] = ac;
            mySO.flush();
        }
    }

}
```

La siguiente aplicación Flex crea una instancia de la clase de ActionScript para cada uno de los tipos de objetos compartidos que necesita. A continuación llama al método de esa clase cuando el usuario añade o elimina direcciones URL de sitios o blogs.

```
<?xml version="1.0"?>
<!-- lsos/BlogAggregator.mxml -->
<mx:Application
    xmlns:local="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initApp()"
    backgroundColor="#ffffff"
>
    <mx:Script>
        <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.utils.ObjectUtil;
        import flash.net.SharedObject;

        [Bindable]
        public var welcomeMessage:String;

        [Bindable]
        public var localFeeds:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var localSites:ArrayCollection = new ArrayCollection();

        public var lsofeeds:LSOHandler;
        public var lsosites:LSOHandler;

        private function initApp():void {
            lsofeeds = new LSOHandler("feeds");
            lsosites = new LSOHandler("sites");

            if (lsofeeds.getObjects()) {
                localFeeds = lsofeeds.getObjects();
```

```
            }
            if (lsosites.getObjects()) {
                localSites = lsosites.getObjects();
            }
        }

        // Adds a new feed to the feeds DataGrid.
        private function addFeed():void {
            // Construct an object you want to store in the
            // LSO. This object can contain any number of fields.
            var o:Object = {name:ti1.text, url:ti2.text, date:new Date()};
            lsofeeds.addObject(o);

            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method.
            localFeeds = lsofeeds.getObjects();

            // Clear the text fields.
            ti1.text = '';
            ti2.text = '';
        }

        // Removes feeds from the feeds DataGrid.
        private function removeFeed():void {
            // Use a method of ArrayCollection to remove a feed.
            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method. You do not need
            // to update it manually.
            if (myFeedsGrid.selectedIndex > -1) {

localFeeds.removeItemAt(myFeedsGrid.selectedIndex);
             }
        }

        private function addSite():void {
            var o:Object = {name:ti3.text, date:new Date()};
            lsosites.addObject(o);
            localSites = lsosites.getObjects();
            ti3.text = '';
        }

        private function removeSite():void {
            if (mySitesGrid.selectedIndex > -1) {

localSites.removeItemAt(mySitesGrid.selectedIndex);
            }
        }

        ]]>
    </mx:Script>

    <mx:Label text="Blog aggregator" fontSize="28"/>

    <mx:Panel title="Blogs">
        <mx:Form id="blogForm">
```

```
            <mx:HBox>
                <mx:FormItem label="Name:">
                    <mx:TextInput id="ti1" width="100"/>
                </mx:FormItem>
                <mx:FormItem label="Location:">
                    <mx:TextInput id="ti2" width="300"/>
                </mx:FormItem>
                <mx:Button id="b1" label="Add Feed" click="addFeed()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Feeds:">
                <mx:DataGrid
                    id="myFeedsGrid"
                    dataProvider="{localFeeds}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b2" label="Remove Feed" click="removeFeed()"/>
        </mx:Form>
    </mx:Panel>

    <mx:Panel title="Sites">
        <mx:Form id="siteForm">
            <mx:HBox>
                <mx:FormItem label="Site:">
                    <mx:TextInput id="ti3" width="400"/>
                </mx:FormItem>
                <mx:Button id="b3" label="Add Site" click="addSite()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Sites:">
                <mx:DataGrid
                    id="mySitesGrid"
                    dataProvider="{localSites}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b4" label="Remove Site" click="removeSite()"/>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

Almacenamiento de objetos con tipo en objetos compartidos

Se pueden almacenar instancias de ActionScript con tipo en objetos compartidos. Esto se lleva a cabo llamando al método `flash.net.registerClassAlias()` para registrar la clase. Si se crea una instancia de la clase y se almacena en el miembro de datos del objeto compartido y posteriormente se lee el objeto, se obtendrá una instancia con tipo. De forma predeterminada, la propiedad `objectEncoding` de SharedObject admite la codificación AMF3 y desempaqueta la instancia almacenada desde el objeto SharedObject; la instancia almacenada conserva el mismo tipo que se especificó cuando se llamó al método `registerClassAlias()`.

(Solo iOS) Impedir copia de seguridad en la nube de objetos compartidos locales

Adobe AIR 3.7 y posterior, solo iOS

Puede definir la propiedad `SharedObject.preventbackup` para controlar si se realizará una copia de seguridad de los objetos compartidos locales en el servicio de copia de seguridad en la nube de iOS. Apple lo exige así para contenido que puede volver a generarse o descargarse y es necesario para el funcionamiento correcto de la aplicación cuando se usa sin conexión.

Creación de varios objetos compartidos

Es posible crear varios objetos compartidos para la misma aplicación Flex. Para ello, cada uno de ellos se asigna a un nombre de instancia distinto, tal y como se muestra en el siguiente ejemplo:

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

Con esto se crean los archivos preferences.sol y history.sol en el directorio local de la aplicación Flex.

Creación de un objeto SharedObject seguro

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se crea una instancia de SharedObject local o remota mediante `getLocal()` o `getRemote()`, hay un parámetro opcional denominado `secure` que determina si el acceso a este objeto compartido queda restringido a los archivos SWF enviados a través de una conexión HTTPS. Si el valor de este parámetro es `true` y el archivo SWF se envía a través de HTTPS, Flash Player crea un nuevo objeto compartido seguro u obtiene una referencia a un objeto compartido seguro existente. Solo pueden leer o escribir en este objeto compartido seguro los archivos SWF enviados a través de HTTPS que llamen a `SharedObject.getLocal()` con el parámetro secure establecido en `true`. Si el valor de este parámetro es `false` y el archivo SWF se envía a través de HTTPS, Flash Player crea un nuevo objeto compartido u obtiene una referencia a un objeto compartido existente.

Los archivos SWF enviados mediante conexiones no HTTPS pueden leer o escribir en este objeto compartido. Si un archivo SWF se ha enviado a través de una conexión no HTTPS y se intenta establecer este parámetro en `true`, no se podrá crear un nuevo objeto compartido (o acceder a un objeto compartido seguro creado anteriormente), se emitirá un error y el objeto compartido se establecerá en `null`. Si se intenta ejecutar el siguiente fragmento de código desde una conexión no HTTPS, el método `SharedObject.getLocal()` emitirá un error:

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Independientemente del valor de este parámetro, los objetos compartidos creados aumentan el espacio en disco total permitido en el dominio.

Visualización del contenido de un objeto compartido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los valores se almacenan en los objetos compartidos dentro de la propiedad `data`. Puede repetir indefinidamente todos los valores de una instancia de objeto compartido utilizando un bucle `for..in`, tal y como se muestra en el siguiente ejemplo:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

Eliminación de objetos compartidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para eliminar un elemento SharedObject en el cliente, utilice el método `SharedObject.clear()`. Con esta operación no se eliminan los directorios en la ruta predeterminada para los objetos compartidos de la aplicación.

En el siguiente ejemplo se elimina el archivo SharedObject del cliente:

```
public function destroySharedObject():void {
    mySO.clear();
}
```

Ejemplo de SharedObject

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El siguiente ejemplo muestra cómo se almacenan objetos sencillos, como un objeto Date, en un objeto SharedObject sin tener que serializarlos ni deserializarlos manualmente.

El ejemplo siguiente comienza dándole la bienvenida como visitante. Al hacer clic en Log Out (Cerrar sesión), la aplicación almacena la fecha actual en un objeto compartido. La próxima vez que inicie esta aplicación o actualice la página, la aplicación le dará la bienvenida con un recordatorio de la hora en que cerró la sesión.

Para ver la aplicación en funcionamiento, haga clic en Log Out y después actualice la página. La aplicación muestra la fecha y la hora en la que hizo clic en el botón Log Out en su visita anterior. En cualquier momento, puede eliminar la información almacenada haciendo clic en el botón Delete LSO.

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

Almacenamiento local cifrado

La clase EncryptedLocalStore (ELS) proporciona un mecanismo de almacenamiento local cifrado que se puede utilizar como pequeña caché para los datos privados de una aplicación. Los datos de ELS no se pueden compartir entre aplicaciones. La finalidad de ELS es permitir que una aplicación pueda almacenar elementos que se facilitan frecuentemente, como las credenciales de inicio de sesión y otra información privada. Los datos ELS no se deben considerar como permanentes, tal y como se detalla en “Limitaciones del almacén local cifrado” y “Prácticas recomendadas” a continuación.

Nota: *además del almacén local cifrado, AIR también proporciona cifrado para el contenido almacenado en bases de datos SQL. Para obtener más información, consulte “Uso del cifrado con bases de datos SQL” en la página 790.*

El almacén local cifrado resulta práctico para guardar información que debe estar protegida, como los datos de inicio de sesión para servicios web. EL ELS está indicado para almacenar información que deba permanecer privada para otros usuarios. Sin embargo, no protege los datos de otros procesos ejecutados en la misma cuenta de usuario. Por esto, no está indicado para proteger datos secretos de aplicaciones, como DRM o claves de cifrado.

En plataformas de escritorio, AIR utiliza DPAPI en Windows, KeyChain en Mac OS e iOS, y KeyRing o KWallet en Linux para asociar el almacén local cifrado a todas las aplicaciones y usuarios. El almacén local cifrado utiliza cifrado AES-CBC de 128 bits.

En Android, los datos guardados con la clase EncryptedLocalStorage no están cifrados. En su lugar, los datos se protegen con la seguridad de usuario proporcionada por el sistema operativo. El sistema operativo Android asigna a cada aplicación un ID de usuario distinto. Las aplicaciones solo pueden acceder a sus propios archivos y a archivos creados en ubicaciones públicas (por ejemplo, una tarjeta de memoria extraíble). Tenga en cuenta que, en dispositivos Android "raíz", las aplicaciones que se ejecutan con privilegios de raíz PUEDEN acceder a archivos de otras aplicaciones. Así, en un dispositivo raíz, el almacenamiento local cifrado no proporciona un alto nivel de protección como sí lo hace en un dispositivo sin raíz.

La información en el almacén local cifrado solo está disponible para contenido de aplicaciones de AIR en el entorno limitado de seguridad de la aplicación.

Si actualiza una aplicación de AIR, la versión actualizada conserva el acceso a todos los datos existentes en el almacén local cifrado a no ser que:

- Los elementos se hayan añadido con el parámetro `stronglyBound` establecido en `true`
- La versión existente y la actualizada se publican en versiones anteriores a AIR 1.5.3 y la actualización está firmada con una firma de migración.

Limitaciones del almacén local cifrado

Los datos del almacén local cifrado se protegen con las credenciales de la cuenta en el sistema operativo del usuario. Otras entidades no pueden acceder a los datos del almacén a menos que puedan iniciar la sesión como usuario. Sin embargo, los datos no están seguros frente al acceso de otras aplicaciones ejecutadas por un usuario autenticado.

El usuario debe autenticarse para que funcionen estos ataques, por lo que los datos privados del usuario aún están protegidos (a no ser que la propia cuenta de usuario sea vulnerable). Sin embargo, la información que la aplicación pueda desear mantener en secreto para los usuarios, como las claves utilizadas para la administración de derechos digitales o de licencia, no está segura. Por lo tanto, el almacén local cifrado no es la ubicación adecuada para almacenar este tipo de información. Solo resulta un lugar apropiado para almacenar datos privados del usuario como, por ejemplo, contraseñas.

Los datos del ELS se pueden perder por muchos motivos. Por ejemplo, el usuario puede desinstalar la aplicación y eliminar el archivo cifrado. O el ID de edición podría cambiarse como resultado de la actualización. Por ello, el ELS debe tratarse como una memoria caché privada, no como espacio de almacenamiento de datos permanente.

El parámetro `stronglyBound` ya no se utiliza y no debe establecerse en `true`. Establecer el parámetro como `true` no aporta protección adicional a los datos. Al mismo tiempo, se pierde el acceso a los datos cada vez que se actualiza la aplicación, incluso si el ID de edición permanece invariable.

Es posible que el almacén local cifrado funcione más lento si los datos guardados superan los 10 MB.

Al desinstalar una aplicación de AIR, el programa de desinstalación no elimina los datos que tenga guardados el almacén local cifrado.

Prácticas recomendadas

Entre las mejores prácticas para el uso del almacén local cifrado (ELS) se incluyen:

- Utilice el ELS para almacenar datos importantes del usuario como, por ejemplo, contraseñas (estableciendo stronglyBound en false)
- No utilice el ELS para almacenar secretos de las aplicaciones como, por ejemplo, claves de DRM (administración de derechos digitales) o tokens de licencia..
- Permite que la aplicación pueda recrear los datos almacenados en el ELS si estos se pierden. Por ejemplo, puede pedir al usuario que vuelva a introducir sus credenciales de cuenta de usuario cuando sea necesario.
- No utilice el parámetro `stronglyBound`.
- Si establece `stronglyBound` como `true`, no migre los elementos almacenados durante una actualización. Vuelva a crear los datos una vez concluida la actualización.
- Almacene únicamente cantidades relativamente pequeñas de datos. Para cantidades más grandes, utilice una base de datos SQL de AIR con cifrado.

Más temas de ayuda

flash.data.EncryptedLocalStore

Cómo añadir datos al almacén local cifrado

Utilice el método estático `setItem()` de la clase EncryptedLocalStore para guardar datos en el almacén local. Los datos se guardan en una tabla hash, utilizando cadenas como claves, con los datos guardados en forma de conjuntos de bytes.

El código del ejemplo siguiente guarda una cadena en el almacén local cifrado:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes);
```

El tercer parámetro del método `setItem()`, el parámetro `stronglyBound`, es opcional. Cuando este parámetro se establece como `true`, el almacén local cifrado vincula el elemento guardado con los bits y la firma digital de la aplicación de AIR:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, false);
```

Para un elemento guardado con `stronglyBound` definido en `true`, las llamadas posteriores a `getItem()` solo tendrán éxito si la aplicación de AIR que llama es idéntica a la aplicación de almacenamiento (si no se ha cambiado ningún dato en los archivos del directorio de la aplicación). Si la aplicación de AIR que llama no es la misma que la que realiza el almacenamiento, la aplicación emite una excepción Error cuando se llama a `getItem()` para un elemento fuertemente vinculado. Si se actualiza la aplicación, esta no podrá leer datos fuertemente vinculados que se habían escrito en el almacén local cifrado. Se omite la definición de `stronglyBound` como `true` en dispositivos móviles; el parámetro siempre se trata como `false`.

Si el parámetro `stronglyBound` es establece en `false` (valor predeterminado), únicamente el ID de editor debe ser el mismo para que la aplicación lea los datos. Los bits de la aplicación pueden cambiar (y debe firmarlos el mismo editor), pero no es necesario que sean exactamente los mismos bits que estaban en la aplicación que almacenó los datos. Las aplicaciones actualizadas con el mismo ID de edición que el original pueden seguir accediendo a los datos.

Nota: *en la práctica, establecer* `stronglyBound` *como* `true` *no añade ninguna protección adicional a los datos. Un usuario "malintencionado" podría alterar la aplicación para tener acceso a los elementos guardados en el ELS. Además, los datos tienen el mismo nivel de protección frente a amenazas externas de usuarios ajenos si se establece* `stronglyBound` *como* `true` *o como* `false`*. Por estos motivos, establecer* `stronglyBound` *como* `true` *no se recomienda.*

Acceso a los datos del almacén local cifrado

Adobe AIR 1.0 y posterior

Se puede recuperar un valor del almacén local cifrado utilizando el método `EncryptedLocalStore.getItem()`, como en el siguiente ejemplo:

```
var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

Eliminación de los datos del almacén local cifrado

Adobe AIR 1.0 y posterior

Se puede eliminar un valor del almacén local cifrado utilizando el método `EncryptedLocalStore.removeItem()`, como en el siguiente ejemplo:

```
EncryptedLocalStore.removeItem("firstName");
```

Para borrar todos los datos del almacén local cifrado, llame al método `EncryptedLocalStore.removeItem()`, como en el siguiente ejemplo:

```
EncryptedLocalStore.reset();
```

Capítulo 40: Trabajo con bases de datos SQL locales en AIR

Adobe AIR 1.0 y posterior

Adobe® AIR® incluye la capacidad de crear y utilizar bases de datos SQL locales. El motor de ejecución incluye un motor de base de datos SQL compatible con muchas funciones estándar SQL, utilizando el sistema de base de datos SQLite de código abierto. Se puede utilizar una base de datos SQL local para almacenar datos localmente persistentes. Por ejemplo, se puede usar para datos de aplicación, parámetros de usuario de aplicación, documentos o cualquier otro tipo de datos que quiere que la aplicación guarde localmente.

Bases de datos SQL locales

Adobe AIR 1.0 y posterior

Para ver una explicación rápida y ejemplos de código del uso de las bases de datos SQL, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Working asynchronously with a local SQL database (Trabajo de forma asíncrona con bases de datos SQL locales, en inglés) (Flex)
- Working synchronously with a local SQL database (Trabajo de forma sincrónica con bases de datos SQL locales, en inglés) (Flex).
- Using an encrypted database (Uso de una base de datos cifrada, en inglés) (Flex).
- Working asynchronously with a local SQL database (Trabajo en forma asíncrona con bases de datos SQL locales, en inglés) (Flash).
- Working asynchronously with a local SQL database (Trabajo en forma asíncrona con bases de datos SQL locales, en inglés) (Flash).
- Using an encrypted database (Uso de una base de datos cifrada, en inglés) (Flash).

Adobe AIR incluye un motor de base de datos relacional basado en SQL que se ejecuta en tiempo de ejecución, con datos almacenados de forma local en archivos de base de datos en el equipo donde se ejecuta la aplicación de AIR (por ejemplo, en el disco duro). Dado que la base de datos ejecuta archivos de datos que se almacenan de forma local, una aplicación de AIR puede usar una base de datos independientemente si hay disponible una conexión de red. Por consiguiente, el motor de la base datos SQL local del motor de ejecución proporciona un mecanismo conveniente para almacenar datos de aplicación local persistentes, especialmente si tiene experiencia con SQL y bases de datos relacionales.

Usos de las bases de datos SQL locales

Adobe AIR 1.0 y posterior

La funcionalidad de la base de datos SQL local de AIR se puede usar para cualquier fin para el que desee almacenar datos de aplicación en el equipo local del usuario. Adobe AIR incluye varios mecanismos para almacenar datos de forma local, cada uno con diferentes ventajas. Los siguientes son algunos usos posibles para una base de datos SQL local en la aplicación de AIR:

- Para una aplicación orientada a datos (por ejemplo una agenda de direcciones) se puede usar una base de datos para almacenar los datos de la aplicación principal.
- Para una aplicación orientada a documentos, donde los usuarios crean documentos para guardar y posiblemente compartir, se puede guardar cada documento como un archivo de base de datos, en una ubicación designada por el usuario. (Sin embargo, se debe tener en cuenta que a no ser que la base de datos esté cifrada, cualquier aplicación de AIR podría abrir el archivo de base de datos. El cifrado se recomienda para documentos que pueden llegar a ser de especial importancia.)
- Para una aplicación de red, se puede usar una base de datos para almacenar caché local de los datos de aplicación o almacenar datos temporalmente cuando no se encuentra disponible una conexión de red. Se puede crear un mecanismo para sincronizar la base de datos local con el almacén de datos de red.
- Para cualquier aplicación, se puede usar una base de datos para almacenar la configuración de aplicación individual de un usuario, como opciones de usuario o información de aplicación como color y posición de la ventana.

Más temas de ayuda

Christophe Coenraets: Director de personal en AIR para Android

Raymond Camden: jQuery and AIR - Moving from web page to application (Movimiento desde la página web a la aplicación; en inglés)

Bases de datos y archivos de base de datos de AIR

Adobe AIR 1.0 y posterior

Una base de datos SQL local de Adobe AIR individual se almacena como un solo archivo en el sistema de archivos del equipo. El motor de ejecución incluye el motor de la base de datos SQL que gestiona la creación y estructura de los archivos de bases de datos y la manipulación y recuperación de datos de un archivo de base de datos. El motor de ejecución no especifica cómo ni dónde se almacenan los datos de la base de datos en el sistema de archivos; sino, cada base de datos se almacena completamente dentro de un único archivo. Se especifica la ubicación en el sistema de archivos donde se almacena el archivo de la base de datos. Una sola aplicación de AIR puede acceder a una o muchas bases de datos por separado (es decir, archivos de base de datos por separado). Dado que el motor de ejecución almacena cada base de datos como un único archivo en el sistema de archivos, se puede encontrar la base de datos según sea necesario por el diseño de la aplicación y las restricciones de acceso a los archivos del sistema operativo. Cada usuario puede tener un archivo de base de datos individual para sus datos específicos o todos los usuarios de la aplicación pueden acceder a un archivo de base de datos en un solo equipo para datos compartidos. Dado que los datos son locales para un solo equipo, los datos no se comparten automáticamente entre usuarios en diferentes equipos. El motor de la base de datos SQL local no proporciona ninguna prestación para ejecutar declaraciones SQL para comparar con una base de datos remota o de servidor.

Bases de datos relacionales

Adobe AIR 1.0 y posterior

Una base de datos relacional es un mecanismo para almacenar (y recuperar) datos en un equipo. Los datos se organizan en tablas: las filas representan registros o elementos y las columnas (a veces denominados "campos") dividen cada registro en valores individuales. Por ejemplo, una aplicación de agenda de direcciones puede tener una tabla "amigos". Cada fila en la tabla representa un amigo almacenado en la base de datos. Las columnas de la tabla representan los datos como nombre, apellido, fecha de nacimiento etc. Para cada fila de amigo en la tabla, la base de datos almacena un valor por separado para cada columna.

Las bases de datos relacionales están diseñadas para almacenar datos complejos, donde un elemento está asociado o relacionado con elementos de otro tipo. En una base de datos relacional, los datos que tienen una relación de uno a muchos —donde un solo registro puede relacionarse con múltiples registros de un tipo diferente— se debe dividir entre diferentes tablas. Por ejemplo, supongamos que desea que la aplicación de la agenda de direcciones almacene múltiples números de teléfono para cada amigo, esta es una relación de uno a muchos. La tabla "amigos" contiene toda la información personal de cada amigo. Una tabla por separado de "números de teléfono" tiene los números de teléfono de todos los amigos.

Además de almacenar los datos sobre amigos y números de teléfono, cada tabla necesita datos para hacer un seguimiento de la relación entre las dos tablas, para hacer coincidir los registros individuales de los amigos con sus números de teléfono. Estos datos se conocen como clave principal, un identificador exclusivo que distingue cada fila en una tabla de otras filas en dicha tabla. La clave principal puede ser una "clave natural", lo que significa que es uno de los elementos de los datos que naturalmente distingue cada registro en una tabla. En la tabla "amigos", si supiera que ninguno de sus amigos tienen la misma fecha de nacimiento, podría usar la columna de fechas de nacimiento como la clave principal (una clave natural) de la tabla "amigos". Si no hay ninguna clave natural, debe crear una columna de clave principal por separado como "ID de amigo", un valor artificial que usa la aplicación para distinguir entre filas.

Al usar una clave principal, se pueden configurar las relaciones entre múltiples tablas. Por ejemplo, supongamos que la tabla "amigos" tiene una columna "ID de amigo" que contiene un número exclusivo para cada fila (cada amigo). La tabla "números de teléfono" relacionada se puede estructurar con dos columnas: una con el "ID de amigo" del amigo al que le pertenece el número de teléfono y una con el número de teléfono real. De ese modo, no importa la cantidad de números de teléfono que tenga un amigo, se pueden almacenar todos en la tabla "números de teléfono" y se pueden vincular al amigo relacionado usando la clave principal "ID de amigo". Cuando se usa una clave principal de una tabla en una tabla relacionada para especificar la conexión entre los registros, el valor en la tabla relacionada se conoce como clave externa. A diferencia de muchas bases de datos, el motor de base de datos local de AIR no permite crear restricciones para la clave externa, que son restricciones que verifican automáticamente que el valor de una clave externa insertada o actualizada tiene una fila correspondiente en la tabla de la clave principal. No obstante, las relaciones de las claves externas son una parte importante de la estructura de una base de datos relacional y las claves externas se deben usar cuando se crean relaciones entre tablas en la base de datos.

Conceptos de SQL

Adobe AIR 1.0 y posterior

Structured Query Language, SQL (del inglés Lenguaje de consulta estructurado) se utiliza con bases de datos relacionales para manipular y recuperar datos. SQL es un lenguaje descriptivo en vez de un lenguaje de procedimientos. En vez de impartir instrucciones al equipo sobre cómo debería recuperar datos, una declaración SQL describe el conjunto de datos que necesita. El motor de base de datos determina la manera de recuperar esos datos.

El lenguaje SQL ha sido estandarizado por el American National Standards Institute, ANSI (del inglés Instituto Nacional Estadounidense de Normas). La base de datos SQL local de Adobe AIR admite la mayoría de las normas SQL-92.

Para obtener descripciones específicas del lenguaje SQL admitido en Adobe AIR, consulte "Compatibilidad de SQL en bases de datos locales" en la página 1154.

Clases de bases de datos SQL

Adobe AIR 1.0 y posterior

Para trabajar con bases de datos SQL locales en ActionScript 3.0, se usan las instancias de estas clases en el paquete flash.data:

Clase	Descripción
flash.data.SQLConnection	Proporciona los medios para crear y abrir bases de datos (archivos de base de datos) así como métodos para realizar operaciones a nivel de base de datos y para controlar las transacciones de bases de datos.
flash.data.SQLStatement	Representa una declaración SQL individual (una sola consulta o comando) que se ejecuta en una base de datos, incluyendo la definición del texto de la instrucción y la configuración de los valores de los parámetros.
flash.data.SQLResult	Proporciona una manera de obtener información o resultados de la ejecución de una declaración, como las filas resultantes de una declaración `SELECT`, el número de filas afectadas por una declaración `UPDATE` o `DELETE` y así sucesivamente.

Para obtener la información de esquemas que describen la estructura de una base de datos, se usan estas clases en el paquete flash.data:

Clase	Descripción
flash.data.SQLSchemaResult	Actúa como un contenedor para los resultados de esquema de la base de datos generados al llamar al método `SQLConnection.loadSchema()`.
flash.data.SQLTableSchema	Proporciona información que describe un sola tabla en una base de datos.
flash.data.SQLViewSchema	Proporciona información que describe un sola vista en una base de datos.
flash.data.SQLIndexSchema	Proporciona información que describe una sola columna de una tabla o vista en una base de datos.
flash.data.SQLTriggerSchema	Proporciona información que describe un solo desencadenador en una base de datos.

Otras clases en el paquete flash.data proporcionan restricciones que se usan con la clase SQLConnection y la clase SQLColumnSchema:

Clase	Descripción
flash.data.SQLMode	Define un conjunto de constantes que representan los valores posibles del parámetro `openMode` de los métodos `SQLConnection.open()` y `SQLConnection.openAsync()`.
flash.data.SQLColumnNameStyle	Define un conjunto de constantes que representan los valores posibles de la propiedad `SQLConnection.columnNameStyle`.
flash.data.SQLTransactionLockType	Define un conjunto de constantes que representan los valores posibles del parámetro de opción del método `SQLConnection.begin()`.
flash.data.SQLCollationType	Define un conjunto de constantes que representan los valores posibles de la propiedad `SQLColumnSchema.defaultCollationType` y el parámetro `defaultCollationType` del constructor `SQLColumnSchema()`.

Además, las siguientes clases en el paquete flash.events representan los eventos (y constantes admitidas) que usa:

Clase	Descripción
flash.events.SQLEvent	Define los eventos que una instancia SQLConnection o SQLStatement distribuye cuando cualquiera de sus operaciones se ejecuta correctamente. Cada operación tiene una constante de tipo de evento asociada definida en la clase SQLEvent.
flash.events.SQLErrorEvent	Define el evento que una instancia SQLConnection o SQLStatement distribuye cuando cualquiera de sus operaciones resulta en error.
flash.events.SQLUpdateEvent	Define el evento que una instancia SQLConnection distribuye cuando los datos de la tabla en una de las bases de datos conectadas cambia como resultado de la ejecución de una declaración SQL `INSERT`, `UPDATE` o `DELETE`.

Por último, las siguientes clases en el paquete flash.errors proporcionan información sobre los errores de operación de la base de datos:

Clase	Descripción
flash.errors.SQLError	Proporciona información sobre un error de operación de la base de datos, incluida la operación que se estaba intentando realizar y la razón del error.
flash.errors.SQLErrorOperation	Define un conjunto de constantes que representa los valores posibles para la propiedad `operation` de la clase SQLError, que indica la operación de la base de datos que resultó en error.

Modos de ejecución sincrónicos y asíncronos

Adobe AIR 1.0 y posterior

Cuando se escribe código para trabajar con una base de datos SQL local, se especifica la ejecución de las operaciones de la base de datos en uno de los dos modos de ejecución: modo de ejecución asíncrono o sincrónico. En general, los ejemplos de código muestran la manera de realizar cada operación en ambos modos, para que pueda usar el ejemplo que sea más apropiado para sus necesidades.

En el modo de ejecución asíncrono, se suministra una instrucción al motor de ejecución y este distribuye un evento cuando la operación solicitada se completa o falla. Primero se indica al motor de la base de datos que realice una operación. El motor de la base de datos hace su trabajo en segundo plano mientras la aplicación continúa ejecutándose. Por último, cuando se completa la operación (o cuando falla) el motor de la base de datos distribuye un evento. El código, activado por el evento, lleva a cabo las operaciones subsiguientes. Este enfoque tiene una gran ventaja: el tiempo motor ejecución realiza las operaciones de la base de datos en segundo plano mientras el código de la aplicación principal continúa ejecutándose. Si la operación de la base de datos tarda considerablemente, la aplicación continúa ejecutándose. Lo más importante es que el usuario puede seguir interactuando sin que se bloquee la pantalla. No obstante, el código de operación asíncrono puede ser más complejo de escribir que otro código. Esta complejidad es generalmente en casos donde varias operaciones dependientes se deben dividir entre diferentes métodos de detectores de evento.

Conceptualmente, es más fácil codificar operaciones como una sola secuencia de pasos, un conjunto de operaciones sincrónicas, en vez de un conjunto de operaciones divididas entre varios métodos de detectores de evento. Además de las operaciones de base de datos asíncronas, Adobe AIR también permite ejecutar operaciones de base de datos sincrónicas. En el modo de ejecución sincrónico, las operaciones no se ejecutan en segundo plano. En cambio se ejecutan en la misma secuencia de ejecución que el resto del código de aplicación. Se le indica al motor de la base de datos que realice una operación. Este código hace una pausa en ese punto mientras que el motor de base de datos hace su trabajo. Cuando se completa la operación, la ejecución continúa con la siguiente línea de código.

Si las operaciones se ejecutan de forma asíncrona o sincrónicamente se define en el nivel SQLConnection. Si se usa una sola conexión de base de datos, no se pueden ejecutar algunas operaciones o sentencias sincrónicamente y otras de forma asíncrona. Especifique si una SQLConnection funciona en el modo de ejecución asíncrono o sincrónico llamando a un método SQLConnection para abrir la base de datos. Si llama a `SQLConnection.open()` la conexión funciona en el modo de ejecución sincrónico y si llama a `SQLConnection.openAsync()` la conexión funciona en el modo de ejecución asíncrono. Una vez que una instancia SQLConnection se conecta a una base de datos usando `open()` o `openAsync()`, se fija en el modo de ejecución asíncrono o sincrónico a menos que elija cerrar y volver a abrir la conexión a la base de datos.

Cada modo de ejecución tiene sus ventajas. Mientras que la mayoría de los aspectos son similares, hay algunas diferencias que debe tener en cuenta cuando trabaja con cada modo. Para más información sobre estos temas y sugerencias para trabajar en cada modo, consulte "Uso de operaciones sincrónicas y asíncronas de base de datos" en la página 785.

Creación y modificación de una base de datos

Adobe AIR 1.0 y posterior

Antes de que la aplicación pueda añadir o recuperar datos, debe existir una base de datos con tablas definidas en la misma a la que puede acceder la aplicación. A continuación se describen las tareas para crear una base de datos y para crear la estructura de datos dentro de una base de datos. Mientras que estas tareas se usan con menos frecuencia que la inserción y recuperación de datos, son necesarias para la mayoría de las aplicaciones.

Más temas de ayuda

Mind the Flex: Updating an existing AIR database (Flex: actualización de una base de datos de AIR existente; en inglés)

Creación de una base de datos

Adobe AIR 1.0 y posterior

Para crear un archivo de base de datos, primero debe crear una instancia de SQLConnection. Se llama al método `open()` para abrirla en el modo de ejecución sincrónico o al método `openAsync()` para abrirla en el modo de ejecución asíncrono. Los métodos `open()` y `openAsync()` se usan para abrir una conexión a una base de datos. Si pasa una instancia File que se refiere a una ubicación de archivo no existente para el parámetro `reference` (el primer parámetro), el método `open()` o `openAsync()` crea un archivo de base de datos en esa ubicación de archivo y abre una conexión a la base de datos recientemente creada.

Independientemente si llama al método `open()` o al método `openAsync()` para crear una base de datos, el nombre de archivo de la base de datos puede ser cualquier nombre de archivo válido, con cualquier extensión de nombre de archivo. Si llama al método `open()` o `openAsync()` con el valor `null` para el parámetro `reference`, se crea una nueva base de datos en memoria en vez de un archivo de base de datos en el disco.

El siguiente ejemplo de códigos muestra el proceso de creación de un archivo de base de datos (una nueva base de datos) usando el modo de ejecución asíncrono. En este caso, el archivo de base de datos se guarda en el "Apuntar al directorio de almacenamiento de la aplicación" en la página 700, con el nombre de archivo "DBSample.db":

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database was created successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Nota: *aunque la clase File permite señalar una ruta de archivos nativos específica, esto puede implicar que las aplicaciones no funcionen correctamente en diversas plataformas. Por ejemplo, la ruta C:\Documents and Settings\joe\test.db únicamente funciona en Windows. Por estos motivos, resulta más adecuado usar propiedades estáticas de la clase File como, por ejemplo,* `File.applicationStorageDirectory`*, así como el método* `resolvePath()` *(tal y como se muestra en el ejemplo anterior). Para obtener más información, consulte "Rutas a objetos File" en la página 696.*

Para ejecutar las operaciones sincrónicamente, cuando abre una conexión de base de datos con la instancia SQLConnection, llame al método `open()`. En el siguiente ejemplo se muestra la manera de crear y abrir una instancia SQLConnection que ejecuta las operaciones sincrónicamente:

```
import flash.data.SQLConnection;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile);
                    trace("the database was created successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Creación de tablas en la base de datos

Adobe AIR 1.0 y posterior

La creación de una tabla en una base de datos implica ejecutar una declaración SQ en dicha base de datos, usando el mismo proceso que se utiliza para ejecutar una declaración SQL, como `SELECT`, `INSERT`, etc. Para crear una tabla, se usa una declaración `CREATE TABLE`, que incluye definiciones de columnas y restricciones para la nueva tabla. Para más información sobre la ejecución de declaraciones SQL, consulte "Trabajo con declaraciones SQL" en la página 760.

En el siguiente ejemplo se demuestra la creación de una tabla denominada "employees" en un archivo de base de datos existente, usando el modo de ejecución asíncrono. Observe que en este código se supone que existe una instancia de SQLConnection denominada `conn` que ya se ha creado y está conectada a una base de datos.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(SQLEvent.RESULT, createResult);
createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event:SQLEvent):void
{
    trace("Table created");
}

function createError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                createStmt.addEventListener(SQLEvent.RESULT, createResult);
                createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

                createStmt.execute();
            }

            private function createResult(event:SQLEvent):void
            {
                trace("Table created");
            }

            private function createError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

En el siguiente ejemplo se demuestra la creación de una tabla denominada “employees” en un archivo de base de datos existente, usando el modo de ejecución sincrónico. Observe que en este código se supone que existe una instancia de SQLConnection denominada `conn` que ya se ha creado y está conectada a una base de datos.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                try
                {
                    createStmt.execute();
                    trace("Table created");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Manipulación de los datos de una base de datos SQL

Adobe AIR 1.0 y posterior

Hay algunas tareas comunes que se realizan cuando se utilizan bases de datos SQL locales. Estas tareas incluyen la conexión a una base de datos, añadir datos a tablas y recuperar datos de las tablas en una base de datos. Asimismo, hay varios puntos que debe tener en cuenta al realizar estas tareas, como la utilización de tipos de datos y la gestión de errores.

Observe que también hay varias tareas de la base de datos que deberá ocuparse con menos frecuencia, pero que se deben realizar antes de que pueda llevar a cabo estas tareas más comunes. Por ejemplo, antes de que se pueda conectar a una base de datos y recuperar los datos de una tabla, necesitará crear la base de datos y crear la estructura de la tabla en la base de datos. Esas tareas iniciales menos frecuentes se describen en la sección "Creación y modificación de una base de datos" en la página 749.

Puede elegir realizar operaciones de base de datos de forma asíncrona, es decir, el motor de la base de datos se ejecuta en segundo plano y le notifica cuando la operación se completa correctamente o falla distribuyendo un evento. Asimismo puede realizar estas operaciones sincrónicamente. En ese caso las operaciones de la base de datos se realizan una después de la otra y toda la aplicación (incluyendo las actualizaciones de la pantalla) esperan a que se completen las operaciones antes de ejecutar otro código. Para más información sobre el uso del modo de ejecución sincrónico o asíncronos, consulte "Uso de operaciones sincrónicas y asíncronas de base de datos" en la página 785.

Conexión a una base de datos

Adobe AIR 1.0 y posterior

Antes de que pueda realizar cualquier operación en la base de datos, primero abra una conexión al archivo de la base de datos. Se usa una instancia de SQLConnection para representar una conexión a una o varias bases de datos. La primera base de datos que se conecta usando una instancia SQLConnection se denomina la base de datos "principal". Esta base de datos se conecta usando el método `open()` (para el modo de ejecución sincrónico) o el método `openAsync()` (para el modo de ejecución asíncrono).

Si abre una base de datos usando la operación asíncrona `openAsync()`, se debe registrar para el evento `open` de la instancia SQLConnection para saber cuándo se completa la operación `openAsync()`. Regístrese para el evento `error` de la instancia SQLConnection para determinar si la operación falla.

En el siguiente ejemplo se muestra cómo abrir un archivo de base de datos existente para la ejecución asíncrona. El archivo de la base de datos se denomina "DBSample.db" y se encuentra en el "Apuntar al directorio de almacenamiento de la aplicación" en la página 700.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, SQLMode.UPDATE);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile, SQLMode.UPDATE);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database opened successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

En el siguiente ejemplo se muestra cómo abrir un archivo de base de datos existente para la ejecución sincrónica. El archivo de la base de datos se denomina “DBSample.db” y se encuentra en el “Apuntar al directorio de almacenamiento de la aplicación” en la página 700.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, SQLMode.UPDATE);
    trace("the database opened successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile, SQLMode.UPDATE);
                    trace("the database opened successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Observe que en la llamada del método `openAsync()` en el ejemplo asíncrono y en la llamada del método `open()` en el ejemplo sincrónico, el segundo argumento es la constante `SQLMode.UPDATE`. La especificación de `SQLMode.UPDATE` para el segundo parámetro (`openMode`) hace que el motor de ejecución distribuya un error si el archivo especificado no existe. Si pasa un valor `SQLMode.CREATE` para el parámetro `openMode` (o si desactiva el parámetro `openMode`), el motor de ejecución intenta crear un archivo de base de datos si el archivo especial no existe. No obstante, si el archivo existe está abierto, que es lo mismo que utilizar `SQLMode.Update`. Asimismo puede especificar `SQLMode.READ` para el parámetro `openMode` para abrir una base de datos existente en el modo de solo lectura. En ese caso, se pueden recuperar los datos de la base de datos pero no se pueden añadir, eliminar ni cambiar datos.

Trabajo con declaraciones SQL

Adobe AIR 1.0 y posterior

Una declaración SQL individual (una consulta o comando) está representada en el motor de ejecución como objeto SQLStatement. Siga estos pasos para crear y ejecutar una declaración SQL:

Crear una instancia SQLStatement.
El objeto SQLStatement representa la declaración SQL en su aplicación.

```
var selectData:SQLStatement = new SQLStatement();
```

Especificar la base de datos donde se realiza la consulta.
Para ello, defina la propiedad `sqlConnection` del objeto SQLStatement a la instancia SQLConnection que está conectada con la base de datos deseada.

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

Especificar la declaración SQL real.
Cree el texto de la declaración como una cadena y asígnela a la propiedad `text` de la instancia SQLStatement.

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

Definir las funciones para gestionar el resultado de la operación de ejecución (solo modo de ejecución asíncrono)
Use el método `addEventListener()` para registrar funciones como detectores para los eventos `result` y `error` de la instancia SQLStatement.

```
// using listener methods and addEventListener()

selectData.addEventListener(SQLEvent.RESULT, resultHandler);
selectData.addEventListener(SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event:SQLEvent):void
{
    // do something after the statement execution succeeds
}

function errorHandler(event:SQLErrorEvent):void
{
    // do something after the statement execution fails
}
```

Como alternativa, puede especificar métodos de detectores usando un objeto Responder. En ese caso, cree la instancia Responder y vincule los métodos de detector a la misma.

```
// using a Responder (flash.net.Responder)

var selectResponder = new Responder(onResult, onError);

function onResult(result:SQLResult):void
{
    // do something after the statement execution succeeds
}

function onError(error:SQLError):void
{
    // do something after the statement execution fails
}
```

Si el texto de la declaración incluye definiciones de parámetro, asigne valores para esos parámetros.
Para asignar valores de parámetro, use la propiedad de conjunto asociativa `parameters` de la instancia SQLStatement.

```
selectData.parameters[":param1"] = 25;
```

Ejecutar la declaración SQL.
Llame la método `execute(()` de la instancia SQLStatement.

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

Además, si está usando Responder en vez de detectores de eventos en el modo de ejecución asíncrono, pase la instancia Responder al método `execute(()`.

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

Para ejemplos específicos que demuestran estos pasos, consulte los siguientes temas:

"Recuperación de datos de una base de datos" en la página 764

"Inserción de datos" en la página 774

"Cambio o eliminación de datos" en la página 780

Uso de parámetros en sentencias

Adobe AIR 1.0 y posterior

Una parámetro de declaración SQL permite crear una declaración SQL reutilizable. Cuando usa parámetros de declaración, los valores de la declaración pueden cambiar (como valores que se añaden en una declaración `INSERT`), pero el texto básico de la declaración no cambia. Por lo tanto, el uso de parámetros proporciona ventajas de rendimiento y facilita la codificación de una aplicación.

Aspectos básicos de parámetros de declaración

Adobe AIR 1.0 y posterior

Con frecuencia, una aplicación usa una sola declaración SQL varias veces en una aplicación, con leves variaciones. Por ejemplo, considere una aplicación de seguimiento de inventario donde un usuario puede añadir nuevos elementos de inventario a la base de datos. El código de aplicación que añade un elemento de inventario a la base de datos ejecuta una declaración SQL `INSERT` que añade los datos a la base de datos. Sin embargo, cada vez que se ejecuta la declaración hay una leve variación. En concreto, los valores reales que se insertan en la tabla son diferentes porque son específicos al elemento de inventario que se añade.

En los casos donde se tiene una declaración SQL que se usa múltiples veces con diferentes valores en la declaración, el mejor método es usar una declaración SQL que incluye parámetros en lugar de valores literales en el texto SQL. Un parámetro es un marcador de posición en el texto de la declaración que se remplaza con un valor real cada vez que se ejecuta la declaración. Para utilizar parámetros en una declaración SQL, se crea una instancia de SQLStatement, como es habitual. Para la declaración SQL real asignada a la propiedad `text`, use los marcadores de posición de parámetros en vez de valores literales. Luego defina el valor para cada parámetro configurando el valor de un elemento en la propiedad `parameters` de la instancia SQLStatement. La propiedad `parameters` es un conjunto asociativo, por lo que define un valor determinado usando la siguiente sintaxis:

```
statement.parameters[parameter_identifier] = value;
```

El valor *parameter_identifier* es una cadena si está usando un parámetro con nombre o un índice de número entero si está usando un parámetro sin nombre.

Uso de parámetros con nombre

Adobe AIR 1.0 y posterior

Un parámetro puede ser un parámetro con nombre. Un parámetro con nombre tiene un nombre específico que la base de datos usa para corresponder el valor del parámetro con la ubicación del marcador de posición en el texto de la declaración. Un nombre de parámetro contiene de un carácter ":" o "@" seguido por el nombre, como en los siguientes ejemplos:

```
:itemName
@firstName
```

El siguiente ejemplo de código demuestra el uso de parámetros con nombre:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

Uso de parámetros sin nombre

Adobe AIR 1.0 y posterior

Como una alternativa al uso de parámetros con nombre, también se pueden utilizar parámetros sin nombre. Para usar un parámetro sin nombre se indica un parámetro en una declaración SQL usando un signo de interrogación "?" . A cada parámetro se le asigna un índice numérico, según el orden de los parámetros en la declaración, comenzando con el índice 0 para el primer parámetro. En el siguiente ejemplo se muestra una versión del ejemplo anterior, usando parámetros sin nombre:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

Ventajas de utilizar parámetros

Adobe AIR 1.0 y posterior

El uso de parámetros en una declaración SQL proporciona varias ventajas:

Mejor rendimiento Una instancia SQLStatement que utiliza parámetros puede ejecutarse más eficazmente comparada con una que crea dinámicamente el texto SQL cada vez que se ejecuta. La mejora del rendimiento se debe a que la declaración se prepara una sola vez y se puede ejecutar múltiples veces usando diferentes valores de parámetro, sin tener que volver a compilar la declaración SQL.

Introducción de datos explícita Se utilizan los parámetros para permitir la sustitución de valores introducidos que se desconocen en el momento de la construcción de la declaración SQL. La utilización de parámetros es la única manera de garantizar la clase de almacenamiento para un valor pasado en la base de datos. Cuando no se utilizan parámetros, el motor de ejecución intenta convertir todos los valores de la representación de texto a una clase de almacenamiento en la afinidad de tipo de la columna asociada.

Para obtener más información sobre las clases de almacenamiento y la afinidad de columnas, consulte "Compatibilidad de tipos de datos" en la página 1177.

Mayor seguridad El uso de los parámetros ayuda a prevenir la ejecución de una técnica malintencionada conocida como ataque de inyección SQL. En un ataque de inyección SQL, un usuario introduce un código SQL en una ubicación accesible al usuario (por ejemplo, un campo de introducción de datos) Si el código de aplicación crea una declaración SQL directamente concatenando entradas del usuario en el texto SQL, el código SQL introducido por el usuario se ejecuta con la base de datos. A continuación se muestra un ejemplo de entradas del usuario concatenadas en el texto SQL. **No utilice esta técnica**:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;
```

La utilización de parámetros de instrucción en lugar de valores introducidos por el usuario concatenados en un texto de declaración impide un ataque de inyección SQL. La inyección SQL no se puede llevar a cabo porque los valores de los parámetros se tratan explícitamente como valores sustituidos, en lugar de ser parte del texto de la declaración literal. La siguiente alternativa es la alternativa que se recomienda del ejemplo anterior:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

Recuperación de datos de una base de datos

Adobe AIR 1.0 y posterior

La recuperación de datos de una base de datos consiste de dos pasos. Primero, debe ejecutar una declaración SQL `SELECT`, que describe el conjunto de datos que desea de la base de datos. A continuación, accede a los datos recuperados y los muestra o manipula según sea necesario para la aplicación.

Ejecución de una declaración SELECT

Adobe AIR 1.0 y posterior

Para recuperar datos existentes de una base de datos, se utiliza una instancia de SQLStatement. Asigne la declaración `SELECT` a la propiedad `text` de la instancia y llame al método `execute`.

Para conocer la sintaxis de la declaración `SELECT`, consulte “Compatibilidad de SQL en bases de datos locales” en la página 1154.

En el siguiente ejemplo se muestra la ejecución de una declaración `SELECT` para recuperar datos de la tabla denominada "products", utilizando el modo de ejecución asíncrono:

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}

function errorHandler(event:SQLErrorEvent):void
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
```

```
            selectStmt.execute();
        }

        private function resultHandler(event:SQLEvent):void
        {
            var result:SQLResult = selectStmt.getResult();

            var numResults:int = result.data.length;
            for (var i:int = 0; i < numResults; i++)
            {
                var row:Object = result.data[i];
                var output:String = "itemId: " + row.itemId;
                output += "; itemName: " + row.itemName;
                output += "; price: " + row.price;
                trace(output);
            }
        }

        private function errorHandler(event:SQLErrorEvent):void
        {
            // Information about the error is available in the
            // event.error property, which is an instance of
            // the SQLError class.
        }
    ]]>
  </mx:Script>
</mx:WindowedApplication>
```

En el siguiente ejemplo se muestra la ejecución de una declaración `SELECT` para recuperar datos de la tabla denominada "products", utilizando el modo de ejecución sincrónica:

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}
catch (error:SQLError)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                try
                {
                    selectStmt.execute();

                    var result:SQLResult = selectStmt.getResult();
```

```
                var numResults:int = result.data.length;
                for (var i:int = 0; i < numResults; i++)
                {
                    var row:Object = result.data[i];
                    var output:String = "itemId: " + row.itemId;
                    output += "; itemName: " + row.itemName;
                    output += "; price: " + row.price;
                    trace(output);
                }
            }
            catch (error:SQLError)
            {
                // Information about the error is available in the
                // error variable, which is an instance of
                // the SQLError class.
            }
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

En el modo de ejecución asíncrono, cuando se termina de ejecutar la declaración, la instancia SQLStatement distribuye un evento `result` (`SQLEvent.RESULT`) que indica que la declaración se ejecutó correctamente. Como alternativa, si se transmite un objeto Responder como argumento al método `execute()`, se llama a la función de control de resultados del objeto Responder. En el modo de ejecución sincrónico, la ejecución hace una pausa hasta que la operación `execute()` termina, y continúa en la siguiente línea de código.

Acceso a los datos del resultado de la declaración SELECT

Adobe AIR 1.0 y posterior

Una vez que la declaración `SELECT` se ha terminado de ejecutar, el siguiente paso es acceder a los datos que se recuperaron. Los datos de resultados se recuperan de la ejecución de una declaración `SELECT` llamando al método `getResult()` del objeto SQLStatement:

```
var result:SQLResult = selectStatement.getResult();
```

El método `getResult()` devuelve un objeto SQLResult. La propiedad `data` del objeto SQLResult es un conjunto que contiene los resultados de la declaración `SELECT`:

```
var numResults:int = result.data.length;
for (var i:int = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row:Object = result.data[i];
}
```

Cada fila de datos del resultado `SELECT` se convierte en una instancia de Object incluida en el conjunto `data`. Dicho objeto tiene propiedades cuyos nombres coinciden con los nombres de columna del conjunto de resultados. Las propiedades contienen los valores de las columnas del conjunto de resultados. Por ejemplo, supongamos que una declaración `SELECT` especifica un conjunto de resultados con tres columnas llamadas “itemId,” “itemName” y “price.” Para cada fila en el conjunto de resultados, se crea una instancia Object con propiedades denominadas `itemId`, `itemName` y `price`. Esas propiedades contienen los valores de sus respectivas columnas.

El siguiente ejemplo de código define una instancia SQLStatement cuyo texto es una instrucción `SELECT`. La declaración recupera las filas que contiene los valores de la columna `firstName` y `lastName` de todas las filas de una tabla denominada `employees`. Este ejemplo utiliza el modo de ejecución asíncrono. Cuando se termina la ejecución, se llama al método `selectResult()`, y se accede a las filas de datos resultantes usando `SQLStatement.getResult()` y se muestran usando el método `trace()`. Observe que este código supone que existe una instancia de SQLConnection denominada `conn` que ya ha sido creada y ya está conectada a una base de datos. Asimismo, supone que la tabla "employees" ya ha sido creada y llenada con datos.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event:SQLEvent):void
{
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                // register listeners for the result and error events
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

                // execute the statement
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void
            {
                // access the result data
                var result:SQLResult = selectStmt.getResult();

                var numRows:int = result.data.length;
                for (var i:int = 0; i < numRows; i++)
                {
                    var output:String = "";
                    for (var columnName:String in result.data[i])
                    {
                        output += columnName + ": " + result.data[i][columnName] + "; ";
                    }
                    trace("row[" + i.toString() + "]\t", output);
                }
            }

            private function selectError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

El siguiente ejemplo de código demuestra las mismas técnicas como la anterior pero utiliza el modo de ejecución sincrónico. El ejemplo define una instancia de SQLStatement cuyo texto es una instancia `SELECT`. La declaración recupera las filas que contiene los valores de la columna `firstName` y `lastName` de todas las filas de una tabla denominada `employees`. Se accede a las filas de datos resultantes utilizando `SQLStatement.getResult()` y se muestran usando el método `trace()`. Observe que este código supone que existe una instancia de SQLConnection denominada `conn` que ya ha sido creada y ya está conectada a una base de datos. Asimismo, supone que la tabla "employees" ya ha sido creada y llenada con datos.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();

    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                try
                {
                    // execute the statement
                    selectStmt.execute();

                    // access the result data
                    var result:SQLResult = selectStmt.getResult();

                    var numRows:int = result.data.length;
                    for (var i:int = 0; i < numRows; i++)
                    {
                        var output:String = "";
                        for (var columnName:String in result.data[i])
                        {
                            output += columnName + ": ";
                            output += result.data[i][columnName] + "; ";
                        }
                        trace("row[" + i.toString() + "]\t", output);
                    }
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Definición de tipos de datos de los datos del resultado SELECT

Adobe AIR 1.0 y posterior

De forma predeterminada, cada fila devuelta por una declaración `SELECT` se crea como una instancia Object con propiedades denominadas para los nombres de columna del conjunto de resultados y con el valor de cada columna como el valor de la propiedad asociada. Sin embargo, antes de ejecutar una declaración `SELECT` SQL puede definir la propiedad `itemClass` de la instancia SQLStatement a una clase. Al configurar la propiedad `itemClass`, cada fila devuelta por la declaración `SELECT` se crea como una instancia de la clase designada. El motor de ejecución asigna valores de columna de resultados a los valores de la propiedad haciendo coincidir los nombres de columnas en el conjunto de resultados `SELECT` con los nombres de las propiedades en la clase `itemClass`.

Cualquier clase asignada como un valor de propiedad `itemClass` debe tener un constructor que no requiere ningún parámetro. Además, la clase debe tener una sola propiedad para cada columna devuelta por la declaración `SELECT`. Se considera un error si una columna en la lista `SELECT` no tiene un nombre de propiedad coincidente en la clase `itemClass`.

Recuperación de resultados SELECT en partes

Adobe AIR 1.0 y posterior

De forma predeterminada, una ejecución de la declaración `SELECT`recupera las filas del conjunto de resultados de una sola vez. Una vez completada la declaración, generalmente se procesan los datos recuperados de alguna manera, como la creación de objetos o mostrando los datos en pantalla. Si la declaración devuelve un gran número de filas, el procesamiento de todos los datos a la vez puede ser exigente para el equipo, que a su vez hará que la interfaz de usuario no se vuelva a dibujar.

Puede mejorar el rendimiento aparente de la aplicación indicando al motor de ejecución que devuelva un número específico de filas de resultados a la vez. Al proceder de esta manera permite que los datos del resultado inicial se devuelvan con más rapidez. También permite dividir las filas de resultados en conjuntos, para que la interfaz de usuario se actualice después que se procese cada conjunto de filas. Observe que solo es práctico utilizar esta técnica en el modo de ejecución asíncrono.

Para recuperar los resultados `SELECT` en partes, especifique un valor para el primer parámetro (el parámetro `prefetch`) del método `SQLStatement.execute()`. El parámetro `prefetch` indica el número de filas que se deben recuperar la primera vez que se ejecuta la declaración. Cuando llame al método execute()de la instancia de `SQLStatement`, especifique un valor de parámetro `prefetch` y solo se recupera dicho número de filas:

```
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

La declaración distribuye el evento `result`, indicando que está disponible el primer conjunto de filas de resultados. La propiedad data de la instancia de `SQLResult` resultante contiene las filas de datos y la propiedad `complete` indica si hay filas de resultados adicionales para recuperar. Para recuperar filas de resultados adicionales, llame al método `next()` de la instancia SQLStatement. Al igual que el método `execute()`, se usa el primer parámetro del método `next()` para indicar la cantidad de filas que se recuperan la próxima vez que se distribuye el evento result.

```
function selectResult(event:SQLEvent):void
{
    var result:SQLResult = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(SQLEvent.RESULT, selectResult);
        }
    }
}
```

El SQLStatement distribuye un evento `result` cada vez que el método`next()` devuelve un conjunto posterior de filas de resultados. En consecuencia, la misma función del detector se puede usar para continuar procesando los resultados (de las llamadas a `next()`) hasta que se hayan recuperado todas las filas.

Para obtener más información, consulte las descripciones en la referencia de para el método `SQLStatement.execute()` (la descripción del parámetro `prefetch`) y el método `SQLStatement.next()`.

Inserción de datos

Adobe AIR 1.0 y posterior

La adición de datos a una base de datos implica ejecutar una declaración SQL `INSERT`. Una vez que se termina de ejecutar la declaración, puede acceder a la clave principal para la fila recientemente insertada si la base de datos generó la clave.

Ejecución de una declaración INSERT

Adobe AIR 1.0 y posterior

Para añadir datos a una tabla en una base de datos, se crea y ejecuta una instancia de SQLStatement cuyo texto es una declaración SQL `INSERT`.

En el siguiente ejemplo se utiliza una instancia SQLStatement para añadir una fila de datos a la tabla de empelados ya existente. En este ejemplo se demuestra la inserción de datos usando el modo de ejecución asíncrono. Observe que este código supone que existe una instancia SQLConnection denominada `conn` que ya se ha creado y está conectada a una base de datos. Asimismo supone que la tabla “employees” ya ha sido creada.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event:SQLEvent):void
{
    trace("INSERT statement succeeded");
}

function insertError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
```

```
                "VALUES ('Bob', 'Smith', 8000)";
            insertStmt.text = sql;

            // register listeners for the result and failure (status) events
            insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
            insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

            // execute the statement
            insertStmt.execute();
        }

        private function insertResult(event:SQLEvent):void
        {
            trace("INSERT statement succeeded");
        }

        private function insertError(event:SQLErrorEvent):void
        {
            trace("Error message:", event.error.message);
            trace("Details:", event.error.details);
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

En el siguiente ejemplo se añade una fila de datos a la tabla de empelados existente, usando el modo de ejecución sincrónico. Observe que este código supone que existe una instancia SQLConnection denominada `conn` que ya se ha creado y está conectada a una base de datos. Asimismo supone que la tabla “employees” ya ha sido creada.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                try
                {
                    // execute the statement
                    insertStmt.execute();
                    trace("INSERT statement succeeded");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Recuperación de la clave principal generada por la base de datos de una fila insertada

Adobe AIR 1.0 y posterior

A menudo, después de insertar una fila de datos en una tabla, el código necesita conocer una clave principal generada por la base de datos o un valor del identificador de fila para la fila recientemente insertada. Por ejemplo, una vez que inserta una fila en una tabla, puede añadir filas en una tabla relacionada. En ese caso querrá insertar el valor de la clave principal como una clave externa en la tabla relacionada. La clave principal de la fila recientemente insertada se puede recuperar usando el objeto SQLResult asociado a la ejecución de la declaración. Este es el mismo objeto que se utiliza para acceder a los datos del resultado después de ejecutar una declaración `SELECT`. Como con cualquier declaración SQL, cuando se completa la ejecución de una declaración `INSERT` el motor de ejecución crea una instancia SQLResult. Se accede a la instancia de SQLResult llamando al método getResult() del objeto `SQLStatement` si se usa un detector

de evento o si usa el modo de ejecución sincrónico. Como alternativa, si usa el modo de ejecución asíncrono y pasa una instancia de Responder a la llamada `execute()`, la instancia SQLResult se pasa como un argumento a la función de control de resultados. En cualquier caso, la instancia SQLResult tiene una a propiedad, `lastInsertRowID`, que contiene el identificador de fila de la fila más recientemente insertada si la declaración SQL ejecutada es una declaración `INSERT`.

En el siguiente ejemplo se demuestra el acceso a la clave principal de una fila insertada en el modo de ejecución asíncrono:

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
```

En el siguiente ejemplo se demuestra el acceso a la clave principal de una fila insertada en el modo de ejecución sincrónico:

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
catch (error:SQLError)
{
    // respond to the error
}
```

Observe que el identificador de fila puede o no ser el valor de la columna que está designada como la columna de la clave principal en la definición de la tabla, según las siguientes reglas:

- Si se define la tabla con una columna de clave principal cuya afinidad (tipo de datos de columna) es `INTEGER`, la propiedad `lastInsertRowID` contiene el valor que se insertó en esa fila (o el valor generado por el motor de ejecución si es una columna `AUTOINCREMENT`).
- Si se define la tabla con múltiples columnas de clave principal (una clave compuesta) o con una sola columna de clave principal cuya afinidad no es `INTEGER`, en segundo plano la base de datos genera un valor del identificador de fila. Dicho valor generado es el valor de la propiedad `lastInsertRowID`.
- El valor siempre es el identificador de fila de la fila más recientemente insertada. Si una declaración `INSERT` activa un desencadenador y a su vez inserta una fila, la propiedad `lastInsertRowID` contiene el identificador de fila de la última fila insertada por el desencadenador en lugar de la fila creada por la declaración `INSERT`.

Como consecuencia, si desea tener una columna de clave principal explícitamente definida cuya variable está disponible después de un comando `INSERT` a través de la propiedad `SQLResult.lastInsertRowID`, la columna debe estar definida como una columna `INTEGER PRIMARY KEY`. Aunque la tabla no incluye una columna explícita `INTEGER PRIMARY KEY`, resulta igualmente aceptable utilizar el identificador de fila generado por la base de datos como una clave principal para la tabla en el sentido de definir relaciones con tablas relacionadas. El valor de la columna del identificador de fila está disponible en cualquier declaración SQL utilizando uno de los nombres especiales de columna `ROWID`, `_ROWID_` o `OID`. Puede crear una columna de clave externa en una tabla relacionada y usar el valor del identificador de fila como el valor de la columna de la clave externa como lo haría con una columna `INTEGER PRIMARY KEY` explícitamente declarada. En ese sentido, si está usando una clave principal arbitraria en lugar de una clave natural y siempre y cuando no le importe que el motor de ejecución genere el valor de la clave principal, no hay diferencia si utiliza una columna `INTEGER PRIMARY KEY` o el identificador de fila generado por el sistema como la clave principal de la tabla para definir una relación de clave externa entre dos tablas.

Para obtener más información sobre las claves principales y los identificadores de fila generados, consulte "Compatibilidad de SQL en bases de datos locales" en la página 1154.

Cambio o eliminación de datos

Adobe AIR 1.0 y posterior

El proceso para ejecutar otras operaciones de manipulación de datos es idéntico al proceso utilizado para ejecutar una declaración SQL `SELECT` o `INSERT`, tal y como se describe en "Trabajo con declaraciones SQL" en la página 760. Simplemente sustituya una declaración SQL diferente en la misma propiedad text de la instancia `SQLStatement`:

- Para cambiar los datos existentes en una tabla, use una declaración `UPDATE`.
- Para eliminar una o varias filas de datos de una tabla, use una declaración `DELETE`.

Para ver las descripciones de estas declaraciones, consulte "Compatibilidad de SQL en bases de datos locales" en la página 1154.

Trabajo con varias bases de datos

Adobe AIR 1.0 y posterior

Se utiliza el método `SQLConnection.attach()` para abrir una conexión a una base de datos adicional en una instancia SQLConnection que ya tiene una base de datos abierta. Se le asigna un nombre a la base de datos que se asocia utilizando el parámetro name en la llamada del método `attach()`. Cuando se escriben declaraciones para manipular esa base de datos, se puede utilizar el nombre en un prefijo (usando el formulario `database-name.table-name`) para calificar cualquier nombre de tabla en las declaraciones SQL, indicando al motor de ejecución que se puede encontrar la tabla en la determinada base de datos.

Se puede ejecutar una sola declaración SQL que incluye tablas desde múltiples bases de datos que están conectadas a la misma instancia SQLConnection. Si se crea una transacción en la instancia SQLConnection, dicha transacción se aplica para todas las declaraciones SQL que se ejecutan usando la instancia SQLConnection. Esto se cumple independientemente de la base de datos asociada donde se ejecuta la declaración.

Como alternativa, también puede crear múltiples instancias SQLConnection en una aplicación, cada cual conectada a una o múltiples bases de datos. Sin embargo, si se utilizan múltiples conexiones a la misma base de datos tenga en cuenta que la transacción de una base de datos no se comparte con otras instancias SQLConnection. En consecuencia, si se conecta al mismo archivo de base de datos usando múltiples instancias SQLConnection, no puede esperar que se apliquen los cambios de datos de ambas conexiones en la manera esperada. Por ejemplo, si dos declaraciones `UPDATE` o `DELETE` se ejecutan en la misma base de datos a través de diferentes instancias SQLConnection y se produce un error de aplicación después de que se realiza una operación, es posible que los datos de la base de datos queden en un estado intermedio que no es reversible y puede afectar la integridad de la base de datos (y, en consecuencia, a la aplicación).

Gestión de errores de la base de datos

Adobe AIR 1.0 y posterior

En general, la gestión de errores en la base de datos es similar a la gestión de errores del motor de ejecución. Se debe escribir código que está preparado para eventuales errores y solucionar los errores en lugar de dejar que el motor de ejecución lo haga. En general, los posibles errores de la base de datos se pueden dividir en tres categorías: errores de conexión, errores de sintaxis SQL y errores restringidos.

Errores de conexión

Adobe AIR 1.0 y posterior

La mayoría de los errores de la base de datos son errores de conexión y se pueden producir durante cualquier operación. Aunque hay estrategias para prevenir errores de conexión, rara vez hay una manera fácil para recuperarse de un error de conexión si la base de datos es una parte vital de la aplicación.

La mayoría de los errores de conexión están relacionados con la manera en que el motor de ejecución interactúa con el sistema operativo, el sistema de archivos y el archivo de base de datos. Por ejemplo, un error de conexión se produce si el usuario no tiene permiso para crear un archivo de base de datos en una determinada ubicación en el sistema de archivos. Las siguientes estrategias ayudan a prevenir errores de conexión:

Utilizar archivos de base de datos específicos del usuario En lugar de utilizar un solo archivo de base de datos para todos los usuarios que usan la aplicación en un solo equipo, proporcione a cada usuario su propio archivo de base de datos. El archivo se debe ubicar en un directorio asociado con la cuenta del usuario. Por ejemplo, puede estar en el directorio de almacenamiento de la aplicación, en la carpeta de documentos del usuario, en el escritorio del usuario etc.

Considerar diferentes tipos de usuario Pruebe la aplicación con diferentes tipos de cuentas de usuario, en diferentes sistemas operativos. No suponga que el usuario tiene permiso de administrador en el equipo. Asimismo, no suponga que el individuo que instala la aplicación es el usuario que ejecuta la aplicación.

Considerar diferentes ubicaciones de archivo Si permite que un usuario especifique dónde guardar un archivo de base de datos o seleccionar un archivo para abrir, considere las posibles ubicaciones de archivo que puede utilizar el usuario. Además, considere definir los límites dónde los usuarios pueden almacenar (o desde donde pueden abrir) archivos de base de datos. Por ejemplo, podría solo permitir que los usuarios abran los archivos que se encuentran en la ubicación de almacenamiento de sus cuentas de usuario.

Si se produce un error de conexión, muy probablemente ocurra en el primer intento de crear o abrir la base de datos. Esto significa que el usuario no puede realizar ninguna operación relacionada con la base de datos en la aplicación. Para ciertos tipos de errores, como errores de solo lectura o de permiso, una técnica de recuperación posible es copiar el archivo de la base de datos en una ubicación diferente. La aplicación puede copiar el archivo de la base de datos en una ubicación diferente de la que el usuario tiene permiso para crear y escribir en los archivos y, en cambio, utilizar esa ubicación.

Errores de sintaxis

Adobe AIR 1.0 y posterior

Un error de sintaxis se produce cuando una declaración SQL se forma incorrectamente y la aplicación intenta ejecutar la declaración. Dado que las declaraciones SQL de la base de datos local se crean como cadenas, no es posible verificar la sintaxis SQL durante el tiempo de compilación. Se deben ejecutar todas las declaraciones SQL para verificar la sintaxis. Utilice las siguientes estrategias para prevenir errores de sintaxis SQL:

Probar detalladamente todas las declaraciones SQL Si es posible, mientras desarrolla la aplicación pruebe las declaraciones SQL de forma separada antes de codificarlas como texto de la declaración en el código de aplicación. Además, use un método de prueba de código como probar las unidades para crear un conjunto de pruebas que ejecutan todas las opciones y variaciones posibles en el código.

Utilizar parámetros de declaración y evitar la concatenación SQL (generada dinámicamente). El uso de parámetros y evitar las declaraciones SQL creadas dinámicamente, significa que el mismo texto de la declaración SQL se usa cada vez que se ejecuta una declaración. En consecuencia, es más fácil probar las declaraciones y limitar posibles variaciones. Si debe generar dinámicamente una declaración SQL, reduzca al mínimo las partes dinámicas de la declaración. Asimismo, valide cuidadosamente cualquier entrada del usuario para asegurarse de que no causará errores de sintaxis.

Para recuperarse de un error de sintaxis, una aplicación necesitaría una lógica compleja para poder examinar una declaración SQL y corregir la sintaxis. Si se siguen las pautas anteriores para prevenir errores de sintaxis, el código puede identificar cualquier origen de tiempo de ejecución potencial de errores de sintaxis SQL (como entradas del usuario utilizadas en una declaración). Para recuperarse de un error de sintaxis, debe asesorar al usuario. Indique lo que se debe corregir para que la declaración se pueda ejecutar correctamente.

Errores de restricción

Adobe AIR 1.0 y posterior

Los errores de restricción ocurren cuando una declaración `INSERT` o `UPDATE` intenta añadir datos a una columna. El error se produce si los nuevos datos infringen una de las restricciones definidas para la tabla o columna. A continuación se describe el conjunto de posibles restricciones:

Restricción exclusiva Indica que en todas las filas de una tabla, no pueden haber valores repetidos en una columna. Como alternativa, cuando se combinan múltiples columnas en una restricción exclusiva, la combinación de valores en dichas columnas no se debe repetir. Es decir, con respecto a la o las columnas exclusivas especificadas, cada fila debe ser diferente.

Restricción de clave principal En cuanto a los datos que permite y no permite una restricción, una restricción de clave principal es idéntica a una restricción exclusiva.

Restricción de valor null Especifica que una sola columna no puede almacenar un valor `NULL` y, en consecuencia, en cada fila dicha columna debe tener un valor.

Restricción de verificación Permite especificar una restricción arbitraria en una o más tablas. Una restricción de verificación común es una regla que define que el valor de una columna debe estar dentro de ciertos límites (por ejemplo, que el valor numérico de una columna debe ser mayor que 0). Otro tipo común de restricción de verificación especifica las relaciones entre valores de columna (por ejemplo, que el valor de una columna debe ser diferente al valor de otra columna en la misma fila).

Restricción de tipos de datos (afinidad de columna) El motor de ejecución impone el tipo de datos de los valores de las columnas y se produce un error si se intenta almacenar un valor del tipo incorrecto en una columna. Sin embargo, en muchas condiciones los valores se convierten para que coincidan con el tipo de datos declarados de la columna. Consulte “Trabajo con tipos de datos de la base de datos” en la página 784 para más información.

El motor de ejecución no impone restricciones en los valores de claves externas. Es decir, los valores de claves externas no tienen que coincidir con el valor de una clave principal existente.

Además de los tipos de restricción predefinidos, el motor SQL de tiempo de ejecución admite el uso de desencadenadores. Un desencadenador es similar a un controlador de eventos. Es un conjunto de instrucciones predefinidas que se llevan a cado cuando se produce una determinada acción. Por ejemplo, se puede definir un desencadenador para que se ejecute cuando se introducen o eliminan datos de una tabla en particular. Un uso posible de un desencadenador es examinar los cambios de datos y generar un error si no se cumplen determinadas condiciones. En consecuencia, un desencadenador puede tener el mismo fin que una restricción y las estrategias para prevenir y recuperarse de errores de restricción también se aplican a los errores generados por el desencadenador. Sin embargo, el ID de error para errores generados por el desencadenador es diferente del ID de error para errores de restricción.

El conjunto de restricciones que se aplica a una tabla en particular se determina mientras que se diseña una aplicación. La creación consciente de restricciones facilita el diseño de la aplicación para prevenir y recuperarse de errores de restricción. Sin embargo, los errores de restricción son difíciles de predecir y prevenir sistemáticamente. La predicción es difícil porque los errores de restricción no aparecen hasta que se añaden datos de aplicación. Los errores de restricción se generan con los datos que se añaden a una base de datos después de que se crea. Estos errores con frecuencia son el resultado de la relación entre los nuevos datos que ya existen en la base de datos. Las siguientes estrategias le pueden ayudar a evitar muchos errores de restricción:

Planificar cuidadosamente la estructura y las restricciones de la base de datos El objetivo de las restricciones es imponer las reglas de aplicación y ayudar a proteger la integridad de los datos de la base de datos. Cuando está planificando la aplicación, considere la manera de estructurar la base de datos para que sea compatible con la aplicación. Como parte de ese proceso, identifique reglas para los datos, como por ejemplo si se requieren ciertos valores, si un valor tiene un valor predeterminado, si se permiten valores repetidos etc. Esas reglas lo guían para definir las restricciones de la base de datos.

Especificar explícitamente los nombres de las columnas Se puede escribir una declaración `INSERT` sin especificar explícitamente las columnas donde se deben insertar los valores, pero hacerlo es correr un riesgo innecesario. Al nombrar explícitamente las columnas donde se insertan los valores, se puede permitir el uso de valores generados automáticamente, columnas con valores predeterminados y columnas que permiten valores `NULL`. Además, al hacerlo garantiza que todas las columnas `NOT NULL` tienen insertadas un valor explicito.

Utilizar valores predeterminados Cuando especifica una restricción `NOT NULL`para una columna, si es posible, especifique un valor predeterminado en la definición de la columna. El código de la aplicación también puede proporcionar valores predeterminados. Por ejemplo, el código puede verificar si una variable String es `null` y asignarle un valor antes de usarla para definir un valor al parámetro de declaración.

Validar los datos introducidos por el usuario Verifique con antelación los datos introducidos por el usuario para asegurarse de cumplen con los límites especificados por las restricciones, especialmente en el caso de las restricciones `NOT NULL` y `CHECK`. Naturalmente, una restricción `UNIQUE` es más difícil de verificar dado que al hacerlo se requiere la ejecución de una consulta `SELECT` para determinar si los datos son exclusivos.

Utilizar desencadenadores Puede escribir un desencadenador que valida (y posiblemente remplaza) los datos insertados o toma otras acciones para corregir los datos no válidos. Esta validación y corrección puede prevenir la generación de un error de restricción.

En muchos sentidos los errores de restricción son más difíciles de prevenir que otros tipos de errores. Afortunadamente, existen estrategias para recuperarse de errores de restricción de manera que no desestabilice ni desactive la aplicación:

Utilizar algoritmos de conflicto Cuando define una restricción en una columna y cuando crea una declaración `INSERT` o `UPDATE`, tiene la opción de especificar un algoritmo de conflicto. Un algoritmo de conflicto define la acción que implementa la base de datos cuando se genera una infracción de restricción. Existen varias acciones posibles que puede

implementar el motor de base de datos. El motor de base de datos puede terminar una sola declaración o toda una transacción. Puede omitir el error. Hasta puede quitar datos antiguos y reemplazarlos con los datos que el código intenta almacenar.

Para obtener más información, consulte la sección "ON CONFLICT (algoritmos de conflicto)" en "Compatibilidad de SQL en bases de datos locales" en la página 1154.

Proporcionar comentarios constructivos se puede identificar con antelación el conjunto de restricciones que pueden afectar un determinado SQL. En consecuencia, puede anticipar los errores de restricción que podría generar una declaración. Sabiendo esto, puede hacer que la lógica de la aplicación responda a un error de restricción. Por ejemplo, supongamos que una aplicación incluye un formulario de entrada de datos para introducir nuevos productos. Si la columna del nombre del producto en la base de datos se define con una restricción `UNIQUE`, la acción de insertar una nueva fila de producto en la base de datos podría generar un error de restricción. En consecuencia, se diseña la aplicación para que anticipe un error de restricción. Cuando se produce el error, la aplicación alerta al usuario indicando que el nombre de producto especificado ya está en uso y le solicita al usuario que elija uno diferente. Otra respuesta posible es permitir que el usuario vea información sobre el otro producto con el mismo nombre.

Trabajo con tipos de datos de la base de datos

Adobe AIR 1.0 y posterior

Cuando se crea una tabla en una base de datos, la declaración SQL utilizada para crear la tabla define la afinidad o tipo de datos para cada columna en la tabla. Aunque se pueden omitir las declaraciones de afinidad, se recomienda declarar explícitamente la afinidad de columna en las declaraciones SQL `CREATE TABLE`.

Como regla general, cualquier objeto que se almacena en una base de datos usando una declaración `INSERT` se devuelve como una instancia del mismo tipo de datos cuando se ejecuta una declaración `SELECT`. Sin embargo, el tipo de datos del valor recuperado puede ser diferente según la afinidad de la columna de la base de datos donde se almacena el valor. Cuando se almacena un valor en una columna, si el tipo de datos no coincide con la afinidad de la columna, la base de datos intenta convertir el valor para que coincida con la afinidad de la columna. Por ejemplo, si una columna de la base de datos se declara con la afinidad `NUMERIC`, la base de datos intenta convertir los datos insertados en una clase de almacenamiento numérico (`INTEGER` o `REAL`) antes de guardar los datos. La base de datos emite un error si no se pueden convertir los datos. Según esta regla, si se inserta la cadena "12345" en una columna `NUMERIC`, la base de datos automáticamente convierte al valor entero 12345 antes de guardarla en la base de datos. Cuando se recupera con una declaración `SELECT`, se devuelve el valor como una instancia de un tipo de datos numérico (como Number) en lugar de una instancia String.

La mejor manera de evitar la conversión no deseada de tipos de datos es seguir estas dos reglas. Primero, defina cada columna con la afinidad que coincide el tipo de datos que se desea almacenar. A continuación, solo inserte los valores cuyos tipos de datos coinciden con la afinidad definida. El seguimiento de estas reglas tiene dos ventajas. Cuando inserta los datos no se convierten inesperadamente (posiblemente pierde su significado original como resultado) Además, cuando recupera los datos se devuelven en el tipo de datos original.

Para obtener más información sobre los tipos disponibles de afinidad de columnas y sober cómo utilizar los tipos de datos en declaraciones SQL, consulte "Compatibilidad de tipos de datos" en la página 1177.

Uso de operaciones sincrónicas y asíncronas de base de datos

Adobe AIR 1.0 y posterior

En secciones anteriores se han descrito las operaciones comunes de la base de datos como por ejemplo recuperar, insertar, actualizar y eliminar datos así como la creación de un archivo de base de datos y tablas y otros objetos en una base de datos. Los ejemplos han demostrado cómo realizar las operaciones tanto de forma asíncrona como sincrónicamente.

Como recordatorio, en el modo de ejecución asíncrono se le indica al motor de la base de datos realizar una operación. El motor de la base de datos trabaja en segundo plano mientras que la aplicación continúa ejecutándose. Cuando termina la operación el motor de la base de datos distribuye un evento para alertarlo sobre el hecho. La ventaja principal de una ejecución asíncrona es que el motor de ejecución realiza las operaciones de la base de datos en segundo plano mientras el código de la aplicación principal continúa ejecutándose. Esto es especialmente valioso cuando la operación tarda considerablemente para ejecutarse.

Por otro lado, en el modo de ejecución sincrónico, las operaciones no se ejecutan en segundo plano. Se le indica al motor de la base de datos que realice una operación. Este código hace una pausa en ese punto mientras que el motor de base de datos hace su trabajo. Cuando se completa la operación, la ejecución continúa con la siguiente línea de código.

Una sola conexión de base de datos no puede ejecutar algunas operaciones o declaraciones sincrónicamente y otras de forma asíncrona. Se especifica si una instancia de SQLConnection funciona en el modo sincrónico o asíncrono cuando se abre la conexión a la base de datos. Si llama a `SQLConnection.open()` la conexión funciona en el modo de ejecución sincrónico y si llama a `SQLConnection.openAsync()` la conexión funciona en el modo de ejecución asíncrono. Una vez que una instancia SQLConnection se conecta a una base de datos usando `open()` o `openAsync()`, se fija al modo de ejecución asíncrono o sincrónico.

Uso de operaciones sincrónicas de base de datos

Adobe AIR 1.0 y posterior

Existe una mínima diferencia en el código real que se usa para ejecutar y responder a las operaciones cuando usa la ejecución sincrónica, comparado con el código para el modo de ejecución asíncrono. Las diferencias principales entre los dos métodos se observan en dos áreas. La primera es la ejecución de una operación que depende de otra operación (como filas de resultados `SELECT` o la clave principal de la fila añadida por una declaración `INSERT`). La segunda área de diferencia es en la gestión de errores.

Cómo escribir código para las operaciones sincrónicas

Adobe AIR 1.0 y posterior

La diferencia principal entre la ejecución sincrónica y la ejecución asíncrona es que en el modo sincrónico se escribe el código como una sola serie de pasos. Por el contrario, en el código asíncrono se registran detectores de eventos y con frecuencia se dividen operaciones entre los métodos de detectores. Cuando una base de datos se conecta en el modo de ejecución sincrónico, se puede ejecutar una serie de operaciones de base de datos sucesivamente dentro de un solo bloque de código. En el siguiente ejemplo se demuestra esta técnica:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

Como puede ver, se llama a los mismos métodos para realizar las operaciones de base de datos independientemente si utiliza la ejecución sincrónica o asíncrona. Las diferencias principales entre los dos métodos son ejecutar una operación que depende de otra operación y la gestión de errores.

Ejecución de una operación que depende de otra operación

Adobe AIR 1.0 y posterior

Cuando utiliza el modo de ejecución sincrónico, no necesita escribir código que detecta un evento para determinar cuando se completa una operación. En cambio, puede suponer que si una operación en una línea de código se completa correctamente, la ejecución continúa con la siguiente línea de código. En consecuencia, para realizar una operación que depende del éxito de otra operación, simplemente escriba el código dependiente inmediatamente después de la operación de la que depende. Por ejemplo, para codificar una aplicación para que inicie una transacción, ejecute una declaración `INSERT`, recupere la clave principal de la fila insertada, inserte esa clave principal en otra fila de una tabla diferente y finalmente confirme la transacción, se puede escribir todo el código como una serie de declaraciones. En el siguiente ejemplo se demuestran estas operaciones:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

Gestión de errores con la ejecución sincrónica

Adobe AIR 1.0 y posterior

En el modo de ejecución sincrónico, no se detecta un evento de error para determinar que ha fallado una operación. En cambio, se rodea el código que podría desencadenar errores en un conjunto de bloques de código `try..catch..finally`. Se agrupa el código de emisión de error en el bloque `try`. Se escriben las acciones para realizar en respuesta a cada tipo de error en bloques `catch` por separado. Coloque el código que desea que siempre se ejecute independientemente del éxito o error (por ejemplo, cerrar una conexión de base de datos que ya no se necesita) en un bloque `finally`. En el siguiente ejemplo se demuestra el uso de los bloques `try..catch..finally` para la gestión de errores. Se basa en el ejemplo anterior añadiendo el código de gestión de error:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer:SQLStatement = new SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId:Number = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber:SQLStatement = new SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error:SQLError)
{
    // rollback the transaction
    conn.rollback();
}
```

Aspectos básicos del modelo de ejecución asíncrono

Adobe AIR 1.0 y posterior

Una preocupación común acerca del uso del modo de ejecución asíncrono es la suposición de que no se puede comenzar a ejecutar una instancia de SQLStatement si otra instancia de SQLStatement se está ejecutando con la misma conexión de base de datos. De hecho, esta suposición no es correcta. Mientras que se ejecuta una instancia SQLStatement no se puede cambiar la propiedad `text` de la declaración. Sin embargo, si se utiliza una instancia SQLStatement por separado para cada declaración SQL diferente que se desea ejecutar, se puede llamar al método `execute()` de una instancia SQLStatement mientras otra instancia SQLStatement aún se está ejecutando, sin generar un error.

Internamente, cuando se ejecutan operaciones de base de datos usando el modo de ejecución asíncrono, cada conexión de base de datos (cada instancia de SQLConnection) tiene su propia cola o lista de operaciones que debe llevar a cabo. El motor de ejecución ejecuta cada operación en secuencia, en el orden en que se añaden a la cola. Cuando se crea una instancia SQLStatement y se llama al método `execute()`, esa operación de ejecución de la declaración se añade a la cola para la conexión. Si no se está ejecutando ninguna operación en esa instancia SQLConnection, la declaración comienza la ejecución en segundo plano. Supongamos que dentro del mismo bloque de código crea otra instancia SQLStatement y también llama al método `execute(()`. Esa segunda operación de ejecución de la declaración se añade a la cola detrás de la primera declaración. En cuanto termina la ejecución de la primera declaración, el motor de ejecución se traslada a la siguiente operación en la cola. El procesamiento de las operaciones posteriores en la cola ocurre en segundo plano, aun cuando el evento `result` para la primera operación se está distribuyendo en el código la aplicación principal. En el siguiente código se demuestra esta técnica:

```
// Using asynchronous execution mode
var stmt1:SQLStatement = new SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2:SQLStatement = new SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

Hay un efecto colateral importante si la base de datos ejecuta automáticamente declaraciones posteriores en la cola. Si una declaración depende del resultado de otra operación, no se puede añadir la declaración a la cola (es decir, no se puede llamar al método `execute()`) hasta que la primera operación se complete. Esto se debe a que una vez que se ha llamado al método `execute()` de la segunda declaración no se pueden cambiar las propiedades `text` o `parameters` de la instrucción. En ese caso se debe esperar a que el evento indique que la primera operación está completada antes de comenzar con la siguiente operación. Por ejemplo, si desea ejecutar una declaración en el contexto de una transacción, la ejecución de la declaración depende de la operación de abrir la transacción. Después de llamar al método `SQLConnection.begin()` para abrir la transacción, necesita esperar a que la instancia SQLConnection distribuya el evento `begin`. Solo entonces puede llamar al método `execute()` de la instancia SQLStatement. En este ejemplo la manera más fácil de organizar la aplicación para asegurar que las operaciones se ejecutan correctamente es crear un método que está registrado como un detector para el evento `begin`. El código para llamar al método `SQLStatement.execute()` se coloca dentro del método del detector.

Uso del cifrado con bases de datos SQL

Adobe AIR 1.5 y posterior

Todas las aplicaciones de Adobe AIR comparten el mismo motor de base de datos local. Por lo tanto, cualquier aplicación de AIR puede conectarse, leer y escribir en un archivo de base de datos no cifrada. Al comenzar con Adobe AIR 1.5, AIR incluye la capacidad de crear y conectarse con archivos de bases de datos cifradas. Cuando se utiliza una base de datos cifrada, con el fin de conectarse a la base de datos una aplicación debe proporcionar la clave de cifrado correcta. Si se indica una clave incorrecta (o no se indica ninguna), la aplicación no podrá conectarse a la base de datos. Por lo tanto, la aplicación no podrá leer información de la base de datos ni escribir o modificar datos.

Para usar una base de datos cifrada, debe crearla como cifrada. Con una base de datos cifrada existente, podrá abrir una conexión a la base de datos. También es posible cambiar la clave de una base de datos cifrada. Además de crear y conectarse a bases de datos cifradas, las técnicas para trabajar con una base de datos cifrada son las mismas que para el trabajo con una base de datos no cifrada. En concreto, la ejecución de declaraciones SQL es la misma independientemente de si una base de datos se encuentra o no cifrada.

Usos de una base de datos cifrada

Adobe AIR 1.5 y posterior

El cifrado resulta útil cuando se desea restringir el acceso a la información almacenada en una base de datos. La funcionalidad del cifrado de la base de datos de Adobe AIR se puede emplear para varios usos. A continuación se incluyen algunos ejemplos de casos en los que podría resultar adecuado el uso de una base de datos cifrada:

- Una memoria caché de solo lectura de datos de aplicación privada descargados desde un servidor.
- Un almacén de aplicación local para datos privados que esté sincronizado con un servidor (los datos se envían al servidor y se descargan desde el mismo).
- Archivos cifrados utilizados como formato de archivo para documentos creados y editados por la aplicación. Los archivos pueden ser privados para un usuario, o bien, pueden estar diseñados para compartirse entre todos los usuarios de la aplicación.
- Cualquier otro uso de un almacén de datos local como, por ejemplo, los que se describen en "Usos de las bases de datos SQL locales" en la página 745, donde los datos deben ser privados para los usuarios que disponen de acceso al equipo o a los archivos de la base de datos.

Conocer el motivo por el que se desea utilizar una base de datos cifrada ayuda a decidir la forma en que se plantea la arquitectura de la aplicación. En concreto, puede afectar al modo en que la aplicación crea, obtiene o almacena la clave de cifrado para la base de datos. Para obtener más información sobre estos puntos, consulte "Consideraciones para el uso del cifrado con una base de datos" en la página 794.

Además de una base de datos cifrada, un mecanismo alternativo para proteger la privacidad de datos importantes es el almacén local cifrado. Con el almacén local cifrado, se almacena un solo valor ByteArray utilizando una clave String. Únicamente la aplicación de AIR que almacenó el valor puede acceder al mismo y solo en el equipo en el que está almacenado. Con el almacén local cifrado, no es necesario crear una clave de cifrado propia. Por estos motivos, el almacén local cifrado resulta más adecuado para almacenar fácilmente un solo valor o conjunto de valores que se pueden codificar con facilidad en un objeto ByteArray. Una base de datos cifrada es más apropiada para conjuntos de datos más grandes donde el almacenamiento de datos estructurados y las consultas resultan más recomendables. Para obtener más información sobre el uso del almacén local cifrado, consulte "Almacenamiento local cifrado" en la página 740.

Creación de una base de datos cifrada

Adobe AIR 1.5 y posterior

Para poder utilizar una base de datos cifrada, el archivo de base de datos debe estar cifrado cuando se cree. Una vez que una base de datos se crea como no cifrada, no puede cifrarse posteriormente. Asimismo, una base de datos cifrada no puede encontrarse sin cifrar más adelante. Si es necesario, puede cambiar la clave de cifrado de una base de datos cifrada. Para obtener más información, consulte "Cambio de la clave de cifrado de una base de datos" en la página 793. Si dispone de una base de datos existente que no está cifrada y desea usar el cifrado de base de datos, puede crear una nueva base de datos cifrada y copiar la estructura de tabla existente y la información en la nueva base de datos.

La creación de una base de datos cifrada es un proceso casi idéntico a la creación de una base de datos no cifrada, tal y como se describe en "Creación de una base de datos" en la página 749. En primer lugar se crea una instancia de SQLConnection que representa la conexión con la base de datos. La base de datos se crea llamando al método `open()` del objeto SQLConnection o al método `openAsync()`, especificando para la ubicación de la base de datos un archivo que aún no exista. La única diferencia al crear una base de datos cifrada es que se proporciona un valor para el parámetro `encryptionKey` (el quinto parámetro del método `open()` y el sexto parámetro del método `openAsync()`).

Un valor del parámetro `encryptionKey` válido es un objeto ByteArray que contiene exactamente 16 bytes.

Los siguientes ejemplos muestran la creación de una base de datos cifrada. Para que resulte más fácil, en estos ejemplos la clave de cifrado está programada en el código de la aplicación. Sin embargo, el uso de esta técnica no es recomendable porque no es segura.

```
var conn:SQLConnection = new SQLConnection();

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, SQLMode.CREATE, false, 1024, encryptionKey);
```

Para obtener un ejemplo que muestre un modo recomendado de generar una clave de cifrado, consulte "Ejemplo: Generación y uso de claves de cifrado" en la página 795.

Conexión con una base de datos cifrada

Adobe AIR 1.5 y posterior

Al igual en sucede en la creación de una base de datos cifrada, el procedimiento para abrir una conexión a una base de datos cifrada es como la conexión a una base de datos no cifrada. Dicho procedimiento se describe con más detalle en "Conexión a una base de datos" en la página 757. El método `open()` se utiliza para abrir una conexión en modo de ejecución sincrónico o el método o el método `openAsync()` para abrir una conexión en modo de ejecución asíncrono. La única diferencia radica en que para abrir una base de datos cifrada, se especifica el valor correcto para el parámetro `encryptionKey` (el quinto parámetro del método `open()` y el sexto parámetro del método `openAsync()`).

Si la clave de cifrado que se proporciona no es correcta, se generará un error. Para el método `open()`, se emite una excepción SQLError. Para el método `openAsync()`, el objeto SQLConnection distribuye SQLErrorEvent, cuya propiedad `error` contiene un objeto SQLError. En cualquier caso, el objeto SQLError generado mediante la excepción cuenta con el valor de propiedad `errorID` 3138. Este ID de error se corresponde con el mensaje de error "File opened is not a database file" (El archivo abierto no es un archivo de base de datos).

En el siguiente ejemplo se muestra la apertura de una base de datos cifrada en modo de ejecución asíncrono. Para que resulte más fácil, en este ejemplo la clave de cifrado está programada en el código de la aplicación. Sin embargo, el uso de esta técnica no es recomendable porque no es segura.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    if (event.error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", event.error.message);
        trace("Details:", event.error.details);
    }
}
```

En el siguiente ejemplo se muestra la apertura de una base de datos cifrada en modo de ejecución sincrónico. Para que resulte más fácil, en este ejemplo la clave de cifrado está programada en el código de la aplicación. Sin embargo, el uso de esta técnica no es recomendable porque no es segura.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, SQLMode.UPDATE, false, 1024, encryptionKey);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    if (error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", error.message);
        trace("Details:", error.details);
    }
}
```

Para obtener un ejemplo que muestre un modo recomendado de generar una clave de cifrado, consulte "Ejemplo: Generación y uso de claves de cifrado" en la página 795.

Cambio de la clave de cifrado de una base de datos

Adobe AIR 1.5 y posterior

Cuando se cifra una base de datos, es posible cambiar su clave de cifrado con posterioridad. Para cambiar una clave de cifrado de la base de datos, en primer lugar abra una conexión con la base de datos, creando una instancia de SQLConnection llamando a su método `open()` o `openAsync()`. Una vez que la base de datos esté conectada, llame al método `reencrypt()`, transmitiendo la nueva clave de cifrado como argumento.

Al igual que sucede en la mayoría de las operaciones de bases de datos, el comportamiento del método `reencrypt()` varía dependiendo de si la conexión de base de datos utiliza el modo de ejecución sincrónico o asíncrono. Si se utiliza el método `open()` para realizar la conexión a la base de datos, la operación `reencrypt()` se ejecuta de forma sincrónica. Cuando se completa la operación, la ejecución continúa con la siguiente línea de código:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

Por otra parte, si la conexión de base de datos se abre usando el método `openAsync()`, la operación `reencrypt()` es asíncrona. Al llamar a `reencrypt()`, comienza el proceso de nuevo cifrado. Cuando la operación finaliza, el objeto SQLConnection distribuye un evento `reencrypt`. El detector de eventos se emplea para determinar cuándo finaliza el nuevo cifrado:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event:SQLEvent):void
{
    // save the fact that the key changed
}
```

La operación `reencrypt()` se ejecuta en su propia transacción. Si la operación se interrumpe o falla (por ejemplo, si la aplicación se cierra antes de que finalice la operación), la transacción se deshace. En este caso, la clave de cifrado original es aún la clave de cifrado para la base de datos.

El método `reencrypt()` no se puede utilizar para eliminar el cifrado de una base de datos. La transmisión de un valor `null` o de una clave de cifrado que no es un objeto ByteArray de 16 bytes al método `reencrypt()`, genera un error.

Consideraciones para el uso del cifrado con una base de datos

Adobe AIR 1.5 y posterior

En la sección "Usos de una base de datos cifrada" en la página 790 se presentan varios casos en los que sería adecuado el uso de una base de datos cifrada. Resulta evidente que los escenarios de uso de diferentes aplicaciones (incluyendo estos y otros escenarios) cuentan con requisitos de privacidad distintos. La forma en que se plantea la arquitectura del uso del cifrado en cada aplicación representa una parte importante en el control del nivel de privacidad de la información de una base de datos. Por ejemplo, si está utilizando una base de datos cifrada parar proteger la privacidad de datos personales, incluso de otros usuarios del mismo equipo, la base de datos de cada usuario necesita su propia clave de cifrado. Para obtener la máxima seguridad, su aplicación puede generar una clave a partir de una contraseña introducida por el usuario. Si la clave de cifrado se basa en una contraseña, se garantiza que aunque otra persona pueda suplantar la cuenta del usuario en el equipo, no sea posible acceder a los datos. En el otro extremo del espectro de la privacidad, supongamos que desea que cualquier usuario de su aplicación, pero no de otras aplicaciones, pueda leer un archivo de la base de datos. En este caso, todas las copias instaladas de la aplicación necesitan acceso a una clave de cifrado compartida.

La aplicación se puede diseñar, y en concreto la técnica utilizada para generar la clave de cifrado, según el nivel de privacidad que se desee para los datos de la aplicación. En la siguiente lista se incluyen sugerencias de diseño para distintos niveles de privacidad de datos:

- Para que cualquier usuario que tenga acceso a la aplicación pueda acceder a una base de datos en cualquier equipo, utilice una sola clave que esté disponible en todas las instancias de la aplicación. Por ejemplo, la primera vez que se ejecute una aplicación, puede descargar la clave de cifrado compartida de un servidor mediante un protocolo seguro como, por ejemplo, SSL. Después puede guardar la clave en el almacén local cifrado para su uso posterior. Como alternativa, cifre los datos por usuario en el equipo y sincronícelos con un almacén de datos remoto como, por ejemplo, un servidor para hacer que la información sea portátil.
- Para que un solo usuario pueda acceder a una base de datos en cualquier equipo, genere la clave de cifrado a partir de un secreto de usuario (por ejemplo, una contraseña). En especial, no utilice ningún valor que esté asociado a un equipo concreto (por ejemplo, un valor almacenado en el almacén local cifrado) para generar la clave. Como alternativa, cifre los datos por usuario en el equipo y sincronícelos con un almacén de datos remoto como, por ejemplo, un servidor para hacer que la información sea portátil.

- Para que únicamente un usuario concreto pueda acceder a una base de datos en un solo equipo, genere la clave a partir de una contraseña y un valor salt generado. Para ver un ejemplo de esta técnica, consulte "Ejemplo: Generación y uso de claves de cifrado" en la página 795.

A continuación se incluyen consideraciones sobre seguridad adicionales que son importantes tener en cuenta a la hora de diseñar una aplicación que utilice una base de datos cifrada:

- Un sistema solo es seguro en su vínculo más débil. Si usa una contraseña introducida por el usuario para generar una clave de cifrado, tenga en cuenta la aplicación de restricciones de complejidad y longitud mínima en las contraseñas. Una contraseña corta que solo utilice caracteres básicos se puede adivinar rápidamente.
- El código fuente de una aplicación de AIR se almacena en un equipo de usuario en texto sin formato (para contenido HTML), o bien, en un formato binario que se puede descompilar (para contenido SWF). Debido a que es posible acceder al código fuente, se deben recordar dos puntos:
  - No programe nunca una clave de cifrado en el código fuente.
  - Se debe tener siempre en cuenta que la técnica empleada para generar una clave de cifrado (por ejemplo, generador de caracteres aleatorios o un algoritmo hash concreto) puede ser desvelada fácilmente por un atacante.
- El cifrado de base de datos de AIR utiliza el algoritmo AES (Advanced Encryption Standard, estándar de cifrado avanzado) con modo de contador con CBC-MAC (CCM). Este sistema de cifrado requiere que una clave introducida por el usuario se combine con un valor salt para ser segura. Para ver un ejemplo, consulte "Ejemplo: Generación y uso de claves de cifrado" en la página 795.
- Si opta por cifrar una base de datos, se cifrarán todos los archivos de disco utilizados por el motor de la base de datos junto con dicha base de datos. Sin embargo, el motor de base de datos retiene algunos datos temporalmente en una caché de memoria interna para mejorar el rendimiento de lectura y escritura en las transacciones. Los datos residentes en memoria no están cifrados. Si un atacante puede acceder a la memoria utilizada por una aplicación de AIR (por ejemplo, mediante un depurador), la información de una base de datos que esté abierta y sin cifrar actualmente podría quedar expuesta.

Ejemplo: Generación y uso de claves de cifrado

Adobe AIR 1.5 y posterior

Esta aplicación de ejemplo muestra una técnica para generar una clave de cifrado. Esta aplicación está diseñada para proporcionar el nivel más elevado de privacidad y seguridad para los datos de los usuarios. Un aspecto importante a la hora de asegurar los datos privados radica en solicitar al usuario que introduzca una contraseña cada vez que la aplicación se conecta a la base de datos. Por lo tanto, tal y como se muestra en este ejemplo, una aplicación que requiere este nivel de privacidad nunca debe almacenar directamente la clave de cifrado de la base de datos.

La aplicación consta de dos partes: una clase ActionScript que genera una clave de cifrado (clase EncryptionKeyGenerator) y una interfaz de usuario básica que muestra cómo utilizar esa clase. Para obtener el código fuente completo, consulte "Código de ejemplo completo para generar y utilizar una clave de cifrado" en la página 797.

Uso de la clase EncryptionKeyGenerator para obtener una clave de cifrado segura

Adobe AIR 1.5 y posterior

No es necesario comprender los detalles del funcionamiento de la clase EncryptionKeyGenerator para utilizarla en la aplicación. Si está interesado en los detalles del modo en que la clase genera una clave de cifrado para una base de datos, consulte "Aspectos básicos de la clase EncryptionKeyGenerator" en la página 802.

Siga los siguientes pasos para usar la clase EncryptionKeyGenerator en su aplicación:

1 Descargue la clase EncryptionKeyGenerator como código fuente o archivo SWC compilado. La clase EncryptionKeyGenerator se incluye en el proyecto (as) de biblioteca principal de ActionScript 3.0 de código abierto (3corelib). Puede descargar el paquete as3corelib, incluyendo el código fuente y la documentación. También puede descargar el archivo SWC o los archivos de código fuente desde la página del proyecto.

2 Sitúe el código fuente de la clase (o el archivo SWC as3corelib) en una ubicación donde el código fuente de la aplicación pueda encontrarla.

3 En el código fuente de la aplicación, añada una declaración `import` para la clase EncryptionKeyGenerator.

```
import com.adobe.air.crypto.EncryptionKeyGenerator;
```

4 Antes del punto donde el código crea la base de datos o abre una conexión con la misma, añada código para crear una instancia de EncryptionKeyGenerator, llamando al constructor `EncryptionKeyGenerator()`.

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
```

5 Obtenga una contraseña del usuario:

```
var password:String = passwordInput.text;

if (!keyGenerator.validateStrongPassword(password))
{
    // display an error message
    return;
}
```

La instancia de EncryptionKeyGenerator utiliza esta contraseña como base para la clave de cifrado (se muestra en el siguiente paso). La instancia de EncryptionKeyGenerator comprueba la contraseña con determinados requisitos de validación de contraseña segura. Si la validación falla, se produce un error. Tal y como muestra el código de ejemplo, puede comprobar la contraseña con anterioridad llamando al método `validateStrongPassword()` del objeto EncryptionKeyGenerator. De este modo, puede determinar si la contraseña cumple con los requisitos mínimos de una contraseña segura y evitar un error.

6 Genere la clave de cifrado a partir de la contraseña:

```
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

El método `getEncryptionKey()` genera y devuelve la clave de cifrado(ByteArray de 16 bytes). Posteriormente la clave de cifrado se puede utilizar para crear una nueva base de datos cifrada o abrir una existente.

El método `getEncryptionKey()` cuenta con un parámetro necesario, que es la contraseña obtenida en el paso 5.

Nota: *para mantener el nivel más alto de seguridad y privacidad de los datos, una aplicación debe solicitar al usuario que indique una contraseña cada vez que la aplicación se conecte a la base de datos. No almacene la contraseña de usuario ni la clave de cifrado de la base de datos directamente. De este modo se expone a riesgos de seguridad. Tal y como se muestra en este ejemplo, una aplicación debe emplear la misma técnica para derivar la clave de cifrado de la contraseña, tanto al crear la base de datos cifrada como al conectarse a ella más adelante.*

El método `getEncryptionKey()` también acepta un segundo parámetro(opcional); el parámetro `overrideSaltELSKey`. EncryptionKeyGenerator crea un valor aleatorio (denominado valor *salt*) que se usa como parte de la clave de cifrado. Para poder volver a crear la clave de cifrado, el valor salt se almacena en el almacén local cifrado (ELS) de la aplicación de AIR. De forma predeterminada, la clase EncryptionKeyGenerator utiliza una cadena concreta como clave de ELS. Aunque es poco probable, es posible que la clave pueda entrar en conflicto con otra clave que utilice la aplicación. En lugar de utilizar la clave predeterminada, puede que desee especificar su propia clave de ELS. En este caso, especifique una clave personalizada transmitiéndola como segundo parámetro `getEncryptionKey()`, tal y como se muestra a continuación:

```
var customKey:String = "My custom ELS salt key";
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password, customKey);
```

7 Creación o apertura de la base de datos

Con una clave de cifrado devuelta por el método `getEncryptionKey()`, el código puede crear una nueva base de datos cifrada o intentar abrir la base de datos cifrada existente. En cualquier caso, se utiliza el método `open()` u `openAsync()` de la clase SQLConnection, tal y como se describe en "Creación de una base de datos cifrada" en la página 791 y "Conexión con una base de datos cifrada" en la página 791.

En este ejemplo, la aplicación está diseñada para abrir la base de datos en modo de ejecución asíncrono. El código establece los detectores de eventos apropiados y llama al método `openAsync()`, transmitiendo la clave de cifrado como argumento final:

```
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
```

En los métodos del detector, el código elimina los registros del detector de eventos. Después muestra un mensaje de estado que indica si se creó o se abrió la base de datos o si se produjo un error. La parte más notable de estos controladores de eventos está en el método `openError()`. En ese método, una declaración `if` comprueba si existe la base de datos (lo que significa que el código está intentando conectarse a una base de datos existente) y si el ID de error coincide con la constante `EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID`. Si ambas condiciones se definen como true, probablemente significa que la contraseña introducida por el usuario no es correcta. (También podría significar que el archivo especificado no es un archivo de base de datos.) A continuación se muestra el código que comprueba el ID de error:

```
if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.text = "Incorrect password!";
}
else
{
    statusMsg.text = "Error creating or opening database.";
}
```

Para obtener el código completo de los detectores de eventos de ejemplo, consulte "Código de ejemplo completo para generar y utilizar una clave de cifrado" en la página 797.

Código de ejemplo completo para generar y utilizar una clave de cifrado

Adobe AIR 1.5 y posterior

A continuación se muestra el código completo para la aplicación de ejemplo "Generación y uso de claves de cifrado." El código consta de dos partes.

El ejemplo utiliza la clase EncryptionKeyGenerator para crear una clave de cifrado a partir de una contraseña. La clase EncryptionKeyGenerator se incluye en el proyecto (as) de biblioteca principal de ActionScript 3.0 de código abierto (3corelib). Puede descargar el paquete as3corelib, incluyendo el código fuente y la documentación. También puede descargar el archivo SWC o los archivos de código fuente desde la página del proyecto.

Ejemplo de Flex

El archivo MXML de la aplicación contiene el código fuente para una aplicación sencilla que crea o abre una conexión con una base de datos cifrada:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted
database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must
contain at least one lowercase letter, at least one uppercase letter, and at least one number
or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);
```

```
                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next
time you open the application, you will need to re-enter the password to open the database
again." width="75%" height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

Ejemplo de Flash Professional

El archivo FLA de la aplicación contiene el código fuente para una aplicación sencilla que crea o abre una conexión con una base de datos cifrada. El archivo FLA dispone de cuatro componentes situados en el escenario:

Nombre de la instancia	Tipo de componente	Descripción
`instructions`	Label	Contiene las instrucciones dadas al usuario.
`passwordInput`	TextInput	Campo de entrada donde el usuario indica la contraseña.
`openButton`	Button	Botón en el que el usuario hace clic una vez introducida la contraseña.
`statusMsg`	Label	Muestra mensajes de estado (errores u operaciones correctas).

El código para la aplicación se define en el fotograma clave del fotograma 1 de la línea de tiempo principal. A continuación se incluye el código para la aplicación:

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time
you open the application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
```

```
    {
        statusMsg.text = "Please specify a password.";
        return;
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);

    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

Aspectos básicos de la clase EncryptionKeyGenerator

Adobe AIR 1.5 y posterior

No es necesario comprender el funcionamiento interno de la clase EncryptionKeyGenerator para utilizarla para crear una clave de cifrado segura para la base de datos de la aplicación. El proceso de uso de la clase se explica en "Uso de la clase EncryptionKeyGenerator para obtener una clave de cifrado segura" en la página 795. Sin embargo, puede que le resulte útil conocer las técnicas que utiliza esta clase. Por ejemplo, puede desear adaptar la clase o incorporar algunas de sus técnicas para situaciones donde se requiere un nivel diferente de privacidad de datos.

La clase EncryptionKeyGenerator se incluye en el proyecto (as) de biblioteca principal de ActionScript 3.0 de código abierto (3corelib). Puede descargar el paquete as3corelib, incluyendo el código fuente y la documentación. También puede ver el código fuente en el sitio del proyecto o descargarlo junto con las explicaciones.

Si el código crea una instancia de EncryptionKeyGenerator y llama a su método `getEncryptionKey()`, se llevan a cabo varios pasos para garantizar que únicamente el usuario adecuado pueda acceder a los datos. El proceso es el mismo para generar una clave de cifrado a partir de la contraseña introducida por el usuario antes de que se cree la base de datos, así como para volver a crear la clave de cifrado para abrir la base de datos.

Obtención y validación de una contraseña segura

Adobe AIR 1.5 y posterior

Cuando el código llama al método `getEncryptionKey()`, transmite una contraseña como parámetro. La contraseña se debe usar como base para la clave de cifrado. Con el uso de una parte de información que solo conoce el usuario, este diseño garantiza que únicamente el usuario que conoce la contraseña pueda acceder a la información de la base de datos. Aunque un atacante acceda a la cuenta del usuario en el equipo, el atacante no podrá tener acceso a la base de datos sin conocer la contraseña. Para obtener la máxima seguridad, la aplicación nunca almacena la contraseña.

El código de una aplicación crea una instancia de EncryptionKeyGenerator y llama a su método `getEncryptionKey()`, pasando una contraseña introducida por el usuario como argumento (variable `password` en este ejemplo):

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

El primer paso que adopta la clase EncryptionKeyGenerator cuando se llama al método `getEncryptionKey()` consiste en comprobar la contraseña introducida por el usuario para garantizar que cumple con los requisitos de longitud de contraseña. La clase EncryptionKeyGenerator requiere que una contraseña incluya de 8 a 32 caracteres. La contraseña debe incluir una mezcla de letras mayúsculas y minúsculas y al menos un número o carácter de símbolos.

La expresión regular que comprueba este modelo se define como una constante denominada `STRONG_PASSWORD_PATTERN`:

```
private static const STRONG_PASSWORD_PATTERN:RegExp =
/(?=^.{8,32}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

El código que comprueba la contraseña se encuentra en el método `validateStrongPassword()` de la clase EncryptionKeyGenerator. El código se presenta del siguiente modo:

```
public function vaidateStrongPassword(password:String):Boolean
{
    if (password == null || password.length <= 0)
    {
        return false;
    }

    return STRONG_PASSWORD_PATTERN.test(password))
}
```

Internamente, el método `getEncryptionKey()` llama a `validateStrongPassword()` de la clase EncryptionKeyGenerator y, si la contraseña no es válida, emite una excepción. `validateStrongPassword()` es un método público, por lo que el código de la aplicación puede comprobar una contraseña sin llamar al método `getEncryptionKey()` para evitar que se produzca un error.

Ampliación de la contraseña a 256 bits

Adobe AIR 1.5 y posterior

Más adelante en el proceso, es necesario que la contraseña tenga una longitud de 256 bits. En lugar de requerir que cada usuario indique una contraseña que sea exactamente de una longitud de 256 bits (32 caracteres), el código crea una contraseña más larga mediante la repetición de sus caracteres.

El método `getEncryptionKey()` llama al método `concatenatePassword()` para llevar a cabo la tarea de crear la contraseña larga.

```
var concatenatedPassword:String = concatenatePassword(password);
```

A continuación se incluye el código para el método `concatenatePassword()`:

```
private function concatenatePassword(pwd:String):String
{
    var len:int = pwd.length;
    var targetLength:int = 32;

    if (len == targetLength)
    {
        return pwd;
    }

    var repetitions:int = Math.floor(targetLength / len);
    var excess:int = targetLength % len;

    var result:String = "";

    for (var i:uint = 0; i < repetitions; i++)
    {
        result += pwd;
    }

    result += pwd.substr(0, excess);

    return result;
}
```

Si la contraseña es inferior a 256 bits, el código concatena la contraseña con sí misma para que tenga 256. Si la longitud no es la deseada exactamente, la última repetición se reduce para que se obtengan justo 256 bits.

Generación o recuperación de un valor salt de 256 bits

Adobe AIR 1.5 y posterior

El siguiente paso consiste en obtener un valor salt de 256 bits que en un paso posterior se combina con la contraseña. Un valor *salt* es un valor aleatorio que se añade o se combina con un valor introducido por el usuario para formar una contraseña. El uso de un valor salt con una contraseña asegura que aunque un usuario elija una palabra real o un término común como contraseña, la combinación de contraseña y valor salt que utiliza el sistema sea un valor aleatorio. Esto ayuda de forma aleatoria en la protección frente a un ataque de diccionario, donde el atacante utiliza una lista de palabras para intentar adivinar una contraseña. Asimismo, con la generación del valor salt y su almacenamiento en el almacén local cifrado, se asocia a la cuenta del usuario en el equipo en el que se ubica el archivo de base de datos.

Si la aplicación llama al método `getEncryptionKey()` por primera vez, el código crea un valor salt aleatorio de 256 bits. De lo contrario, el código carga el valor salt desde el almacén local cifrado.

El valor salt se almacena en una variable denominada `salt`. El código determina si ya se ha creado un valor salt, intentando cargar el valor desde el almacén local cifrado:

```
var salt:ByteArray = EncryptedLocalStore.getItem(saltKey);
if (salt == null)
{
    salt = makeSalt();
    EncryptedLocalStore.setItem(saltKey, salt);
}
```

Si el código crea un nuevo valor salt, el método `makeSalt()` genera un valor aleatorio de 256 bits. Debido a que el valor se almacena finalmente en el almacén local cifrado, se genera como un objeto ByteArray. El método `makeSalt()` utiliza el método `Math.random()` para generar el valor de forma aleatoria. El método `Math.random()` no puede generar 256 bits de una sola vez. El código utiliza un bucle para llamar a `Math.random()` ocho veces. Cada vez, se genera un valor uint aleatorio entre 0 y 4294967295 (valor uint máximo). El valor uint se utiliza para mayor comodidad, ya que un valor de este tipo usa exactamente 32 bits. Al escribir ocho valores uint en ByteArray, se genera un valor de 256 bits. A continuación se incluye el código para el método `makeSalt()`:

```
private function makeSalt():ByteArray
{
    var result:ByteArray = new ByteArray;

    for (var i:uint = 0; i < 8; i++)
    {
        result.writeUnsignedInt(Math.round(Math.random() * uint.MAX_VALUE));
    }

    return result;
}
```

Cuando el código guarda el valor salt en el almacén local cifrado (ELS) o lo recupera desde ELS, necesita una clave String bajo la que se guarda el valor salt. Si no se conoce la clave, el valor salt no podrá recuperarse. En este caso, la clave de cifrado no puede volver a crearse cada vez que se vuelva a abrir la base de datos. De forma predeterminada, EncryptionKeyGenerator utiliza una clave de ELS predefinida que se define en la constante `SALT_ELS_KEY`. En lugar de emplear la clave predeterminada, el código de la aplicación también puede especificar una clave de ELS para utilizarla en la llamada al método `getEncryptionKey()`. Tanto la clave predeterminada como la clave de ELS del valor salt especificada por la aplicación se almacenan en una variable denominada `saltKey`. Esa variable se utiliza en las llamadas a `EncryptedLocalStore.setItem()` y `EncryptedLocalStore.getItem()`, tal y como se ha mostrado anteriormente.

Combinación de la contraseña de 256 bits y el valor salt utilizando el operador XOR

Adobe AIR 1.5 y posterior

El código cuenta ahora con una contraseña de 256 bits y un valor salt de 256 bits. A continuación utiliza una operación XOR en modo bit para combinar el valor salt y la contraseña concatenada en un solo valor. Realmente, esta técnica crea una contraseña de 256 bits que consta de caracteres procedentes del intervalo completo de caracteres posibles. Este principio es verdadero, aunque es más probable que la entrada de contraseña real conste de caracteres alfanuméricos principalmente. Con este grado aumentado de aleatoriedad se obtiene la ventaja de crear el conjunto de posibles contraseñas largas sin que el usuario deba introducir una contraseña extensa y compleja.

El resultado de la operación XOR se almacena en la variable `unhashedKey`. El proceso real de realizar una operación XOR en modo bit en los dos valores se produce en el método `xorBytes()`.

```
var unhashedKey:ByteArray = xorBytes(concatenatedPassword, salt);
```

El operador XOR en modo bit (`^`) adopta dos valores uint y devuelve un valor de este tipo. (Un valor uint contiene 32 bits.) Los valores input transmitidos como argumentos al método `xorBytes()` son un elemento String (contraseña) y un objeto ByteArray (valor salt). Por lo tanto, el código utiliza un bucle para extraer 32 bits cada vez de cada entrada para combinar utilizando el operador XOR.

```
private function xorBytes(passwordString:String, salt:ByteArray):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 4)
    {
        // ...
    }

    return result;
}
```

En el bucle, en primer lugar se extraen 32 bits (4 bytes) del parámetro `passwordString`. Estos bits se extraen y se convierten en un valor uint (`o1`) en un proceso de dos partes. En primer lugar, el método `charCodeAt()` obtiene el valor numérico de cada carácter. A continuación, el valor se desplaza a la posición adecuada en el valor uint utilizando el operador de desplazamiento a la izquierda en modo bit (`<<`) y el valor desplazado se añade a `o1`. Por ejemplo, el primer carácter (`i`) pasa a ser los primeros 8 bits, utilizando el operador de desplazamiento a la izquierda en modo bit (`<<`) para desplazar los bits a la izquierda 24 bits y asignado este valor a `o1`. El segundo carácter `(i + 1)` pasa a ser los segundos 8 bits, desplazando su valor a la izquierda 16 bits y añadiendo el resultado a `o1`. Los valores de los caracteres tercero y cuarto se agregan del mismo modo.

```
        // ...

        // Extract 4 bytes from the password string and convert to a uint
        var o1:uint = passwordString.charCodeAt(i) << 24;
        o1 += passwordString.charCodeAt(i + 1) << 16;
        o1 += passwordString.charCodeAt(i + 2) << 8;
        o1 += passwordString.charCodeAt(i + 3);

        // ...
```

La variable `o1` contiene ahora 32 bits a partir del parámetro `passwordString`. Posteriormente, se extraen 32 bits del parámetro `salt` llamando a su método `readUnsignedInt()`. Los 32 bits se almacenan en la variable uint `o2`.

```
        // ...

        salt.position = i;
        var o2:uint = salt.readUnsignedInt();

        // ...
```

Finalmente, los dos valores (uint) de 32 bits se combinan mediante el operador XOR y el resultado se escribe en un objeto ByteArray llamado `result`.

```
        // ...

        var xor:uint = o1 ^ o2;
        result.writeUnsignedInt(xor);
        // ...
```

Una vez completado el bucle, se devuelve el objeto ByteArray que contiene el resultado XOR.

```
        // ...
    }

    return result;
}
```

Operaciones hash de la clave

Adobe AIR 1.5 y posterior

Una vez combinados el valor salt y la contraseña concatenada, el siguiente paso consiste en asegurar este valor incluyéndolo en hash mediante el algoritmo hash SHA-256. Al incluir en hash el valor, resulta más difícil para un atacante aplicarle una ingeniería inversa.

En este momento el código cuenta con un elemento ByteArray denominado `unhashedKey` que contiene la contraseña concatenada combinada con el valor salt. El proyecto de biblioteca principal (as3corelib) de ActionScript 3.0 incluye una clase SHA256 en el paquete com.adobe.crypto. El método `SHA256.hashBytes()` que realiza un algoritmo hash SHA-256 en ByteArray y devuelve una cadena que contiene el resultado hash de 256 bits como número hexadecimal. El código utiliza la clase EncryptionKeyGenerator para realizar la operación de hash de la clave:

```
var hashedKey:String = SHA256.hashBytes(unhashedKey);
```

Extracción de la clave de cifrado del hash

Adobe AIR 1.5 y posterior

La clave de cifrado debe ser un objeto ByteArray que cuente exactamente con 16 bytes (128 bits). El resultado del algoritmo hash SHA-256 siempre es 256 bits. Por lo tanto, en el paso final se seleccionan 128 bits del resultado de la operación de hash para utilizar como clave de cifrado real.

En la clase EncryptionKeyGenerator, el código reduce la clave a 128 bits llamando al método `generateEncryptionKey()`: Después devuelve el resultado de ese método como resultado de `getEncryptionKey()`:

```
var encryptionKey:ByteArray = generateEncryptionKey(hashedKey);
return encryptionKey;
```

No es necesario utilizar los primeros 128 bits como clave de cifrado. Puede seleccionar un intervalo de bits comenzando en un punto arbitrario, puede optar por cualquier otro bit, o bien, usar otro modo de seleccionar bits. Lo más importante es que el código seleccione 128 bits distintos y que los mismos 128 bits se utilicen cada vez.

En este caso, el método `generateEncryptionKey()` usa el intervalo de bits que comienza en el byte en posición 18 como clave de cifrado. Tal y como se mencionó anteriormente, la clase SHA256 devuelve una cadena que contiene un hash de 256 bits como número hexadecimal. Un solo bloque de 128 bits tiene demasiados bytes para añadir a un elemento ByteArray a la vez. Por ello, el código utiliza un bucle `for` para extraer caracteres de la cadena hexadecimal, convertirlos en valores numéricos reales y agregarlos a ByteArray. La cadena de resultado de SHA-256 tiene 64 caracteres. Un intervalo de 128 bits es igual a 32 caracteres en la cadena, por lo que cada carácter representa 4 bits. El incremento más bajo de datos que se puede agregar a ByteArray es un byte (8 bits), lo que equivale a dos caracteres en la cadena `hash`. Por lo tanto, el bucle cuenta de 0 a 31 (32 caracteres) en incrementos de 2 caracteres.

En el bucle, el código determina en primer lugar la posición de inicio del par actual de caracteres. Debido a que el intervalo deseado comienza en el carácter en la posición de índice 17 (byte 18), a la variable `position` se le asigna el valor del iterador actual (`i`) más 17. El código utiliza el método `substr()` del objeto String para extraer los dos caracteres en la posición actual. Estos caracteres se almacenan en la variable `hex`. A continuación, el código utiliza el método `parseInt()` para convertir la cadena `hex` en un valor entero decimal. Almacena este valor en la variable int `byte`. Finalmente, el código agrega el valor en `byte` al objeto ByteArray`result` utilizando su método `writeByte()`. Cuando finaliza el bucle, el objeto ByteArray `result` contiene 16 bytes y está listo para utilizarse como clave de cifrado de la base de datos.

```
private function generateEncryptionKey(hash:String):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 2)
    {
        var position:uint = i + 17;
        var hex:String = hash.substr(position, 2);
        var byte:int = parseInt(hex, 16);
        result.writeByte(byte);
    }

    return result;
}
```

Estrategias para la utilización de bases de datos SQL

Adobe AIR 1.0 y posterior

Hay diferentes maneras en que una aplicación puede acceder y utilizar una base de datos SQL local. El diseño de la aplicación puede variar en cuanto a la organización del código de la aplicación, la secuencia y la sincronización de cómo se llevan a cabo las operaciones etc. Las técnicas que elije pueden tener un impacto en la facilidad para desarrollar la aplicación. Pueden afectar la facilidad de modificar la aplicación en futuras actualizaciones. Asimismo pueden afectar el rendimiento de la aplicación desde el punto de vista del usuario.

Distribución de una base de datos llenada previamente

Adobe AIR 1.0 y posterior

Cuando utiliza una base de datos SQL local de AIR en la aplicación, la aplicación espera una base de datos con cierta estructura de tablas, columnas etc. Asimismo, algunas aplicaciones esperan que ciertos datos se introduzcan con anterioridad en el archivo de la base de datos. Una manera de asegurarse que la base de datos cuenta con la correcta estructura es crear la base de datos dentro del código de aplicación. Cuando se carga la aplicación, la misma verifica la existencia del archivo de base de datos en una determinada ubicación. Si el archivo no existe, la aplicación ejecuta un conjunto de comandos para crear el archivo de base de datos, crear la estructura de la base de datos y llenar las tablas con los datos iniciales.

El código que crea la base de datos y las respectivas tablas generalmente es complejo. Con frecuencia, solo se utiliza una vez en la instalación de la aplicación, pero no obstante contribuye al tamaño y a la complejidad de la aplicación. Como alternativa a crear la base de datos, la estructura y los datos mediante programación, se puede distribuir con la aplicación una base de datos previamente llenada. Para distribuir una base de datos predefinida, incluya el archivo de la base de datos en el paquete de AIR de la aplicación.

Como todos los archivos que se incluyen en un paquete de AIR, se instala un archivo de base de datos empaquetado en el directorio de la aplicación (el directorio representado por la propiedad `File.applicationDirectory`). Sin embargo, los archivos en ese directorio son de solo lectura. Use el archivo empaquetado como una base de datos de "plantilla". La primera vez que el usuario ejecuta la aplicación, copie el archivo de la base de datos original en el "Apuntar al directorio de almacenamiento de la aplicación" en la página 700 (u otra ubicación) y use esa base de datos dentro de la aplicación.

Prácticas recomendadas para utilizar bases de datos SQL locales

Adobe AIR 1.0 y posterior

En la siguiente lista se muestra un conjunto de técnicas recomendadas que se pueden utilizar para mejorar el rendimiento, la seguridad y la facilidad de mantener las aplicaciones cuando utiliza bases de datos SQL locales.

Crear previamente conexiones a la base de datos

Adobe AIR 1.0 y posterior

Aun si la aplicación no ejecuta declaraciones cuando se carga, cree una instancia al objeto SQLConnection y llame al método `open()` o `openAsync()` con antelación (como por ejemplo después del inicio de la aplicación) para evitar demoras cuando se ejecutan las declaraciones. Consulte "Conexión a una base de datos" en la página 757

Volver a utilizar las conexiones a la base de datos

Adobe AIR 1.0 y posterior

Si accede a una determinada base de datos durante el tiempo de ejecución de la aplicación, mantenga una referencia a la instancia SQLConnection y vuelva a utilizarla en la aplicación, en lugar de cerrar y volver a abrir la conexión. Consulte "Conexión a una base de datos" en la página 757

Utilizar el modo de ejecución asíncrono

Adobe AIR 1.0 y posterior

Cuando se escribe código de acceso a datos, puede ser tentador ejecutar las operaciones sincrónicamente en vez de forma asíncrona, ya que el uso de operaciones sincrónicas con frecuencia requieren un código más breve y menos complejo. Sin embargo, como se describe en "Uso de operaciones sincrónicas y asíncronas de base de datos" en la página 785, las operaciones sincrónicas pueden impactar en el rendimiento que es obvio para los usuarios y perjudicial en el uso de la aplicación. La cantidad de tiempo que tarda una sola operación varía según la operación y en particular la cantidad de datos que involucra. Por ejemplo, una declaración SQL `INSERT` que solo añade una fila a la base de datos tarda menos que una declaración `SELECT` que recupera miles de filas de datos. Sin embargo, cuando utiliza una ejecución sincrónica para realizar varias operaciones, en general las operaciones de agrupan. Si bien el tiempo que demora cada operación individual es muy corto, la aplicación se bloquea hasta que terminen todas las operaciones sincrónicas. Como resultado, el tiempo acumulado de varias operaciones agrupadas puede ser suficiente para bloquear la aplicación.

Utilice operaciones asíncronas como un método estándar, especialmente con operaciones que involucran grandes cantidades de filas. Existe una técnica para dividir el procesamientos de grandes conjuntos de resultados de la declaración `SELECT` que se describe en "Recuperación de resultados SELECT en partes" en la página 773. Sin embargo, esta técnica solo se puede utilizar en el modo de ejecución asíncrono. Solo utilice operaciones sincrónicas cuando no puede lograr cierta funcionalidad usando la programación asíncrona, cuando haya considerado las desventajas en el rendimiento que afrontarán los usuarios de la aplicación y cuando haya probado la aplicación para que sepa como se ve afectado el rendimiento de la aplicación. El uso de la ejecución asíncrona puede implicar una codificación más compleja. Sin embargo, recuerde que solo tiene que escribir el código una sola vez pero los usuarios de la aplicación tienen que utilizarlo repetidas veces, de forma veloz o lenta.

En muchos casos, al utilizar una instancia SQLStatement por separado para que cada declaración SQL se ejecute, se pueden poner en cola varias operaciones SQL a la vez, que hace que el código asíncrono sea como el código sincrónico en cuanto a cómo está escrito el código. Para más información, consulte "Aspectos básicos del modelo de ejecución asíncrono" en la página 788.

Utilizar declaraciones SQL por separado y no cambiar la propiedad text de la declaración SQLStatement

Adobe AIR 1.0 y posterior

Para cualquier declaración SQL que se ejecuta más de una vez en una aplicación, cree una instancia SQLStatement por separado para cada declaración SQL. Use esa instancia SQLStatement cada vez que se ejecuta el comando SQL. Por ejemplo, supongamos que está creando una aplicación que incluye cuatro operaciones SQL diferentes que se llevan a cabo múltiples veces. En ese caso, cree cuatro instancias SQLStatement por separado y llame al método `execute()` de cada instancia para ejecutarla. Evite la alternativa de utilizar una sola instancia SQLStatement para todas las declaraciones SQL, redefiniendo la propiedad `text` cada vez antes de ejecutar la declaración.

Utilizar parámetros de declaración

Adobe AIR 1.0 y posterior

Utilice parámetros SQLStatement, nunca concatene las entradas del usuario en un texto de declaración. La utilización de parámetros hace que la aplicación sea más segura ya que evita la posibilidad de ataques de inyección SQL. Hace posible usar objetos en consultas (en lugar de solo valores literales SQL). Asimismo hace que las declaraciones se ejecuten más eficientemente ya que se pueden volver a utilizar sin la necesidad de recompilar cada vez que se ejecutan. Consulte "Uso de parámetros en sentencias" en la página 761 para más información.

Capítulo 41: Trabajo con conjuntos de bytes

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ByteArray permite leer y escribir en un flujo de datos binarios, que es en esencia un conjunto de bytes. Esta clase ofrece una forma de acceder a datos al nivel más elemental. Dado que los datos informáticos constan de bytes (grupos de 8 bits), la capacidad de leer datos en bytes significa que se puede tener acceso a datos para los cuales no existen clases y métodos de acceso. La clase ByteArray permite analizar a nivel de bytes cualquier flujo de datos, desde un mapa de bits hasta un flujo de datos que se transmite por la red.

El método `writeObject()` permite escribir un objeto en formato de mensajes de acción (AMF) serializado en un ByteArray, mientras que el método `readObject()` permite leer un objeto serializado desde un ByteArray en una variable del tipo de datos original. Se puede serializar cualquier objeto con excepción de los objetos de visualización, que son aquellos que se pueden colocar en la lista de visualización. También se pueden asignar objetos serializados en instancias de la clase personalizada si dicha clase está disponible para el motor de ejecución. Tras convertir un objeto en formato AMF, se puede transmitir con eficacia a través de una conexión de red o guardar en un archivo.

La aplicación de Adobe® AIR® de muestra que se describe aquí lee un archivo .zip como ejemplo del procesamiento de un flujo de bytes, extrayendo una lista de los archivos que contiene el archivo .zip y escribiéndolos en el escritorio.

Más temas de ayuda

flash.utils.ByteArray

flash.utils.IExternalizable

Especificación de formato de mensaje de acción

Lectura y escritura de un ByteArray

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase ByteArray forma parte del paquete flash.utils. Para crear un objeto ByteArray en ActionScript 3.0, importe la clase ByteArray e invoque el constructor, como se muestra en el ejemplo siguiente:

```
 import flash.utils.ByteArray;
var stream:ByteArray = new ByteArray();
```

Métodos ByteArray

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Todo flujo de datos significativo se organiza en un formato que se pueda analizar para localizar la información deseada. Una ficha en un archivo sencillo de empleados, por ejemplo, probablemente incluiría un número de ID, nombre, dirección, teléfono, etc. Un archivo de audio MP3 contiene una etiqueta ID3 que identifica el título, autor, álbum, fecha de edición y género del archivo que se descarga. El formato permite saber el orden en que cabe esperar los datos en el flujo de datos. Permite leer el flujo de bytes con inteligencia.

La clase ByteArray incluye varios métodos que facilitan la lectura y escritura en un flujo de datos, entre ellos: `readBytes()` y `writeBytes()`, `readInt()` y `writeInt()`, `readFloat()` y `writeFloat()`, `readObject()` y `writeObject()`, y `readUTFBytes()` y `writeUTFBytes()`. Estos métodos permiten leer datos desde el flujo de datos en variables de tipos de datos específicos y escribir desde tipos de datos específicos directamente en el flujo de datos binarios.

En el siguiente ejemplo el código lee un conjunto sencillo de cadenas y números de punto flotante y escribe cada elemento en un ByteArray. La organización del conjunto permite que el código llame a los métodos adecuados de ByteArray (`writeUTFBytes()` y `writeFloat()`) para escribir los datos. El patrón de datos reiterado hace posible leer el conjunto con un bucle.

```
 // The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

import flash.utils.ByteArray;

// define the grocery list Array
var groceries:Array = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes:ByteArray = new ByteArray();
// for each item in the array
for (var i:int = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        trace("bytes.position is: " + bytes.position);//display the position in ByteArray
}
trace("bytes length is: " +  bytes.length);// display the length
```

Propiedad position

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La propiedad "position" guarda la posición actual del puntero que indexa el ByteArray durante la lectura o escritura. El valor inicial de la propiedad "position" es 0 (cero) como se muestra en el código siguiente:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
```

Cuando se lee o escribe en un ByteArray, el método empleado actualiza la propiedad de posición para que apunte al lugar inmediatamente después del último byte leído o escrito. En el siguiente ejemplo, el código escribe una cadena en un ByteArray y después la propiedad de posición apunta al byte que sigue a dicha cadena en el ByteArray:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
```

Asimismo, una operación de lectura incrementa la propiedad de posición con el número de bytes leídos.

```
var bytes:ByteArray = new ByteArray();

trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

Obsérvese que se puede definir la propiedad de posición en un lugar concreto del ByteArray para leer o escribir en esa posición desplazada.

Propiedades bytesAvailable y length

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las propiedades `length` y `bytesAvailable` indican la longitud de un ByteArray y cuántos bytes quedan desde la posición actual hasta el final. El ejemplo siguiente muestra cómo utilizar estas propiedades. En el ejemplo se escribe una cadena de texto en el ByteArray y después se lee el ByteArray byte por byte hasta llegar al carácter "a" o al final (`bytesAvailable <= 0`).

```
var bytes:ByteArray = new ByteArray();
var text:String = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    trace("Found the letter a; position is: " + bytes.position); // 23
    trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

Propiedad endian

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los ordenadores pueden variar en la forma en que guardan números que requieren varios bytes de memoria. Un entero, por ejemplo, puede necesitar 4 bytes (32 bits) de memoria. En alguno ordenadores se guarda primero el byte más significativo del número, en la dirección de memoria más baja, y en otros se guarda primero el byte menos significativo. Este atributo del ordenador, o de la ordenación de bytes, se denomina *big endian* (primero el byte más significativo) o *little endian* (primero el byte menos significativo). Por ejemplo: el número 0x31323334 se guardaría de la forma siguiente con ordenación de bytes big endian y little endian, en que a0 representa la dirección de memoria más baja de los 4 bytes y a3 representa la más alta:

Big endian	**Big endian**	**Big endian**	**Big endian**
a0	a1	a2	a3
31	32	33	34

Little endian	**Little endian**	**Little endian**	**Little endian**
a0	a1	a2	a3
34	33	32	31

La propiedad `endian` de la clase ByteArray permite indicar este orden de bytes para los números de varios bytes que se procesen. Los valores aceptables para esta propiedad son `"bigEndian"` o `"littleEndian"` y la clase Endian define las constantes `BIG_ENDIAN` y `LITTLE_ENDIAN` para definir la propiedad `endian` con estas cadenas.

Métodos compress() y uncompress()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `compress()` permite comprimir un ByteArray de acuerdo con un algoritmo de compresión especificado como parámetro. El método `uncompress()` permite descomprimir un ByteArray comprimido de acuerdo con un algoritmo de compresión. Tras llamar a `compress()` y `uncompress()` se define la nueva longitud del conjunto de bytes y la propiedad de posición se define en el final.

La clase CompressionAlgorithm define las constantes que se pueden utilizar para especificar el algoritmo de compresión. La clase ByteArray admite los algoritmos deflate (solo AIR), zlib e lzma. El formato de datos comprimidos zlib se describe en http://www.ietf.org/rfc/rfc1950.txt. El algoritmo lzma se añadió en Flash Player 11.4 y AIR 3.4. Encontrará una descripción en http://www.7-zip.org/7z.html.

El algoritmo de compresión deflate se utiliza en varios formatos de compresión, como zlib, gzip y algunas implementaciones de zip. El algoritmo de compresión deflate se describe en http://www.ietf.org/rfc/rfc1951.txt.

En el siguiente ejemplo se comprime un ByteArray denominado `bytes` con el algoritmo Izma:

```
bytes.compress(CompressionAlgorithm.LZMA);
```

En el siguiente ejemplo se descomprime un ByteArray comprimido con el algoritmo deflate:

```
bytes.uncompress(CompressionAlgorithm.LZMA);
```

Lectura y escritura de objetos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `readObject()` y `writeObject()` leen y escriben un objeto en un ByteArray, codificado en formato de mensaje de acción (AMF) serializado. AMF es un protocolo de mensajes propio de Adobe que se utiliza en diversas clases de ActionScript 3.0, entre ellas Netstream, NetConnection, NetStream, LocalConnection y Shared Objects.

Se utiliza un marcador de un byte para describir el tipo de datos codificados que siguen. AMF utiliza los siguientes 13 tipos de datos:

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

Los datos codificados siguen al marcador de tipo, a menos que el marcador represente un solo valor posible (como "null", "true" o "false"), en cuyo caso no se codifica nada más.

Existen dos versiones de AMF: AMF0 y AMF3. AMF 0 es compatible con la transmisión de objetos complejos por referencia y admite puntos finales para restaurar relaciones de objetos. AMF 3 representa una mejora de AMF 0 al enviar cadenas y características de objetos mediante referencia, además de referencias de objetos, y al admitir nuevos tipos de datos que se introdujeron en ActionScript 3.0. La propiedad `ByteArray.objectEcoding` especifica la versión de AMF que se utiliza para codificar los datos de objeto. La clase flash.net.ObjectEncoding define las constantes para especificar la versión de AMF: `ObjectEncoding.AMF0` y `ObjectEncoding.AMF3`.

En el siguiente ejemplo se llama a `writeObject()` para escribir un objeto XML en un ByteArray, que a continuación se comprime con el algoritmo deflate y se escribe en el archivo `order` en el escritorio. En el ejemplo se utiliza una etiqueta para presentar el mensaje “Wrote order file to desktop!” en la ventana de AIR una vez terminada la operación.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;
public class WriteObjectExample extends Sprite
{
    public function WriteObjectExample()
    {
        var bytes:ByteArray = new ByteArray();
        var myLabel:TextField = new TextField();
        myLabel.x = 150;
        myLabel.y = 150;
        myLabel.width = 200;
        addChild(myLabel);

        var myXML:XML =
            <order>
                <item id='1'>
                    <menuName>burger</menuName>
                    <price>3.95</price>
                </item>
                <item id='2'>
                    <menuName>fries</menuName>
                    <price>1.45</price>
                </item>
            </order>;

        // Write XML object to ByteArray
        bytes.writeObject(myXML);
        bytes.position = 0;//reset position to beginning
        bytes.compress(CompressionAlgorithm.DEFLATE);// compress ByteArray
        writeBytesToFile("order.xml", bytes);
        myLabel.text = "Wrote order file to desktop!";
    }

    private function writeBytesToFile(fileName:String, data:ByteArray):void
    {
        var outFile:File = File.desktopDirectory; // dest folder is desktop
        outFile = outFile.resolvePath(fileName);  // name of file to write
        var outStream:FileStream = new FileStream();
        // open output file stream in WRITE mode
        outStream.open(outFile, FileMode.WRITE);
        // write out the file
        outStream.writeBytes(data, 0, data.length);
        // close it
        outStream.close();
    }
}
```

El método `readObject()` lee un objeto de un ByteArray en AMF serializado y lo guarda en un objeto del tipo especificado. En el siguiente ejemplo se lee el archivo `order` del escritorio para ponerlo en un ByteArray (`inBytes`), se descomprime y se llama a `readObject()` para guardarlo en el objeto XML `orderXML`. En el ejemplo se utiliza una construcción de bucle `for each()` para añadir cada nodo a un área de texto para su visualización. En el ejemplo se muestra también el valor de la propiedad `objectEncoding` junto con una cabecera para el contenido del archivo `order`.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;

public class ReadObjectExample extends Sprite
{
    public function ReadObjectExample()
    {
        var inBytes:ByteArray = new ByteArray();
        // define text area for displaying XML content
        var myTxt:TextField = new TextField();
        myTxt.width = 550;
        myTxt.height = 400;
        addChild(myTxt);
        //display objectEncoding and file heading
        myTxt.text = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";
        readFileIntoByteArray("order", inBytes);

        inBytes.position = 0; // reset position to beginning
        inBytes.uncompress(CompressionAlgorithm.DEFLATE);
        inBytes.position = 0;//reset position to beginning
        // read XML Object
        var orderXML:XML = inBytes.readObject();

        // for each node in orderXML
        for each (var child:XML in orderXML)
        {
            // append child node to text area
            myTxt.text += child + "\n";
        }
    }

    // read specified file into byte array
    private function readFileIntoByteArray(fileName:String, data:ByteArray):void
    {
        var inFile:File = File.desktopDirectory; // source folder is desktop
        inFile = inFile.resolvePath(fileName);  // name of file to read
        var inStream:FileStream = new FileStream();
        inStream.open(inFile, FileMode.READ);
        inStream.readBytes(data);
        inStream.close();
    }
}
```

Ejemplo de ByteArray: Lectura de un archivo .zip

Adobe AIR 1.0 y posterior

Este ejemplo muestra cómo leer un archivo .zip sencillo que contiene varios archivos de diversos tipos. Para realizarlo, se extraen los datos pertinentes de los metadatos para cada archivo, se descomprime cada archivo para ponerlo en un ByteArray y se escribe el archivo en el escritorio.

La estructura general de un archivo .zip se basa en la especificación de PKWARE Inc., que se mantiene en http://www.pkware.com/documents/casestudies/APPNOTE.TXT. Primero hay una cabecera de archivo y datos de archivo para el primer archivo individual del archivo .zip, seguido de un par de cabecera-datos de archivo para cada archivo adicional. (La estructura de la cabecera de archivo se describe más adelante). A continuación, el archivo .zip puede incluir como opción un registro de descriptor de datos (generalmente al crear el archivo zip en la memoria, más que al guardarlo en un disco). A continuación vienen varios elementos opcionales más: cabecera de descifrado del archivo comprimido, registro de datos adicionales del archivo comprimido, estructura del directorio central, registro de fin del directorio central Zip64, localizador de fin del directorio central Zip64, y registro de fin del directorio central.

El código en este ejemplo se escribe para que solo se analicen los archivos zip que no contengan carpetas y no espera que haya registros de descripción de datos. Pasa por alto toda la información que haya después del último dato del archivo.

El formato de la cabecera de archivo para cada archivo es el siguiente:

firma de cabecera de archivo	4 bytes
versión necesaria	2 bytes
indicador de bits universal	2 bytes
método de compresión	2 bytes (8=DEFLATE; 0=UNCOMPRESSED)
hora de última modificación del archivo	2 bytes
fecha de última modificación del archivo	2 bytes
crc-32	4 bytes
tamaño comprimido	4 bytes
tamaño descomprimido	4 bytes
longitud de nombre de archivo	2 bytes
longitud de campo adicional	2 bytes
nombre de archivo	variable
campo adicional	variable

Después de la cabecera del archivo vienen los datos del archivo, que pueden estar comprimidos o sin comprimir, según el indicador de método de compresión. El indicador será 0 (cero) si los datos están sin comprimir, u 8 si los datos están comprimidos con el algoritmo DEFLATE, u otro valor para otros algoritmos de compresión.

La interfaz de usuario para este ejemplo consta de una etiqueta y un área de texto (`taFiles`). La aplicación escribe la información siguiente en el área de texto para cada archivo que encuentra en el archivo .zip: el nombre del archivo, el tamaño comprimido y el tamaño sin comprimir. El siguiente documento MXML define la interfaz de usuario para la versión Flex de la aplicación:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

El principio del programa realiza las siguientes tareas:

- Importa las clases que se requieran.

```
import flash.filesystem.*;
import flash.utils.ByteArray;
import flash.events.Event;
```

- Define la interfaz de usuario para Flash.

```
import fl.controls.*;

//requires TextArea and Label components in the Library
var taFiles = new TextArea();
var output = new Label();
taFiles.setSize(320, 150);
taFiles.move(10, 30);
output.move(10, 10);
output.width = 150;
output.text = "Contents of HelloAir.zip";
addChild(taFiles);
addChild(output);
```

- Define el ByteArray `bytes`.

```
var bytes:ByteArray = new ByteArray();
```

- Define variables para guardar los metadatos de la cabecera del archivo.

```
// variables for reading fixed portion of file header
var fileName:String = new String();
var flNameLength:uint;
var xfldLength:uint;
var offset:uint;
var compSize:uint;
var uncompSize:uint;
var compMethod:int;
var signature:int;
```

- Define los objetos File (`zfile`) y FileStream (`zStream`) para representar el archivo .zip, y especifica la ubicación del archivo .zip del que se extrajeron los archivos (un archivo llamado “HelloAIR.zip” en el directorio del escritorio).

```
// File variables for accessing .zip file
var zfile:File = File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream:FileStream = new FileStream();
```

En Flex, el código del programa empieza en el método `init()`, al que se llama como controlador de `creationComplete` para la etiqueta raíz `mx:WindowedApplication`.

```
// for Flex
private function init():void
{
```

El programa empieza por abrir el archivo .zip en modo READ (lectura).

```
    zStream.open(zfile, FileMode.READ);
```

A continuación define la propiedad `endian` de `bytes` en `LITTLE_ENDIAN` para indicar que el orden de bytes de los campos numéricos es con el byte menos significativo primero.

```
    bytes.endian = Endian.LITTLE_ENDIAN;
```

Seguidamente, la sentencia `while()` inicia un bucle que continúa hasta que la posición actual en la secuencia de archivos sea superior o igual al tamaño del archivo.

```
    while (zStream.position < zfile.size)
    {
```

La primera sentencia del bucle lee los 30 primeros bytes de la secuencia de archivos para ponerlo en el ByteArray `bytes`. Los 30 primeros bytes conforman la parte de tamaño fijo de la cabecera del primer archivo.

```
        // read fixed metadata portion of local file header
        zStream.readBytes(bytes, 0, 30);
```

A continuación el código lee un entero (`signature`) en los primeros bytes de la cabecera de 30 bytes. La definición del formato ZIP estipula que la firma de cada cabecera de archivo es el valor hexadecimal `0x04034b50`; si la firma es distinta, significa que el código se refiere a la parte del archivo .zip que es ajena a los archivos y no hay más archivos que extraer. En ese caso el código sale inmediatamente del bucle `while` en lugar de esperar hasta alcanzar el final del conjunto de bytes.

```
        bytes.position = 0;
        signature = bytes.readInt();
        // if no longer reading data files, quit
        if (signature != 0x04034b50)
        {
            break;
        }
```

La siguiente parte del bucle lee el byte de la cabecera en la posición de desplazamiento 8 y guarda el valor en la variable `compMethod`. Este byte contiene un valor que indica el método de compresión que se utilizó para comprimir este archivo. Se admiten varios métodos de compresión, pero en la práctica para casi todos los archivos .zip se utiliza el algoritmo de compresión DEFLATE. Si el archivo actual está comprimido con compresión DEFLATE, `compMethod` es 8; si el archivo está sin comprimir, `compMethod` es 0.

```
        bytes.position = 8;
        compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

A los 30 primeros bytes sigue una parte de longitud variable de la cabecera que contiene el nombre del archivo y, tal vez, un campo adicional. El tamaño de esta parte se guarda en la variable `offset`. El tamaño se calcula sumando la longitud del nombre del archivo y la longitud del campo adicional, leídas en la cabecera en las posiciones de desplazamiento 26 y 28.

```
        offset = 0;// stores length of variable portion of metadata
        bytes.position = 26;  // offset to file name length
        flNameLength = bytes.readShort();// store file name
        offset += flNameLength; // add length of file name
        bytes.position = 28;// offset to extra field length
        xfldLength = bytes.readShort();
        offset += xfldLength;// add length of extra field
```

A continuación el programa lee la parte de longitud variable de la cabecera de archivo donde se indica la cantidad de bytes guardados en la variable `offset`.

```
        // read variable length bytes between fixed-length header and compressed file data
        zStream.readBytes(bytes, 30, offset);
```

El programa lee el nombre del archivo en la parte de longitud variable de la cabecera y lo muestra en el área de texto junto con los tamaños del archivo comprimido (en el archivo zip) y sin comprimir (original).

```
// Flash version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.appendText(fileName + "\n"); // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.appendText("\tCompressed size is: " + compSize + '\n');
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.appendText("\tUncompressed size is: " + uncompSize + '\n');


// Flex version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.text += fileName + "\n"; // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.text += "\tCompressed size is: " + compSize + '\n';
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.text += "\tUncompressed size is: " + uncompSize + '\n';
```

En el ejemplo se lee el resto del archivo de la secuencia de archivos para ponerlo en `bytes` durante la longitud especificada por el tamaño comprimido, sobrescribiendo la cabecera del archivo en los primeros 30 bytes. El tamaño comprimido es exacto aunque el archivo esté sin comprimir porque en ese caso el tamaño comprimido es igual al tamaño del archivo sin comprimir.

```
    // read compressed file to offset 0 of bytes; for uncompressed files
    // the compressed and uncompressed size is the same
    if (compSize == 0) continue;
    zStream.readBytes(bytes, 0, compSize);
```

A continuación el código en el ejemplo descomprime el archivo comprimido y llama a la función `outfile()` para escribirlo en la secuencia de archivos de salida. Pasa a `outfile()` el nombre del archivo y el conjunto de bytes que contiene los datos del archivo.

```
        if (compMethod == 8) // if file is compressed, uncompress
        {
            bytes.uncompress(CompressionAlgorithm.DEFLATE);
        }
        outFile(fileName, bytes);   // call outFile() to write out the file
```

En el ejemplo mencionado anteriormente, `bytes.uncompress(CompressionAlgorithm.DEFLATE)` solo funcionará en las aplicaciones de AIR. Para no compimir los datos desinflados para AIR y Flash Player, invoque la función `inflate()` de ByteArray.

Las llaves finales indican el final del bucle `while` y del método `init()` y el código de la aplicación de Flex, excepto para el método `outFile()`. La ejecución regresa al principio del bucle `while` y sigue procesando los siguientes bytes del archivo .zip, sea extrayendo otro archivo o finalizando el procesamiento del archivo .zip si es que se ha procesado el último archivo.

```
    } // end of while loop
} // for Flex version, end of init() method and application
```

La función `outfile()` abre un archivo de salida en modo WRITE (lectura) en el escritorio y le da el nombre suministrado por el parámetro `filename`. A continuación escribe los datos de los archivos del parámetro `data` en la secuencia de archivos de salida (`outStream`) y cierra el archivo.

```
// Flash version
function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // destination folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}

private function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

Capítulo 42: Fundamentos de redes y comunicación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se crean aplicaciones en Flash Player o en AIR, se suele necesitar tener acceso a recursos externos a la aplicación. Por ejemplo, tal vez quiera enviar una solicitud para una imagen en un servidor web de Internet y obtener los datos de dicha imagen. O tal vez enviar objetos serializados a través de una conexión de socket con un servidor de aplicaciones. Las API de Flash Player y AIR ofrecen varias clases que permiten que las aplicaciones participen en este intercambio. Estas API admiten redes basadas en IP para protocolos como UDP, TCP, HTTP, RTMP y RTMFP.

Se pueden utilizar las siguientes clases para enviar y recibir datos en una red:

Clase	Formatos de datos admitidos	Protocolos	Descripción
Loader	SWF, PNG, JPEG, GIF	HTTP, HTTPS	Carga tipos de datos admitidos y los convierte en un objeto de visualización. Consulte "Carga dinámica de contenido de visualización" en la página 204.
URLLoader	Cualquiera (texto, XML, binario, etc.)	HTTP, HTTPS	Carga formatos arbitrarios de datos. Es responsabilidad de su aplicación interpretar los datos. Consulte "Uso de la clase URLLoader" en la página 848
FileReference	Cualquiera	HTTP	Carga y descarga archivos. Consulte "Uso de la clase FileReference" en la página 679
NetConnection	Vídeo, audio, formato de mensaje de ActionScript (AMF)	HTTP, HTTPS, RTMP, RTMFP	Se conecta a flujos de vídeo, audio y objetos remotos. Consulte "Trabajo con vídeo" en la página 494.
Sound	Audio	HTTP	Carga y reproduce formatos de audio admitidos. Consulte "Carga de archivos de sonido externos" en la página 464.
XMLSocket	XML	TCP	Intercambia mensajes XML con un servidor XMLSocket. Consulte "Sockets XML" en la página 835.
Socket	Cualquiera	TCP	Se conecta a un servidor de socket TCP. Consulte "Sockets de cliente binarios" en la página 830.
SecureSocket (AIR)	Cualquiera	TCP con SSLv3 o TLSv1	Se conecta a un servidor de sockets TCP que requiere seguridad SSL o TLS. Consulte "Sockets de cliente seguros (AIR)" en la página 830.
ServerSocket (AIR)	Cualquiera	TCP	Actúa como servidor para conexiones de socket TCP entrantes. Consulte "Sockets de servidor" en la página 838.
DatagramSocket (AIR)	Cualquiera	UDP	Envía y recibe paquetes UDP. Consulte "Sockets UDP (AIR)" en la página 840

A menudo, cuando se crea una aplicación web resulta útil almacenar información persistente sobre el estado de la aplicación del usuario. Las páginas y las aplicaciones suelen usar cookies para esto. En Flash Player, puede utilizar la clase SharedObject para el mismo fin. Consulte "Objetos compartidos" en la página 731. (La clase SharedObject se puede utilizar en aplicaciones de AIR, pero hay menos restricciones cuando solo se guardan los datos en un archivo normal.)

Cuando la aplicación de Flash Player o de AIR necesita comunicarse con otra aplicación de Flash Player o de AIR en el mismo equipo, puede utilizar la clase LocalConnection. Por ejemplo, dos (o más) archivos SWF de la misma página web pueden comunicarse entre sí. Del mismo modo, un archivo SWF que se ejecute en una página web puede comunicarse con una aplicación de AIR. Consulte "Conexión con otras instancias de Flash Player y AIR" en la página 863.

Cuando necesite comunicarse con otros procesos no SWF del equipo local, puede utilizar la clase NativeProcess añadida en AIR 2. La clase NativeProcess permite a la aplicación de AIR iniciar y comunicarse con otras aplicaciones. Consulte "Comunicación con procesos nativos en AIR" en la página 870.

Cuando necesite información sobre el entorno de red del equipo en el que se ejecuta la aplicación de AIR, puede utilizar las siguientes clases:

- NetworkInfo: proporciona información sobre las interfaces de red disponibles, como la dirección IP del equipo. Consulte "Interfaces de red" en la página 823.
- DNSResolver: permite buscar en registros del DNS. Consulte "Registros del sistema de nombres de dominio (DNS)" en la página 827.
- ServiceMonitor: permite supervisar la disponibilidad de un servidor. Consulte "Supervisión del servicio" en la página 825.
- URLMonitor: permite supervisar la disponibilidad de un recurso en una URL particular. Consulte "Control de HTTP" en la página 826.
- SocketMonitor y SecureSocketMonitor: permite supervisar la disponibilidad de un recurso en un socket. Consulte "Control de Socket" en la página 826.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que aparecerán al programar código de redes y comunicaciones:

Datos externos Datos que se guardan fuera de la aplicación de y se cargan en la aplicación cuando se necesitan. Estos datos pueden estar almacenados en un archivo que se carga directamente o en una base de datos u otra forma, y se recuperan llamando a scripts o programas que se ejecutan en un servidor.

Variables con codificación URL El formato de codificación URL permite representar varias variables (pares de nombre y valor de variables) en una sola cadena de texto. Las variables individuales se escriben con el formato nombre=valor. Cada variable (es decir, cada par nombre-valor) se separa con caracteres ampersand del modo siguiente: variable1=valor1&variable2=valor2. De este modo, es posible enviar un número indefinido de variables en un único mensaje.

Tipo MIME Código estándar empleado para identificar el tipo de un archivo determinado en comunicaciones de Internet. Cualquier tipo de archivo tiene un código específico que se utiliza para su identificación. Cuando se envía un archivo o un mensaje, un programa (como un servidor web o la instancia de Flash Player o AIR de un usuario) especifica el tipo de archivo que se envía.

HTTP protocolo de transferencia de hipertexto. Es un formato estándar para entregar páginas web y otros tipos de contenido enviado a través de Internet.

Método de petición Cuando una aplicación (como una aplicación de AIR o un navegador web) envía un mensaje (denominado solicitud HTTP) a un servidor web, los datos enviados pueden incorporarse en la solicitud de dos maneras distintas: los dos métodos de solicitud GET y POST. En el extremo del servidor, el programa que recibe la solicitud tendrá que mirar en la parte apropiada de la solicitud para encontrar los datos, por lo que el método de solicitud utilizado para enviar datos desde ActionScript debe coincidir con el método de solicitud utilizado para leer los datos en el servidor.

Conexión de socket Conexión constante para permitir la comunicación entre dos ordenadores.

Carga Envío de un archivo a otro equipo.

Descarga Recuperación de un archivo de otro equipo.

Interfaces de red

Adobe AIR 2 y posterior

El objeto NetworkInfo se puede utilizar para descubrir las interfaces de red de hardware y software disponibles en la aplicación. NetworkInfo es un objeto *singleton* y no es necesario crear uno. Utilice la propiedad de clase estática, `networkInfo`, para acceder a un solo objeto NetworkInfo. El objeto NetworkInfo también distribuye un evento `networkChange` cuando cambia una de las interfaces disponibles.

Llame al método `findInterfaces()` para obtener una lista de objetos NetworkInterface. Cada objeto NetworkInterface de la lista describe una de las interfaces disponible. El objeto NetworkInterface proporciona información como la dirección IP, la dirección de hardware, la unidad de transmisión máxima y si la interfaz está activa.

En el siguiente ejemplo de código se realiza el seguimiento de las propiedades de NetworkInterface en cada interfaz del equipo cliente:

```
package {
import flash.display.Sprite;
import flash.net.InterfaceAddress;
import flash.net.NetworkInfo;
import flash.net.NetworkInterface;

public class NetworkInformationExample extends Sprite
{
    public function NetworkInformationExample()
    {
        var networkInfo:NetworkInfo = NetworkInfo.networkInfo;
        var interfaces:Vector.<NetworkInterface> = networkInfo.findInterfaces();

        if( interfaces != null )
        {
            trace( "Interface count: " + interfaces.length );
            for each ( var interfaceObj:NetworkInterface in interfaces )
            {
                trace( "\nname: " + interfaceObj.name );
                trace( "display name: " + interfaceObj.displayName );
                trace( "mtu: "  + interfaceObj.mtu );
```

```
                trace( "active?: " + interfaceObj.active );
                trace( "parent interface: " + interfaceObj.parent );
                trace( "hardware address: " + interfaceObj.hardwareAddress );
                if( interfaceObj.subInterfaces != null )
                {
                    trace( "# subinterfaces: " + interfaceObj.subInterfaces.length );
                }
                trace("# addresses: " + interfaceObj.addresses.length );
                for each ( var address:InterfaceAddress in interfaceObj.addresses )
                {
                    trace( "  type: "           + address.ipVersion );
                    trace( "  address: " + address.address );
                    trace( "  broadcast: " + address.broadcast );
                    trace( "  prefix length: " + address.prefixLength );
                }
            }
        }
    }
}
}
```

Para obtener más información, consulte:

- NetworkInfo
- NetworkInterface
- InterfaceAddress
- Flexpert: Detecting the network connection type with Flex 4.5 (Flexpert: Detección del tipo de conexión de red con Flex 4.5; en inglés)

Cambios de conectividad de la red

Adobe AIR 1.0 y posterior

Es posible que la aplicación de AIR se ejecute en un entorno en que la conectividad de la red es inestable o variable. Para ayudar a una aplicación gestionar la conexión a los recursos en línea, Adobe AIR envía un evento de cambio en la red siempre que se corta o vuelve a disponer de la conexión a la red. Tanto NetworkInfo como el objeto NativeApplication de la aplicación distribuyen el evento `networkChange`. Para reaccionar a este evento, añada un detector:

```
NetworkInfo.networkInfo.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

Defina también una función de controlador de eventos:

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

El evento `networkChange` no indica un cambio en toda la actividad de la red, sino solamente que ha cambiado la conexión. AIR no intenta interpretar el significado del cambio en la red. Un ordenador conectado en red puede tener muchas conexiones reales y virtuales, de modo que si se pierde una conexión, no significa necesariamente que se pierde un recurso. Por otro lado, las conexiones nuevas tampoco garantizan una mejor disponibilidad del recurso. A veces una conexión nueva puede incluso bloquear el acceso a los recursos que antes estaban disponibles (por ejemplo, cuando se realiza una conexión a una VPN).

En general, la única forma de que una aplicación determine si puede conectarse a un recurso remoto es intentar hacerlo. La arquitectura de supervisión del servicio proporciona medios basados en eventos para responder a los cambios de conectividad de red en un host especificado.

Nota: *el marco de supervisión del servicio detecta si un servidor responde a una petición de forma aceptable. Una comprobación correcta no garantiza la total conectividad. Los servicios web escalables hacen uso frecuente de los aparatos de caché y equilibrio de carga para redirigir el flujo de tráfico a un grupo de servidores web. En esta situación, los proveedores de servicios solo ofrecen un diagnóstico parcial de la conectividad de la red.*

Supervisión del servicio

Adobe AIR 1.0 y posterior

El marco de supervisión del servicio, que es independiente de la arquitectura de AIR, reside en el archivo aircore.swc. Para poder utilizar el marco hay que incluir el archivo aircore.swc en el proceso de creación.

Adobe® Flash® Builder incluye esta biblioteca automáticamente.

La clase ServiceMonitor implementa el marco para supervisar los servicios de red y ofrece funciones básicas para los supervisores del servicio. De forma predeterminada, una instancia de la clase ServiceMonitor distribuye eventos relacionados con la conectividad de la red. El objeto ServiceMonitor distribuye estos eventos cuando se crea la instancia y siempre que Adobe AIR detecte un cambio en la red. Se puede además definir la propiedad `pollInterval` de una instancia ServiceMonitor para que compruebe la conectividad en intervalos especificados en milisegundos, independientemente de los eventos de conectividad de la red en general. Un objeto ServiceMonitor no comprueba la conectividad de la red hasta que se haya llamado al método `start()`.

La clase URLMonitor, una subclase de la clase ServiceMonitor, detecta cambios en la conectividad de HTTP para una petición URLRequest especificada.

La clase SocketMonitor, otra subclase de la clase ServiceMonitor, detecta cambios en la conectividad a un host especificado en un puerto especificado.

Nota: *antes de AIR 2, el marco de supervisión del servicio se pubicaba en la biblioteca servicemonitor.swc. Esta biblioteca ya no se utiliza en esta versión. Utilice en su lugar la bibilioteca aircore.swc.*

Flash CS4 y CS5 Professional

Para utilizar estas clases en Adobe® Flash® CS4 o CS5 Professional:

1. Seleccione Archivo > Configuración de publicación.
2. Haga clic en el botón Configuración en la opción para seleccionar ruta de biblioteca de ActionScript 3.0.
3. Haga clic en el botón Navegar hasta el archivo SWC y busque la carpeta AIK en la carpeta de instalación de Flash Professional.
4. En esta carpeta, localice /frameworks/libs/air/aircore.swc (para AIR 2) o /frameworks/libs/air/servicemonitor.swc (para AIR 1.5).
5. Haga clic en el botón Aceptar.

6 Agregue la siguiente sentencia de importación a su código de ActionScript 3.0:

```
import air.net.*;
```

Flash CS3 Professional

Para utilizar estas clases en Adobe® Flash® CS3 Professional, arrastre el componente ServiceMonitorShim del panel Componentes a la Biblioteca y después añada la siguiente sentencia `import` al código ActionScript 3.0:

```
import air.net.*;
```

Control de HTTP

Adobe AIR 1.0 y posterior

La clase URLMonitor determina si se pueden realizar peticiones de HTTP a una dirección especificada en el puerto 80 (el puerto habitual para la comunicación con HTTP). El siguiente código utiliza una instancia de la clase URLMonitor para detectar cambios de conectividad con el sitio web de Adobe:

```
import air.net.URLMonitor;
import flash.net.URLRequest;
import flash.events.StatusEvent;
var monitor:URLMonitor;
monitor = new URLMonitor(new URLRequest('http://www.example.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

Control de Socket

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR también pueden utilizar conexiones de socket para la conectividad modelo "push". Por razones de seguridad, los cortafuegos y encaminadores de red suelen restringir la comunicación por la red a través de puertos no autorizados. Por este motivo los desarrolladores deben tener en cuenta la posibilidad de que los usuarios no puedan realizar conexiones de socket.

El siguiente código utiliza una instancia de la clase SocketMonitor para detectar cambios de conectividad en una conexión de socket. El puerto supervisado es 6667, un puerto común para IRC:

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.example.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

Si el servidor de socket requiere una conexión segura, puede utilizar la clase SecureSocketMonitor en vez de SocketMonitor.

Registros del sistema de nombres de dominio (DNS)

Adobe AIR 2.0 y posterior

Se pueden realizar búsquedas en los registros del recurso DNS utilizando la clase DNSResolver. Los registros del recurso DNS ofrecen información como la dirección IP de un nombre de dominio y el nombre de dominio de una dirección IP. Se pueden realizar búsquedas en los siguientes tipos de registros del recurso DNS:

- ARecord: dirección IPv4 de un host.
- AAAARecord: dirección IPv6 de un host.
- MXRecord: registro de intercambio de correo para un host.
- PTRRecord: nombre de host para una dirección IP.
- SRVRecord: registro de servicio para un servicio..

Para buscar en un registro, se transmite una cadena de consulta y el objeto de la clase que representa el tipo de registro al método `lookup()` del objeto DNSResolver. La cadena de consulta depende del tipo de registro:

Clase de registro	Cadena de consulta	Cadena de consulta de ejemplo
ARecord	Nombre de host	“example.com”
AAAARecord	Nombre de host	“example.com”
MXRecord	Nombre de host	“example.com”
PTRRecord	Dirección IP	“208.77.188.166”
SRVRecord	Identificador de servicio: _service._protocol.host	“_sip._tcp.example.com”

El siguiente ejemplo de código busca la dirección IP del host “example.com”.

```
package
{
    import flash.display.Sprite;
    import flash.events.DNSResolverEvent;
    import flash.events.ErrorEvent;
    import flash.net.dns.ARecord;
    import flash.net.dns.DNSResolver;

    public class DNSResolverExample extends Sprite
    {

        public function DNSResolverExample()
        {
            var resolver:DNSResolver = new DNSResolver();
            resolver.addEventListener( DNSResolverEvent.LOOKUP, lookupComplete );
            resolver.addEventListener( ErrorEvent.ERROR, lookupError );

            resolver.lookup( "example.com.", ARecord );
        }

        private function lookupComplete( event:DNSResolverEvent ):void
        {
            trace( "Query string: " + event.host );
            trace( "Record count: " + event.resourceRecords.length );
            for each( var record:* in event.resourceRecords )
            {
                if( record is ARecord ) trace( record.address );
            }

        }

        private function lookupError( error:ErrorEvent ):void
        {
            trace("Error: " + error.text );
        }
    }
}
```

Para obtener más información, consulte:

- DNSResolver
- DNSResolverEvent
- ARecord
- AAAARecord
- MXRecord
- PTRRecord
- SRVRecord

Capítulo 43: Sockets

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un socket es un tipo de conexión de red establecida entre dos procesos de ordenadores. Normalmente, los procesos se ejecutan en dos equipos distintos conectados a la misma red de Protocolo de Internet (IP). Sin embargo, los procesos conectados se pueden ejecutar en el mismo equipo si se utiliza una dirección IP especial "local host".

Adobe Flash Player admite sockets de protocolo de control de transporte (TCP). Una aplicación de Flash Player se puede conectar a otro proceso que actúe como servidor de socket, pero no puede aceptar solicitudes de conexión entrantes de otros procesos. Dicho de otro modo, una aplicación de Flash Player se puede conectar a un servidor TCP, pero no puede hacer de servidor.

La API de Flash Player también contiene la clase XMLSocket. La clase XMLSocket utiliza un protocolo específico de Flash Player que permite intercambiar mensajes XML con un servidor que comprenda dicho protocolo. La clase XMLSocket se introdujo en ActionScript 1 y sigue admitiéndose para tener compatibilidad con versiones anteriores. En general, la clase Socket debe utilizarse para nuevas aplicaciones a no ser que se esté conectando a un servidor creado específicamente para comunicarse con XMLSockets de Flash.

Adobe AIR añade varias clases adicionales para programación de redes basadas en sockets. Las aplicaciones de AIR pueden actuar como servidores de socket TCP con la clase ServerSocket y pueden conectarse a servidores de socket que requieran seguridad SSL o TLS con la clase SecureSocket. Las aplicaciones de AIR también pueden enviar y recibir mensajes de protocolo de datagrama universal (UDP) con la clase DatagramSocket.

Más temas de ayuda

Paquete flash.net

"Conexión a sockets" en la página 1114

Sockets TCP

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El protocolo TCP (Transmission Control Protocol) proporciona un modo de intercambiar mensajes a través de una conexión de red permanente. TCP garantiza que los mensajes enviados llegan en el orden adecuado (siempre y cuando no haya serios problemas de red). Las conexiones TCP requieren un "cliente" y un "servidor". Flash Player puede crear sockets de cliente. Adobe AIR puede, además, crear sockets de servidor.

Las siguientes API de ActionScript proporcionan conexiones TCP:

- Socket: permite que una aplicación cliente pueda conectarse a un servidor. La clase Socket no detecta conexiones entrantes.
- SecureSocket (AIR): permite que una aplicación cliente pueda conectarse a un servidor de confianza e iniciar una comunicación cifrada.
- ServerSocket (AIR): permite que una aplicación detecte conexiones entrantes y actúe como servidor.
- XMLSocket: permite que una aplicación cliente se conecte a un servidor XMLSocket.

Sockets de cliente binarios

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Una conexión de socket binario es similar a un socket XML, pero el cliente y el servidor no necesitan intercambiar mensajes XML. En lugar de eso, la conexión puede transferir datos como información binaria. Esto permite conectar una amplia gama de servicios, como servidores de correo (POP3, SMTP e IMAP) y servidores de noticias (NNTP).

Clase Socket

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Socket, introducida en ActionScript, permite que el código ActionScript realice conexiones de socket, y que lea y escriba datos binarios sin formato. La clase Socket resulta útil para interactuar con servidores que utilicen protocolos binarios. Se pueden utilizar conexiones de socket binario para escribir código que permita la interacción con varios protocolos de Internet distintos, como POP3, SMTP, IMAP y NNTP. Esto, a su vez, permite a las aplicaciones conectarse a servidores de correos y noticias.

Flash Player puede interactuar con un servidor directamente mediante el protocolo binario de dicho servidor. Algunos servidores utilizan el orden de bytes big-endian y otros emplean little-endian. La mayoría de los servidores de Internet utilizan big-endian, ya que es el "orden de bytes de la red". El orden de bytes littleEndian es popular porque la arquitectura Intel® x86 lo utiliza. Debe utilizarse el orden de bytes Endian que coincida con el orden de bytes del servidor que envía o recibe los datos. Todas las operaciones que se realizan mediante las interfaces IDataInput e IDataOutput, y las clases que implementan dichas interfaces (ByteArray, Socket y URLStream) se codifican de forma predeterminada en formato bigEndian. Esto supone que el byte más significativo va primero. El orden de bytes predeterminado se elige para que coincida con el de Java y con el orden de bytes de red oficial. Para cambiar entre el uso de orden de bytes bigEndian o littleEndian, se puede establecer la propiedad `endian` en `Endian.BIG_ENDIAN` o `Endian.LITTLE_ENDIAN`.

La clase Socket hereda todos los métodos definidos por las interfaces IDataInput e IDataOutput (ubicadas en el paquete flash.utils). Estos métodos se deben utilizar para escribir y leer desde el socket.

Para obtener más información, consulte:

- Socket
- IDataInput
- IDataOutput
- Evento socketData

Sockets de cliente seguros (AIR)

Adobe AIR 2 y posterior

La clase SecureSocket se puede utilizar para conectarse a los servidores socket que utilicen la versión 4 de Secure Sockets Layer (SSLv4) o Transport Layer Security, versión 1 (TLSv1). Un socket seguro ofrece tres ventajas: autenticación del servidor, integridad de los datos y confidencialidad de los mensajes. El motor de ejecución de autentica un servidor utilizando el certificado del servidor y su relación con los certificados raíz o intermedios de la entidad emisora de certificados en el almacén de confianza del usuario. El motor de ejecución se basa en los algoritmos de criptografía utilizados por las implementaciones de protocolo SSL y TLS para proporcionar integridad de los datos y confidencialidad de los mensajes.

Cuando se realiza la conexión a un servidor utilizando el objeto SecureSocket, el motor de ejecución valida el certificado del servidor utilizando el almacén de confianza del certificado. En Windows y Mac, el sistema operativo proporciona el almacén de confianza. En Linux, el motor de ejecución de proporciona su propio almacén de confianza.

Si el certificado del servidor no es válido ni de confianza, el motor de ejecución distribuye un evento `ioError`. Se puede comprobar la propiedad `serverCertificateStatus` del objeto SecureSocket para determinar por qué se ha producido un error de validación. No existen previsiones para la comunicación con un servidor que no disponga de un certificado de confianza válido.

La clase CertificateStatus define constantes de cadena que representan los posibles resultados de validación:

- Caducado: ha pasado la fecha de caducidad del certificado.
- No válido: existen varios motivos por los que un certificado puede no ser válido. Por ejemplo, el certificado puede haber sido modificado, dañado o ser de tipo incorrecto.
- Cadena no válida: uno o varios certificados de la cadena del servidor no son válidos.
- Falta de coincidencia principal: el nombre de host del servidor y el nombre común del certificado no coinciden. Es decir, el servidor está utilizando el certificado incorrecto.
- Revocado: la entidad emisora de certificados ha revocado el certificado.
- De confianza: el certificado es válido y de confianza. Un objeto SecureSocket solo se puede conectar a un servidor que utilice un certificado válido y de confianza.
- Desconocido: el objeto SecureSocket no ha validado aún el certificado. La propiedad `serverCertificateStatus` presenta este valor de estado antes de llamar a `connect()` y antes de que se distribuya un evento `connect` o `ioError`.
- Firmantes que no son de confianza: el certificado no se “vincula” a un certificado raíz de confianza en el almacén de confianza del equipo cliente.

La comunicación con un objeto SecureSocket requiere que el servidor utilice un protocolo seguro y cuente con un certificado válido y de confianza. En otros aspectos, el uso de un objeto SecureSocket es el mismo que la utilización de un objeto Socket.

El objeto SecureSocket no se admite en todas las plataformas. Utilice la propiedad `isSupported` de la clase SecureSocket para comprobar si el motor de ejecución admite el uso del objeto SecureSocket en el equipo cliente actual.

Para obtener más información, consulte:

- SecureSocket
- CertificateStatus
- IDataInput
- IDataOutput
- Evento socketData

Ejemplo de socket TCP: Creación de un cliente Telnet

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el ejemplo de Telnet se muestran las técnicas para conectar con un servidor remoto y transmitir datos con la clase Socket. El ejemplo ilustra las técnicas siguientes:

- Creación de un cliente Telnet personalizado mediante la clase Socket
- Envío de texto al servidor remoto mediante un objeto ByteArray

- Gestión de datos recibidos de un servidor remoto

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Telnet se encuentran en la carpeta Samples/Telnet. La aplicación consta de los siguientes archivos:

Archivo	Descripción
TelnetSocket.fla o TelnetSocket.mxml	El archivo principal de la aplicación formado por la interfaz de usuario de Flex (MXML) o de Flash (FLA).
TelnetSocket.as	Clase de documento que proporciona la lógica de la interfaz de usuario (solo Flash).
com/example/programmingas3/Telnet/Telnet.as	Proporciona la funcionalidad del cliente Telnet para la aplicación como, por ejemplo, la conexión a un servidor remoto y el envío, recepción y visualización de datos.

Información general de la aplicación de socket Telnet

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El archivo principal TelnetSocket.mxml es responsable de crear la interfaz de usuario de toda la aplicación.

Además de la interfaz de usuario, este archivo también define dos métodos, `login()` y `sendCommand()`, para conectar el usuario al servidor especificado.

A continuación se muestra el código ActionScript del archivo de la aplicación principal:

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

La primera línea de código importa la clase Telnet del paquete com.example.programmingas.socket personalizado. La segunda línea de código declara una instancia de la clase Telnet, `telnetClient`, que se inicializará posteriormente mediante el método `connect()`. A continuación, se declara el método `connect()` e inicializa la variable `telnetClient` declarada previamente. Este método pasa el nombre del servidor Telnet especificado por el usuario, el puerto del servidor Telnet y una referencia a un componente TextArea de la lista de visualización, que se utiliza para mostrar las respuestas de texto del servidor de socket. Las dos últimas líneas del método `connect()` establecen la propiedad `title` de Panel y activan el componente Panel, que permite al usuario enviar datos al servidor remoto. El método final del archivo principal de la aplicación, `sendCommand()`, se utiliza para enviar los comandos del usuario al servidor remoto como un objeto ByteArray.

Información general de la clase Telnet

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Telnet es la responsable de conectar con el servidor Telnet remoto, y de enviar y recibir datos.

La clase Telnet declara las siguientes variables privadas:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

La primera variable, `serverURL`, contiene la dirección del servidor, especificada por el usuario, con la que se va a conectar.

La segunda variable, `portNumber`, es el número de puerto en el que se está ejecutando el servidor Telnet. De forma predeterminada, el servicio Telnet se ejecuta en el puerto 23.

La tercera variable, `socket`, es una instancia de Socket que intentará conectar con el servidor definido por las variables `serverURL` y `portNumber`.

La cuarta variable, `ta`, es una referencia a una instancia del componente TextArea en el escenario. Este componente se utiliza para mostrar las respuestas del servidor Telnet remoto o cualquier posible mensaje de error.

La última variable, `state`, es un valor numérico que se utiliza para determinar las opciones que admite el cliente Telnet.

Tal y como se ha visto previamente, se llama a la función constructora de la clase Telnet a través del método `connect()` en el archivo de la aplicación principal.

El constructor Telnet adopta tres parámetros: `server`, `port` y `output`. Los parámetros `server` y `port` especifican el nombre de servidor y el número de puerto donde se ejecuta el servidor Telnet. El parámetro final, `output`, es una referencia a una instancia del componente TextArea en el escenario, donde los usuarios verán la salida del servidor.

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

Escritura de datos en un socket

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para escribir datos en una conexión de socket, es necesario llamar a cualquiera de los métodos de la clase Socket. Entre estos métodos de escritura se incluyen `writeBoolean()`, `writeByte()`, `writeBytes()` y `writeDouble()`, entre otros. Posteriormente, vacíe los datos en el búfer de salida con el método `flush()`. En el servidor Telnet, los datos se escriben en la conexión de socket mediante el método `writeBytes()`, que utiliza el conjunto de bytes como parámetro y lo envía al búfer de salida. La sintaxis del método `writeBytesToSocket()` es la siguiente:

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Este método se llama mediante el método `sendCommand()` del archivo de la aplicación principal.

Visualización de mensajes del servidor de socket

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se recibe un mensaje del servidor de socket o se produce un evento, se llama al método personalizado `msg()`. Este método añade una cadena a TextArea en el escenario y llama a un método personalizado `setScroll()`, que desplaza el componente TextArea hasta la parte más baja. La sintaxis del método `msg()` es la siguiente:

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Si el contenido del componente TextArea no se desplaza automáticamente, el usuario deberá arrastrar manualmente las barras de desplazamiento en el área de texto para ver la última respuesta del servidor.

Desplazamiento de un componente TextArea

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `setScroll()` contiene una sola línea de código ActionScript que desplaza verticalmente el contenido del componente TextArea para que el usuario pueda ver la última línea del texto devuelto. El siguiente fragmento de código muestra el método `setScroll()`:

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Este método establece la propiedad `verticalScrollPosition`, que es el número de línea de la fila superior de caracteres que se visualiza, con el valor de la propiedad `maxVerticalScrollPosition`.

Sockets XML

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un socket XML permite crear una conexión en un servidor remoto que permanece abierta hasta que se cierra explícitamente. Puede intercambiar datos de cadenas, como XML, entre el servidor y el cliente. Una ventaja de utilizar un servidor de socket XML es que el usuario no tiene que solicitar datos de forma explícita. El servidor envía datos sin esperar una solicitud, y puede enviar datos a cualquier cliente conectado.

En Flash Player y en contenido de Adobe AIR fuera del entorno limitado de la aplicación, las conexiones de socket XML requieren la presencia de un archivo de política de socket en el servidor de destino. Para obtener más información, consulte "Controles de sitio web (archivos de política)" en la página 1095 y "Conexión a sockets" en la página 1114.

La clase XMLSocket no puede atravesar cortafuegos automáticamente, ya que, a diferencia del protocolo RTMP (Real-Time Messaging Protocol), XMLSocket no dispone de prestaciones de tunelación HTTP. Si es necesario utilizar tunelación HTTP, debe considerarse la posibilidad de utilizar Flash Remoting o Flash Media Server (que admite RTMP).

Las siguientes restricciones afectan a la forma y el lugar en que el contenido de Flash Player o de una aplicación de AIR puede utilizar un objeto XMLSocket para conectarse con el servidor fuera del entorno limitado de seguridad de la aplicación:

- Para el contenido fuera del entorno limitado de seguridad de la aplicación, el método `XMLSocket.connect()` solo puede conectarse a números de puerto TCP superiores o iguales a 1024. Como consecuencia de esta limitación, los dominios del servidor que se comunica con el objeto `XMLSocket` también deben asignarse a números de puerto superiores o iguales a 1024. Los números de puerto inferiores a 1024 suelen utilizarse para servicios del sistema como FTP (21), Telnet (23), SMTP (25), HTTP (80) y POP3 (110), por lo que los objetos XMLSocket no pueden utilizar estos puertos por razones de seguridad. La restricción de número de puerto limita la posibilidad de que se pueda acceder y hacer un uso indebido de estos recursos.
- Para el contenido fuera del entorno limitado de seguridad de la aplicación, el método `XMLSocket.connect()` solo puede conectarse a equipos del mismo dominio en que reside el contenido. (Esta restricción es idéntica a la incluida en las reglas de seguridad establecidas para `URLLoader.load()`.) Para conectar con un dominio de servidor que se ejecuta en un dominio diferente a aquel en el que reside el contenido, puede crear un archivo de política de seguridad de varios dominios que permita el acceso desde dominios específicos. Para más información sobre archivos de política de varios dominios, consulte "Seguridad en AIR" en la página 1122.

Nota: *la configuración de un servidor para que se comunique con el objeto XMLSocket puede resultar compleja. Si la aplicación no requiere interactividad en tiempo real, utilice la clase URLLoader en vez de la clase XMLSocket.*

Puede utilizar los métodos `XMLSocket.connect()` y `XMLSocket.send()` de la clase XMLSocket para transferir XML desde y hacia un servidor a través de una conexión de socket. El método `XMLSocket.connect()` establece una conexión de socket con un puerto de servidor web. El método `XMLSocket.end()` pasa un objeto XML al servidor especificado en la conexión de socket.

Cuando se invoca el método `XMLSocket.connect()`, la aplicación 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 `XMLSocket.close()` de la clase XMLSocket.
- No existen más referencias al objeto XMLSocket.
- Se sale de Flash Player.
- Se interrumpe la conexión (por ejemplo, se desconecta el módem).

Conexión con un servidor con la clase XMLSocket

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para crear una conexión de socket debe crear una aplicación de servidor que espere la petición de conexión de socket y envíe una respuesta a Flash Player o a la aplicación de AIR. Este tipo de aplicación de servidor puede escribirse en AIR o en un lenguaje de programación como Java, Python o Perl. Para utilizar la clase XMLSocket, el equipo servidor debe ejecutar un dominio que entienda el protocolo simple utilizado por la clase XMLSocket:

- Los mensajes XML se envían a través de una conexión de socket ininterrumpida TCP/IP dúplex.
- Cada mensaje XML es un documento XML completo terminado en un byte cero (0).
- Pueden enviarse y recibirse un número ilimitado de mensajes XML a través de una misma conexión XMLSocket.

Creación y conexión de un servidor de socket XML de Java

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El siguiente código muestra un sencillo servidor XMLSocket escrito en Java que acepta conexiones entrantes y muestra los mensajes recibidos en la ventana de símbolo del sistema. De forma predeterminada, se crea un nuevo servidor en el puerto 8080 del equipo local, aunque se puede especificar otro número de puerto cuando se inicia el servidor desde la línea de comandos.

Cree un nuevo documento de texto y añada el siguiente código:

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
```

```
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
                    done = true;
                }
                else
                {
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

Guarde el documento en el disco duro como SimpleServer.java y compílelo con la ayuda de un compilador de Java para crear un archivo de clase de Java denominado SimpleServer.class.

Para iniciar el servidor XMLSocket, abra un símbolo del sistema y escriba `java SimpleServer`. El archivo SimpleServer.class puede almacenarse en cualquier lugar del equipo local o la red; no es necesario que esté en el directorio raíz del servidor web.

Si no puede iniciar el servidor porque los archivos no se encuentran en la ruta de clases de Java, intente iniciar el servidor con `java -classpath. SimpleServer`.

Para conectarse a XMLSocket desde la aplicación , es necesario crear una nueva instancia de la clase XMLSocket y llamar al método `XMLSocket.connect()`, pasando un nombre de host y un número de puerto, del siguiente modo:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Cuando se reciben datos del servidor, se distribuye el evento `data` (`flash.events.DataEvent.DATA`):

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Para enviar datos al servidor XMLSocket, se utiliza el método `XMLSocket.send()` y se pasa una cadena u objeto XML. Flash Player convierte el parámetro suministrado en un objeto String y envía el contenido al servidor XMLSocket, seguido de un byte cero (0):

```
xmlsock.send(xmlFormattedData);
```

El método `XMLSocket.send()` no devuelve ningún valor que indique si los datos se han transmitido correctamente. Si se produce un error al intentar enviar datos, se emite un error IOError.

Cada mensaje que se envía al servidor de socket XML debe terminar con un carácter de nueva línea (`\n`).

Para obtener más información, consulte XMLSocket.

Sockets de servidor

Adobe AIR 2 y posterior

Utilice la clase ServerSocket para permitir que otro proceso se conecte a la aplicación utilizando un socket TCP (Transport Control Protocol). El proceso de conexión se puede ejecutar en el equipo local o en otro equipo conectado a la red. Cuando un objeto recibe una solicitud de conexión, distribuye un evento `connect`. El objeto ServerSocketConnectEvent distribuido con el evento contiene un objeto Socket. Este objeto Socket se puede emplear para una comunicación posterior con los demás procesos.

Para detectar conexiones de socket entrantes:

1. Cree un objeto ServerSocket y vincúlelo con un puerto local.
2. Añada detectores de eventos para el evento `connect`.
3. Llamada al método `listen()`
4. Responda al evento `connect`, lo que proporciona un objeto Socket para cada conexión entrante.

El objeto ServerSocket sigue detectando conexiones adicionales hasta que se llama al método `close()`.

En el siguiente ejemplo de código se muestra cómo crear una aplicación de servidor de socket. En el ejemplo se detectan las conexiones entrantes en el puerto 8087. Cuando se recibe una conexión, el ejemplo envía un mensaje (cadena “Connected.”) al socket de cliente. Por lo tanto, el servidor reproduce todos los mensajes recibidos del cliente.

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.ServerSocketConnectEvent;
    import flash.net.ServerSocket;
    import flash.net.Socket;

    public class ServerSocketExample extends Sprite
    {
        private var serverSocket:ServerSocket;
        private var clientSockets:Array = new Array();

        public function ServerSocketExample()
        {
            try
            {
                // Create the server socket
                serverSocket = new ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );

                // Bind to local port 8087
                serverSocket.bind( 8087, "127.0.0.1" );

                // Listen for connections
                serverSocket.listen();
                trace( "Listening on " + serverSocket.localPort );

            }
            catch(e:SecurityError)
            {
                trace(e);
            }
        }

        public function connectHandler(event:ServerSocketConnectEvent):void
        {
            //The socket is provided by the event object
            var socket:Socket = event.socket as Socket;
            clientSockets.push( socket );

            socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
            socket.addEventListener( Event.CLOSE, onClientClose );
            socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );

            //Send a connect message
            socket.writeUTFBytes("Connected.");
            socket.flush();

            trace( "Sending connect message" );
        }
```

```
        public function socketDataHandler(event:ProgressEvent):void
        {
            var socket:Socket = event.target as Socket

            //Read the message from the socket
            var message:String = socket.readUTFBytes( socket.bytesAvailable );
            trace( "Received: " + message);

            // Echo the received message back to the sender
            message = "Echo -- " + message;
            socket.writeUTFBytes( message );
            socket.flush();
            trace( "Sending: " + message );
        }

        private function onClientClose( event:Event ):void
        {
            trace( "Connection to client closed." );
            //Should also remove from clientSockets array...
        }

        private function onIOError( errorEvent:IOErrorEvent ):void
        {
            trace( "IOError: " + errorEvent.text );
        }

        private function onClose( event:Event ):void
        {
            trace( "Server socket closed by OS." );
        }
}}
```

Para obtener más información, consulte:

- ServerSocket
- ServerSocketConnectEvent
- Socket

Sockets UDP (AIR)

Adobe AIR 2 y posterior

El protocolo UDP (Universal Datagram Protocol) proporciona un modo de intercambiar mensajes a través de una conexión de red sin estado. UDP no garantiza que los mensajes se distribuyan en orden o incluso que los mensajes se envíen en modo alguno. Con UDP, el código de red del sistema operativo suele emplear menos tiempo en la identificación, seguimiento y reconocimiento de mensajes. De este modo, los mensajes UDP suelen llegar a la aplicación de destino con un retraso menor que los mensajes TCP.

La comunicación del socket UDP resulta útil cuando se debe enviar información en tiempo real como, por ejemplo, actualizaciones de posición en un juego o paquetes de sonido en una aplicación de chat de audio. En estas aplicaciones, se tolera cierta pérdida de datos y una latencia de transmisión baja es más importante que la llegada garantizada. Para la mayoría de los demás objetivos, los sockets TCP representan un mejor opción.

La aplicación de AIR puede enviar y recibir mensajes UDP con las clases DatagramSocket y DatagramSocketDataEvent. Para enviar o recibir un mensaje UDP:

1. Cree un objeto DatagramSocket.
2. Añada un detector de eventos al evento `data`.
3. Enlace el socket a un puerto y dirección IP local utilizando el método `bind()`.
4. Envíe mensajes llamando al método `send()` y transmitiendo el puerto y la dirección IP del equipo de destino.
5. Reciba mensajes respondiendo al evento `data`. El objeto DatagramSocketDataEvent distribuido para este evento contiene un objeto ByteArray que incluye los datos del mensaje.

El siguiente ejemplo de código muestra el modo en que una aplicación puede enviar y recibir mensajes UDP. El ejemplo envía un solo mensaje que contiene la cadena, “Hello.”, al equipo de destino. También realiza un seguimiento del contenido de todos los mensajes recibidos.

```
package
{
import flash.display.Sprite;
import flash.events.DatagramSocketDataEvent;
import flash.events.Event;
import flash.net.DatagramSocket;
import flash.utils.ByteArray;

public class DatagramSocketExample extends Sprite
{
    private var datagramSocket:DatagramSocket;

    //The IP and port for this computer
    private var localIP:String = "192.168.0.1";
    private var localPort:int = 55555;

    //The IP and port for the target computer
    private var targetIP:String = "192.168.0.2";
    private var targetPort:int = 55555;

    public function DatagramSocketExample()
    {
        //Create the socket
        datagramSocket = new DatagramSocket();
        datagramSocket.addEventListener( DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
```

```
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();

        //Create a message in a ByteArray
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    private function dataReceived( event:DatagramSocketDataEvent ):void
    {
        //Read the data from the datagram
        trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
}}
```

Al emplear sockets UDP se debe tener en cuenta lo siguiente:

- Un solo paquete de datos no puede ser superior a la unidad de transmisión máxima más pequeña (MTU) de la interfaz de red o cualquier nodo de red entre el remitente y el destinatario. Todos los datos del objeto ByteArray transmitidos al método send() se envían como un solo paquete. (En TCP, los mensajes grandes se dividen en paquetes independientes.)
- No existe conexión entre el remitente y el destino. Los mensajes se descartan sin error si el destino no existe o no dispone de un detector activo en el puerto especificado.
- Si se utiliza el método `connect()`, los mensajes enviados desde otros recursos se omitirán. Una conexión UDP solo proporciona el filtrado de paquetes adecuado. No significa que sea necesariamente un proceso de detección válido en la dirección y el puerto de destino.
- El tráfico UDP puede sobrecargar una red. Es posible que los administradores de red deban implementar controles de calidad del servicio si se produce la saturación en la red. (TCP dispone de control de tráfico integrado para reducir el impacto de la saturación de la red.)

Para obtener más información, consulte:

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

Direcciones IPv6

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player 9.0.115.0 y las versiones posteriores admiten IPv6 (Protocolo de Internet versión 6). IPv6 es una versión del Protocolo de Internet que admite direcciones de 128 bits (una mejora con respecto al protocolo anterior IPv4 que admite direcciones de 32 bits). Es posible que tenga que activar IPv6 en sus interfaces de red. Para más información, consulte la Ayuda del sistema operativo que aloja los datos.

Si se admite IPv6 en el sistema de host, puede especificar direcciones literales numéricas de IPv6 en URL entre corchetes ([]), como se muestra a continuación:

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Flash Player devuelve valores IPv6 literales, en función de las siguiente reglas:

- Flash Player devuelve el formulario largo de la cadena para las direcciones IPv6.
- El valor IP no tiene abreviaciones de dos puntos dobles.
- Los dígitos hexadecimales solo aparecen en minúscula.
- Las direcciones IPv6 aparecen entre corchetes ([]).
- Cada cuarteto de direcciones se representa como dígitos hexadecimales de 0 a 4, donde se omiten los primeros ceros.
- Un cuarteto de direcciones de todos los ceros aparece como un solo cero (no dos puntos dobles), excepto donde se indica en la siguiente lista de excepciones.

Los valores IPv6 que devuelve Flash Player presentan las siguientes excepciones:

- Una dirección IPv6 no especificada (todos los ceros) aparece como [::].
- La dirección IPv6 de localhost o bucle invertido se presenta como [::1].
- Las direcciones asignadas a IPv4 (convertidas a IPv6) se presentan como [::ffff:a.b.c.d], siendo a.b.c.d un valor típico decimal con puntos de IPv4.
- Las direcciones compatibles con IPv4 aparecen como [::a.b.c.d], siendo a.b.c.d un valor típico decimal con puntos de IPv4.

Capítulo 44: Comunicaciones HTTP

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las aplicaciones de Adobe® AIR® y Adobe® Flash® Player se pueden comunicar con servidores basados en HTTP para cargar datos, imágenes, vídeo y para intercambiar mensajes.

Más temas de ayuda

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

Carga de datos externos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 incluye mecanismos para cargar datos desde fuentes externas. Estas fuentes pueden ser contenido estático, como archivos de texto, o contenido dinámico generado por un script web. Se puede aplicar formato a los datos de varias maneras y ActionScript proporciona funcionalidad para descodificar y acceder a los datos. También se pueden enviar datos al servidor externo como parte del proceso de recuperación de datos.

Uso de la clase URLRequest

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Muchas API que cargan datos externos utilizan la clase URLRequest para definir las propiedades de la petición de red necesaria.

Propiedades de URLRequest

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede utilizar las siguientes propiedades de un objeto URLRequest en cualquier entorno limitado de seguridad:

Propiedad	Descripción
contentType	El tipo de contenido MIME de todos los datos enviados con la petición de URL. Si no se establece contentType, los valores se envían como `application/x-www-form-urlencoded`.
data	Un objeto que contiene datos que se van a transmitir con la petición de URL.
digest	Una cadena que identifica de forma unívoca el componente firmado de la plataforma de Adobe que se va a almacenar (o recuperar) en la caché de Adobe® Flash® Player.
method	El método de petición HTTP, por ejemplo, GET o POST. (El contenido ejecutado en el dominio de seguridad de la aplicación de AIR puede especificar cadenas que no sean `"GET"` o `"POST"` como propiedad `method`. Se permite cualquier palabra clave HTTP, aunque `"GET"` es el método predeterminado. Consulte "Seguridad en AIR" en la página 1122.
requestHeaders	El conjunto de encabezados de petición HTTP que se añadirán a la petición HTTP. Tenga en cuenta que el permiso para definir determinados encabezados en Flash Player y en contenido de AIR ejecutado fuera del entorno limitado de seguridad de la aplicación está restringido.
url	Especifica la dirección URL que se va a solicitar.

En AIR, puede definir propiedades adicionales de la clase URLRequest que solo están disponibles en contenido de AIR ejecutado fuera del entorno limitado de seguridad de la aplicación. El contenido del entorno limitado de la aplicación también puede definir las direcciones URL mediante los nuevos esquemas de URL (además de los esquemas estándar como `file` y `http`).

Propiedad	Descripción
followRedirects	Especifica si se siguen las redirecciones (`true`, valor predeterminado) o no (`false`). Esto solo se admite en el entorno limitado de la aplicación AIR.
manageCookies	Especifica si la cola del protocolo HTTP debe gestionar las cookies (`true`, valor predeterminado) o no (`false`) en esta petición. La configuración de esta propiedad solo se admite en el entorno limitado de la aplicación de AIR.
authenticate	Especifica si deben manejarse peticiones de autenticación (`true`) en esta petición. La configuración de esta propiedad solo se admite en el entorno limitado de la aplicación de AIR. El comportamiento predeterminado es autenticar peticiones. Esto puede provocar la aparición de un cuadro de diálogo si el servidor requiere credenciales. También puede establecer el nombre de usuario y la contraseña con la clase URLRequestDefaults; consulte "Configuración de los valores predeterminados de URLRequest (solo AIR)" en la página 846.
cacheResponse	Especifica si los datos de respuesta de esta petición se deben guardar en la memoria caché. La configuración de esta propiedad solo se admite en el entorno limitado de la aplicación de AIR. El valor predeterminado es guardar la respuesta en la memoria caché (`true`).
useCache	Especifica si se debe consultar la memoria caché local antes de que el objeto URLRequest tome los datos. La configuración de esta propiedad solo se admite en el entorno limitado de la aplicación de AIR. El comportamiento predeterminado (`true`) es utilizar la versión guardada en la memoria local, si está disponible.
userAgent	Especifica la cadena de agente de usuario que se utiliza en la petición HTTP.

Nota: *la clase HTMLLoader tiene propiedades relacionadas para ajustes que pertenecen al contenido cargado mediante un objeto HTMLLoader. Para obtener más información, consulte "Información sobre la clase HTMLLoader" en la página 1019.*

Configuración de los valores predeterminados de URLRequest (solo AIR)

Adobe AIR 1.0 y posterior

La clase URLRequestDefaults permite definir la configuración predeterminada específica de la aplicación de los objetos URLRequest. Por ejemplo, el siguiente código establece los valores predeterminados de las propiedades `manageCookies` y `useCache`. Todos los nuevos objetos URLRequest utilizarán los valores especificados para estas propiedades y no los valores predeterminados:

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
```

Nota: *la clase URLRequestDefaults se define para el contenido que se ejecuta únicamente en Adobe AIR. No se admite en Flash Player.*

La clase URLRequestDefaults contiene un método `setLoginCredentialsForHost()` que permite especificar un nombre de usuario y una contraseña predeterminados para usarlos con un host específico. El host, definido en el parámetro hostname del método, puede ser un dominio (por ejemplo `"www.example.com"`) o un dominio y un número de puerto (por ejemplo `"www.example.com:80"`. Observe que `"example.com"`, `"www.example.com"`, y `"sales.example.com"` se consideran hosts únicos.

Estas credenciales solo se utilizan si las requiere el servidor. Si el usuario ya se ha autenticado (por ejemplo, desde el cuadro de diálogo de autenticación), llamar al método `setLoginCredentialsForHost()` no cambiará al usuario autenticado.

El siguiente código establece el nombre de usuario y contraseña predeterminados para las peticiones enviadas a www.example.com:

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

La configuración de URLRequestDefaults solo se aplica al dominio de la aplicación actual con una excepción. Las credenciales transferidas al método `setLoginCredentialsForHost()` se utilizan para peticiones realizadas en cualquier dominio de aplicación dentro de la aplicación de AIR.

Para obtener más información, consulte la clase URLRequestDefaults en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Esquemas de URI

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los esquemas de URI estándar, como el siguiente, se pueden utilizar en peticiones realizadas desde cualquier entorno limitado de seguridad:

http: y https:

Utilice este esquema para URL de Internet estándar (del mismo modo que se utilizan en un navegador web).

file:

Utilice `file:` para especificar la URL de un archivo ubicado en el sistema de archivos local. Por ejemplo:

```
file:///c:/AIR Test/test.txt
```

En AIR, también se pueden utilizar los siguientes esquemas al definir una dirección URL para el contenido que se ejecuta en el entorno limitado de seguridad de la aplicación:

app:
Utilice `app:` para especificar una ruta relativa al directorio raíz de la aplicación instalada. Por ejemplo, la siguiente ruta apunta a un subdirectorio de recursos del directorio de la aplicación instalada:

```
app:/resources
```

Cuando se inicia una aplicación de AIR con AIR Debug Launcher (ADL), el directorio de la aplicación es el directorio que contiene el archivo descriptor de la aplicación.

La dirección URL (y la propiedad `url`) de un objeto File creado con `File.applicationDirectory` utiliza el esquema de URI `app`, tal como se puede ver a continuación:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("assets");
trace(dir.url); // app:/assets
```

app-storage:
Use `app-storage:` para especificar una ruta relativa al directorio de almacenamiento de datos de la aplicación. AIR crea un directorio de almacenamiento exclusivo para cada una de las aplicaciones instaladas (y usuarios), lo cual proporciona un lugar útil para guardar datos que son específicos para esa aplicación. Por ejemplo, la siguiente ruta apunta al archivo "prefs.xml" en el subdirectorio "settings" del directorio de almacenamiento de la aplicación:

```
app-storage:/settings/prefs.xml
```

La dirección URL (y la propiedad `url`) de un objeto File creado con `File.applicationStorageDirectory` utiliza el esquema de URI `app-storage`, tal como se indica a continuación:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
trace(dir.prefsFile); // app-storage:/prefs.xml
```

mailto:
Puede utilizar el esquema mailto en objetos URLRequest transmitidos a la función`navigateToURL()`. Consulte "Apertura de una URL en otra aplicación" en la página 861.

Puede elegir un objeto URLRequest que utilice cualquiera de estos esquemas de URI para definir la petición de URL para varios objetos, como FileStream o Sound. También puede utilizar estos esquemas en contenido HTML que se ejecute en AIR; por ejemplo, es posible utilizarlos en el atributo `src` de una etiqueta `img`.

No obstante, solo se pueden utilizar estos esquemas de URI específicos de AIR (`app:` y `app-storage:`) en contenido del entorno limitado de seguridad de la aplicación. Para obtener más información, consulte "Seguridad en AIR" en la página 1122.

Configuración de variables URL

Aunque puede añadir variables a la cadena URL directamente, es más sencillo utilizar la clase URLVariables para definir cualquier variable necesaria para la petición.

Hay tres formas de añadir parámetros a un objeto URLVariables:

- En el constructor URLVariables
- Con el método `URLVariables.decode()`
- Como propiedades dinámicas del propio objeto URLVariables

El siguiente ejemplo ilustra los tres métodos y cómo asignar las variables a un objeto URLRequest:

```
var urlVar:URLVariables = new URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest:URLRequest = new URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

Cuando se definen variables en el constructor URLVariables o en el método `URLVariables.decode()`, es importante codificar en la URL los caracteres que puedan tener un significado especial en una cadena URI. Por ejemplo, si utiliza un ampersand en un nombre de parámetro o valor, debe codificarlo cambiando su forma `&` por `%26` para que el ampersand haga de delimitador en los parámetros. La función `encodeURIComponent()` de nivel superior se puede utilizar para este fin.

Uso de la clase URLLoader

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase URLLoader permite enviar una petición a un servidor y acceder a la información devuelta. También puede utilizar la clase URLLoader para acceder a archivos del sistema de archivos local en contextos en los que el acceso está permitido (por ejemplo, en el entorno limitado con sistema de archivos local de Flash Player y el entorno limitado de la aplicación de AIR). La clase URLLoader descarga datos desde una URL como texto, datos binarios o variables con codificación URL. La clase URLLoader distribuye eventos como, por ejemplo, `complete`, `httpStatus`, `ioError`, `open`, `progress` y `securityError`.

El modelo de gestión de eventos de ActionScript 3.0 difiere bastante del modelo de ActionScript 2.0, que utilizaba los controladores de eventos `LoadVars.onData`, `LoadVars.onHTTPStatus` y `LoadVars.onLoad`. Para obtener más información sobre la gestión de eventos en ActionScript 3.0, consulte "Gestión de eventos" en la página 127

Los datos descargados no están disponibles hasta que finaliza la descarga. Es posible supervisar el progreso de la descarga (bytes cargados y bytes totales) detectando el evento `progress` que se va a distribuir. No obstante, en ocasiones los archivos se cargan tan rápido que no se llega a distribuir ningún evento `progress`. Cuando el archivo se ha descargado correctamente, se distribuye el evento `complete`. Al establecer la propiedad `dataFormat` de URLLoader, puede recibir datos como texto, como datos binarios sin procesar o como un objeto URLVariables.

El método `URLLoader.load()` (y opcionalmente el constructor de la clase URLLoader) admite un solo parámetro, `request`, que es un objeto URLRequest. Un objeto URLRequest contiene toda la información de una sola petición HTTP, como la URL de destino, el método de petición (`GET` o `POST`), información de encabezado adicional y el tipo MIME (por ejemplo, cuando se carga contenido XML.

Por ejemplo, para cargar un paquete XML en un script de servidor, se podría usar el siguiente código

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <clock>
        <time>{secondsUTC}</time>
    </clock>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
loader.load(request);
```

El fragmento de código anterior crea un documento XML denominado `dataXML` que contiene el paquete XML que se enviará al servidor. El ejemplo establece la propiedad `contentType` de URLRequest como `"text/xml"` y asigna el documento XML a la propiedad `data` de URLRequest. Finalmente, este ejemplo crea un objeto URLLoader y envía la petición al script remoto mediante el método `load()`.

Uso de la clase URLStream

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase URLStream proporciona acceso a los datos de descarga a medida que se reciben. La clase URLStream también permite cerrar una transmisión antes de que finalice su descarga. Los datos descargados están disponibles como datos binarios sin procesar.

Cuando lea los datos desde un objeto URLStream, utilice la propiedad `bytesAvailable` para determinar si hay suficientes datos disponibles antes de leerlos. Se emite un excepción EOFError si se intenta leer más datos de los disponibles.

El evento httpResponseStatus (AIR)

En Adobe AIR, la clase URLStream distribuye un evento `httpResponseStatus` además del evento `httpStatus`. El evento `httpResponseStatus` se distribuye antes de recibid cualquier dato de respuesta. El evento `httpResponseStatus` (representado por la clase HTTPStatusEvent) incluye una propiedad `responseURL` (que es la dirección URL desde la que se devolvió la respuesta) y una propiedad `responseHeaders` (que es un conjunto de objetos URLRequestHeader que representan los encabezados de respuesta devueltos).

Carga de datos desde documentos externos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se crean aplicaciones dinámicas, puede ser útil cargar datos de archivos externos o scripts de servidor. De esta forma es posible generar aplicaciones dinámicas sin tener que editar o recompilar los archivos de ActionScript. Por ejemplo, si se genera una aplicación de "sugerencia del día", se puede escribir un script de servidor que recupere una sugerencia aleatoria de una base de datos y la guarde en un archivo de texto una vez al día. Luego la aplicación puede cargar el contenido de un archivo de texto estático en lugar de consultar la base de datos cada vez.

El siguiente fragmento de código crea un objeto URLRequest y URLLoader, que carga el contenido de un archivo de texto externo, params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

De forma predeterminada, si no se define un método de petición, Flash Player y Adobe AIR cargan el contenido con el método HTTP `GET`. Si se desea enviar los datos con el método `POST`, es necesario establecer la propiedad `request.method` en `POST` con la constante estática `URLRequestMethod.POST`, como se muestra en el siguiente código:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

El documento externo (params.txt) que se carga en tiempo de ejecución contiene los siguientes datos:

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

El archivo contiene dos parámetros, `monthNames` y `dayNames`. Cada parámetro incluye una lista separada por comas que se analiza como cadenas. Se puede dividir esta lista en un conjunto con el método `String.split()`.

Conviene evitar la utilización de palabras reservadas y construcciones del lenguaje como nombres de variables en archivos de datos externos, ya que dificulta la lectura y depuración del código.

Cuando se han cargado los datos, se distribuye el evento `Event.COMPLETE` y el contenido del documento externo está disponible para utilizarlo en la propiedad `data` de URLLoader, como se muestra en el siguiente código:

```
function completeHandler(event)
{
    var loader2 = event.target;
    air.trace(loader2.data);
}
```

Si el documento remoto contiene pares nombre-valor, puede analizar los datos con la clase URLVariables analizando el contenido del archivo cargado del modo siguiente:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Cada par nombre-valor del archivo externo se crea como una propiedad en el objeto URLVariables. Cada propiedad del objeto variables del ejemplo de código anterior se trata como una cadena. Si el valor del par nombre-valor es una lista de elementos, puede convertir la cadena en un conjunto llamando al método `String.split()`, como se indica a continuación:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

Si se cargan datos numéricos de archivos de texto externos, es necesario convertir los valores en valores numéricos, mediante una función de nivel superior como `int()`, `uint()` o `Number()`.

En lugar de cargar el contenido del archivo remoto como una cadena y crear un nuevo objeto URLVariables, se puede establecer la propiedad `URLLoader.dataFormat` en una de las propiedades estáticas de la clase URLLoaderDataFormat. Los tres valores posibles para la propiedad `URLLoader.dataFormat` son los siguientes:

- `URLLoaderDataFormat.BINARY`: la propiedad `URLLoader.data` contendrá datos binarios almacenados en un objeto ByteArray.
- `URLLoaderDataFormat.TEXT`: la propiedad `URLLoader.data` contendrá texto en un objeto String.
- `URLLoaderDataFormat.VARIABLES`: la propiedad `URLLoader.data` contendrá variables con codificación URL almacenadas en un objeto URLVariables.

En el código siguiente se muestra que, al establecer la propiedad `URLLoader.dataFormat` en `URLLoaderDataFormat.VARIABLES`, se permite analizar automáticamente los datos cargados en un objeto URLVariables:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

Nota: *el valor predeterminado de* `URLLoader.dataFormat` *es* `URLLoaderDataFormat.TEXT`*.*

Como puede verse en el siguiente ejemplo, cargar XML desde un archivo externo es lo mismo que cargar URLVariables. Puede crear una instancia de URLRequest y una instancia de URLLoader, y utilizarlas para descargar un documento XML remoto. Cuando el archivo se ha descargado completamente, se distribuye el evento `Event.COMPLETE` y el contenido del archivo externo se convierte en una instancia de XML, que puede analizarse con los métodos y propiedades de XML.

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

Comunicación con scripts externos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además de cargar archivos de datos externos, se puede utilizar la clase URLVariables para enviar variables a un script de servidor y procesar la respuesta del servidor. Esto resulta útil, por ejemplo, si está programando un juego y quiere enviar la puntuación del jugador a un servidor para calcular si se deben añadir a la lista de máximas puntuaciones o incluso para enviar la información de conexión del usuario a un servidor para su validación. Un script de servidor puede procesar el nombre y la contraseña del usuario y devolver una confirmación para saber si las credenciales facilitadas por el usuario son válidas.

El siguiente fragmento de código crea un objeto URLVariables denominado `variables`, que crea una nueva variable denominada `name`. A continuación, se crea un objeto URLRequest que especifica la dirección URL del script del servidor a la que se deben enviar las variables. Después, establezca la propiedad `method` del objeto URLRequest para que envíe las variables como una petición HTTP `POST`. Para añadir el objeto URLVariables a la petición de URL, establezca la propiedad `data` del objeto URLRequest como el objeto URLVariables creado anteriormente. Para terminar, se crea la instancia de URLLoader y se invoca el método `URLLoader.load()`, iniciándose así la petición.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

El código siguiente incluye el contenido del documento greeting.cfm de Adobe ColdFusion® utilizado en el ejemplo anterior:

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Peticiones de servicios web

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen diversos servicios web basados en HTTP. Algunos de los tipos principales son:

- REST
- XML-RPC
- SOAP

Para utilizar un servicio web en ActionScript 3, debe crear un objeto URLRequest, construir la llamada al servicio web con variables URL o con un documento XML, y enviar la llamada al servicio mediante un objeto URLLoader. La arquitectura de Flex contiene varias clases que facilitan el uso de los servicios web, de utilidad especial cuando se accede a servicios SOAP complejos. Desde Flash Professional CS3, es posible utilizar las clases de Flex en aplicaciones desarrolladas con Flash Professional, además de en aplicaciones desarrolladas en Flash Builder.

En aplicaciones de AIR basadas en HTML, puede usar las clases URLRequest y URLLoader, o bien la clase XMLHttpRequest de JavaScript. Si lo desea, también puede crear una biblioteca SWF que contenga los componentes de servicios web de la arquitectura de Flex y los ponga a disposición del código JavaScript.

Cuando la aplicación se ejecuta en un navegador, solo es posible utilizar servicios web del mismo dominio de Internet que el archivo SWF que llama, a no ser que el servidor que aloja el servicio web también tenga un archivo de política entre dominios para permitir el acceso desde otros dominios. Una técnica que se suele utilizar cuando no hay archivo de política entre dominios disponible es pasar por el proxy las peticiones del propio servidor. Adobe Blaze DS y Adobe LiveCycle admiten proxy de servicios web.

En aplicaciones de AIR, no se requiere un archivo de política entre dominios si la llamada al servicio web se origina en el entorno limitado de seguridad de la aplicación. El contenido de la aplicación de AIR nunca se ofrece desde un dominio remoto, por lo que no puede participar en tipos de ataques prevenidos por políticas entre dominios. En aplicaciones de AIR basadas en HTML, el contenido del entorno limitado de seguridad de la aplicación puede realizar peticiones XMLHttpRequest entre dominios. Puede permitir que el contenido de otros entornos limitados de seguridad de la aplicación realicen peticiones XMLHttpRequest entre dominios siempre y cuando el contenido se cargue en un iframe.

Más temas de ayuda

"Controles de sitio web (archivos de política)" en la página 1095

Adobe BlazeDS

Adobe LiveCycle ES2

Arquitectura REST

XML-RPC

Protocolo SOAP

Peticiones de servicios web de estilo REST

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los servicios web de estilo REST utilizan métodos HTTP para designar la acción básica y variables URL para especificar los detalles de la acción. Por ejemplo, una petición para obtener datos de un elemento podría utilizar GET y variables URL para especificar el nombre del método y el ID del elemento. La cadena URL resultante podría ser algo así:

```
http://service.example.com/?method=getItem&id=d3452
```

Para acceder a un servicio web de tipo REST con ActionScript, puede utilizar las clases URLRequest, URLVariables y URLLoader. El código JavaScript dentro de una aplicación de AIR, también puede utilizar una petición XMLHttpRequest.

La programación de llamadas a servicios web de tipo REST desde ActionScript suele implicar los pasos siguientes:

1. Cree un objeto URLRequest.
2. Establezca el método HTTP y la URL del servicio en el objeto solicitado.
3. Cree un objeto URLVariables.
4. Establezca los parámetros de llamada al servicio como propiedades dinámicas del objeto variables.
5. Asigne el objeto variables a la propiedad data del objeto solicitado.
6. Envíe la llamada al servicio con un objeto URLLoader.
7. Gestione el evento `complete` distribuido por URLLoader que indica que la llamada al servicio ha finalizado. Nunca está de más detectar diversos eventos de error que puedan ser distribuidos por un objeto URLLoader.

Por ejemplo, considere un servicio web que exponga un método de prueba que devuelva los parámetros de la llamada al solicitante. El siguiente código ActionScript se puede utilizar para llamar al servicio:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;

private var requestor:URLLoader = new URLLoader();
public function restServiceCall():void
{
    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/" );
    request.method = URLRequestMethod.GET;

    //Add the URL variables
    var variables:URLVariables = new URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    requestor = new URLLoader();
    requestor.addEventListener( Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
private function httpRequestComplete( event:Event ):void
{
    trace( event.target.data );
}

private function httpRequestError( error:ErrorEvent ):void{
    trace( "An error occured: " + error.message );
}
```

En JavaScript, dentro de una aplicación de AIR, puede hacer la misma petición con el objeto XMLHttpRequest:

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

Peticiones de servicios web XML-RPC

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un servicio web XML-RPC toma sus parámetros de llamada como un documento XML y no como un conjunto de variables URL. Para llevar a cabo una transacción con un servicio web XML-RPC, cree correctamente un mensaje XML con formato y envíelo al servicio web con el método HTTP `POST`. Además, se debe establecer el encabezado `Content-Type` para la petición de modo que el servidor trate los datos solicitados como XML.

El siguiente ejemplo ilustra cómo utilizar la misma llamada al servicio web del ejemplo REST, pero esta vez como un servicio XML-RPC:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
public function xmlRPCRequest():void
{
    //Create the XML-RPC document
    var xmlRPC:XML = <methodCall>
                                    <methodName></methodName>
                                    <params>
                                        <param>
                                            <value>
                                                <struct/>
                                            </value>
                                        </param>
                                    </params>
                                </methodCall>;

    xmlRPC.methodName = "test.echo";

    //Add the method parameters
    var parameters:Object = new Object();
    parameters.api_key = "123456ABC";
    parameters.message = "Able was I, ere I saw Elba.";

    for( var propertyName:String in parameters )
    {
        xmlRPC..struct.member[xmlRPC..struct.member.length + 1] =
            <member>
                <name>{propertyName}</name>
                <value>
                    <string>{parameters[propertyName]}</string>
                </value>
            </member>;
    }

    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/xml-rpc/" );
    request.method = URLRequestMethod.POST;
    request.cacheResponse = false;
    request.requestHeaders.push(new URLRequestHeader("Content-Type", "application/xml"));
```

```
    request.data = xmlRPC;

    //Initiate the request
    requestor = new URLLoader();
    requestor.dataFormat = URLLoaderDataFormat.TEXT;
    requestor.addEventListener( Event.COMPLETE, xmlRPCRequestComplete );
    requestor.addEventListener( IOErrorEvent.IO_ERROR, xmlRPCRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, xmlRPCRequestError );
    requestor.load( request );
}

private function xmlRPCRequestComplete( event:Event ):void
{
    trace( XML(event.target.data).toXMLString() );
}

private function xmlRPCRequestError( error:ErrorEvent ):void
{
    trace( "An error occurred: " + error );
}
```

WebKit en AIR no admite la sintaxis E4X, por lo que el método utilizado para crear el documento XML del ejemplo anterior no funcionaría con código JavaScript. Deberían utilizarse métodos DOM para crear el documento XML o crearlo como una cadena y utilizar la clase DOMParser de JavaScript para convertir la cadena en XML.

El siguiente ejemplo utiliza métodos DOM para crear un mensaje XML-RPC y un objeto XMLHttpRequest para llevar a cabo la transacción del servicio web:

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );

    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
```

```
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );

            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
    }
    if( rootNode.childElementCount > 0 )
```

```
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

Peticiones de servicios web SOAP

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

SOAP se basa en el concepto general del servicio XML-RPC y ofrece un modo más completo y complejo para transferir datos. Los servicios web SOAP suelen proporcionar un archivo WSDL que especifica las llamadas, tipos de datos y URL de servicio del servicio web. Aunque ActionScript 3 no admite SOAP directamente, puede construir un mensaje XML de SOAP "a mano", publicarlo en el servidor y analizar los resultados. Sin embargo, a no ser que se trate del servicio web SOAP más sencillo del mundo, probablemente ahorrará mucho tiempo si utiliza la biblioteca SOAP existente.

La arquitectura de Flex incluye bibliotecas para poder acceder a servicios web SOAP. En Flash Builder, la biblioteca rpc.swc se incluye automáticamente en los proyectos de Flex, ya que forma parte de la arquitectura de Flex. En Flash Professional, puede añadir framework.swc y rpc.swc a la ruta de biblioteca de un proyecto y, después, podrá acceder a las clases de Flex en ActionScript.

Más temas de ayuda

Uso del componente de servicio web de Flex en Flash Professional

Cristophe Coenraets: Real-time Trader Desktop for Android

Apertura de una URL en otra aplicación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede utilizar la función `navigateToURL()` para abrir una dirección URL en una página web u otra aplicación. Para el contenido que se ejecuta en AIR, la función `navigateToURL()` abre la página en el navegador web del sistema predeterminado.

Para el objeto URLRequest que transfiere como parámetro `request` de esta función, solo se utiliza la propiedad `url`.

El primer parámetro de la función `navigateToURL()` (el parámetro `navigate`) es un objeto URLRequest (consulte "Uso de la clase URLRequest" en la página 844). El segundo parámetro, `window`, es opcional y sirve para especificar el nombre de la ventana. Por ejemplo, la página web de Adobe se abre de la manera siguiente:

```
var url:String = "http://www.adobe.com";
var urlReq:URLRequest = new URLRequest(url);
navigateToURL(urlReq);
```

Nota: *al utilizar la función `navigateToURL()`, el motor de tiempo de ejecución trata un objeto URLRequest que utiliza el método POST (un objeto con su propiedad `method` establecida como `URLRequestMethod.POST`) como si utilizara el método GET.*

Cuando se utiliza la función `navigateToURL()`, se permiten esquemas de URI basados en el entorno limitado de seguridad del código que llama a la función `navigateToURL()`.

Algunas de las siguientes API permiten lanzar contenido en un navegador web. Por motivos de seguridad, algunos esquemas de URI no están permitidos cuando se utilizan estas API en AIR. La lista de esquemas no permitidos depende del entorno limitado de seguridad del código que utilice la API. (Para obtener más información sobre los entornos limitados de seguridad, consulte "Seguridad en AIR" en la página 1122.)

Entorno limitado de la aplicación (solo AIR)

Cualquier esquema URI se puede utilizar en una dirección URL iniciada por contenido ejecutándose en el entorno de seguridad de la aplicación de AIR. Una aplicación debe estar registrada para poder gestionar el esquema URI, si no, la solicitud no servirá de nada. Muchos equipos y dispositivos admiten los siguientes esquemas:

- `http:`
- `https:`
- `file:`
- `mailto:` - AIR dirige estas solicitudes a la aplicación de correo registrada en el sistema
- `sms:` - AIR dirige solicitudes `sms:` a la aplicación de mensajes de texto predeterminada. El formato de URL se debe ajustar a las convenciones del sistema con las que se ejecuta la aplicación. Por ejemplo, en Android, el esquema de URI debe escribirse en minúsculas.

  ```
  navigateToURL( new URLRequest( "sms:+15555550101") );
  ```

- `tel:` - AIR dirige solicitudes `tel:` a la aplicación de marcado del teléfono predeterminada. El formato de URL se debe ajustar a las convenciones del sistema con las que se ejecuta la aplicación. Por ejemplo, en Android, el esquema de URI debe escribirse en minúsculas.

  ```
  navigateToURL( new URLRequest( "tel:5555555555") );
  ```

- `market:` - AIR dirige solicitudes `market:` a la aplicación Market que suele admitirse en dispositivos Android.

  ```
  navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
  navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
  ```

Siempre que lo permita el sistema operativo, las aplicaciones pueden definir y registrar esquemas URI personalizados. Puede crear una dirección URL con el esquema para iniciar la aplicación desde AIR.

Entornos limitados remotos

Se permiten los siguientes esquemas. Utilice estos esquemas del mismo modo que lo haría en un navegador web.

- `http:`
- `https:`
- `mailto:` AIR dirige estas peticiones a la aplicación de correo registrada en el sistema

El resto de esquemas de URI no están permitidos.

Entorno limitado de archivos locales del sistema de archivos

Se permiten los siguientes esquemas. Utilice estos esquemas del mismo modo que lo haría en un navegador web.

- `file:`
- `mailto:` AIR dirige estas peticiones a la aplicación de correo registrada en el sistema

El resto de esquemas de URI no están permitidos.

Entorno limitado de archivos locales de red

Se permiten los siguientes esquemas. Utilice estos esquemas del mismo modo que lo haría en un navegador web.

- `http:`
- `https:`
- `mailto:` AIR dirige estas peticiones a la aplicación de correo registrada en el sistema

El resto de esquemas de URI no están permitidos.

Entorno limitado de archivos de confianza

Se permiten los siguientes esquemas. Utilice estos esquemas del mismo modo que lo haría en un navegador web.

- `file:`
- `http:`
- https:
- `mailto:` AIR dirige estas peticiones a la aplicación de correo registrada en el sistema

El resto de esquemas de URI no están permitidos.

Capítulo 45: Conexión con otras instancias de Flash Player y AIR

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase LocalConnection permite la comunicación entre las aplicaciones de Adobe® AIR®, así como entre el contenido SWF que se ejecuta en el navegador. La clase LocalConnection también se puede utilizar para comunicarse entre una aplicación de AIR y el contenido SWF que se ejecuta en el navegador. La clase LocalConnection permite crear aplicaciones muy versátiles que pueden compartir datos entre instancias de Flash Player y AIR.

Información sobre la clase LocalConnection

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase LocalConnection permite desarrollar archivos SWF capaces de enviar instrucciones a otros archivos SWF sin utilizar el método fscommand() ni JavaScript. Los objetos LocalConnection solo pueden comunicarse con archivos SWF que se ejecuten en el mismo equipo cliente, aunque pueden ejecutarse en aplicaciones distintas. Por ejemplo, un archivo SWF que se ejecuta en un navegador y un archivo SWF que se ejecuta en un proyector pueden compartir información, y el proyector conserva la información local y el archivo basado en el navegador se conecta de forma remota. (Un proyector es un archivo SWF guardado en un formato que puede ejecutarse como una aplicación autónoma, es decir, que el proyector no requiere que se instale Flash Player porque está incorporado en el ejecutable.)

Los objetos LocalConnection se pueden utilizar para comunicarse entre archivos SWF con distintas versiones de ActionScript:

- Los objetos LocalConnection de ActionScript 3.0 pueden comunicarse con objetos LocalConnection creados con ActionScript 1.0 o 2.0.
- Los objetos LocalConnection de ActionScript 1.0 o 2.0 pueden comunicarse con objetos LocalConnection creados con ActionScript 3.0.

Flash Player controla automáticamente esta comunicación entre objetos LocalConnection de distintas versiones.

La manera más simple de utilizar un objeto LocalConnection es permitir la comunicación solo entre objetos LocalConnection ubicados en el mismo dominio o en la misma aplicación de AIR. De este modo no es necesario preocuparse por la seguridad. Sin embargo, si debe permitir la comunicación entre dominios, tiene varias maneras de implementar medidas de seguridad. Para obtener información, consulte la descripción del parámetro `connectionName` del método `send()` y las entradas `allowDomain()` y `domain` en el listado de clases LocalConnection en Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Es posible utilizar objetos LocalConnection para enviar y recibir datos en un solo archivo SWF, pero Adobe no recomienda esta práctica. Utilice objetos compartidos.

Hay tres formas de añadir métodos callback a los objetos LocalConnection:

- Crear una subclase de la clase LocalConnection y añadir métodos.
- Establecer la propiedad `LocalConnection.client` en un objeto que implemente los métodos.
- Crear una clase dinámica que amplíe LocalConnection y asociar métodos de forma dinámica.

La primera forma de añadir métodos callback consiste en ampliar la clase LocalConnection. Los métodos se definen en la clase personalizada en lugar de añadirlos de forma dinámica a la instancia de LocalConnection. Este enfoque se muestra en el siguiente código:

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Para crear una nueva instancia de la clase CustomLocalConnection, se puede usar el siguiente código:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

La segunda forma de añadir métodos callback consiste en utilizar la propiedad `LocalConnection.client`. Esto implica crear una clase personalizada y asignar una nueva instancia a la propiedad `client`, como se muestra en el siguiente código:

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

La propiedad `LocalConnection.client` indica los métodos callback del objeto que deben llamarse. En el código anterior, la propiedad `client` se estableció en una nueva instancia de una clase personalizada, CustomClient. El valor predeterminado de la propiedad `client` es la instancia de LocalConnection actual. Se puede utilizar la propiedad `client` si se tienen dos controladores de datos con el mismo conjunto de métodos pero distintos comportamientos; por ejemplo, en una aplicación donde un botón de una ventana activa o desactiva la visualización en una segunda ventana.

Para crear la clase CustomClient, se podría usar el siguiente código:

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

La tercera forma de añadir métodos callback, que consiste en crear una clase dinámica y asociar los métodos de forma dinámica, es muy similar a la utilización de la clase LocalConnection en versiones anteriores de ActionScript, como se muestra en el siguiente código:

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Los métodos callback pueden añadirse de forma dinámica a esta clase con el siguiente código:

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

No se recomienda la anterior forma de añadir métodos callback porque el código tiene una escasa portabilidad. Además, si se utiliza este método de creación de conexiones locales, podrían surgir problemas de rendimiento debido a que el acceso a las propiedades dinámicas es considerablemente más lento que el acceso a las propiedades cerradas.

Propiedad isPerUser

La propiedad `isPerUser` se añadió a Flash Player (10.0.32) y a AIR (1.5.2) para resolver un conflicto que se producía cuando más de un usuario se conectaba a un ordenador Mac. En otros sistemas operativos, esta propiedad se omite porque la conexión local siempre está destinada a usuarios individuales. La propiedad `isPerUser` debe establecerse como `true` en código nuevo. Sin embargo, el valor predeterminado es actualmente `false` para permitir compatibilidad con versiones anteriores. El valor predeterminado puede cambiar en versiones futuras del motor de ejecución.

Envío de mensajes entre dos aplicaciones

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se usa la clase LocalConnection para la comunicación entre aplicaciones de AIR y entre aplicaciones de AIR y aplicaciones de Adobe® Flash® Player (SWF) que se ejecutan en un navegador. La clase LocalConnection también se puede utilizar para comunicarse entre una aplicación de AIR y una aplicación SWF que se ejecuta en un navegador.

Por ejemplo, se pueden tener varias instancias de Flash Player en una página web o se puede hacer que una instancia de Flash Player recupere datos de otra instancia de Flash Player en una ventana emergente.

El siguiente código define un objeto LocalConnection que actúa como un servidor y acepta llamadas entrantes LocalConnection de otras aplicaciones:

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

Este código primero crea un objeto LocalConnection denominado `lc` y define la propiedad `client` a un objeto `clientObject`. Cuando otra aplicación llama a un método en esta instancia LocalConnection, el motor de ejecución busca ese método en el objeto `clientObject`.

Si ya se dispone de una conexión con el nombre especificado, se genera una excepción de error de argumento, que indica que el intento de conexión falló porque el objeto ya está conectado.

Cuando una instancia de Flash Player se conecta a este archivo SWF e intenta llamar a cualquier método en la conexión local especificada, la petición se envía a la clase especificada por la propiedad `client`, que se establece en la clase CustomClient1:

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = "")):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

Para crear un servidor LocalConnection, hay que llamar al método `LocalConnection.connect()` y proporcionar un nombre de conexión exclusivo. Si ya se dispone de una conexión con el nombre especificado, se genera un error ArgumentError, que indica que el intento de conexión falló porque el objeto ya está conectado.

El siguiente fragmento demuestra cómo crear un LocalConnection con el nombre `conn1`:

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

La conexión a una aplicación principal desde una aplicación secundaria requiere primero crear un objeto LocalConnection en el objeto LocalConnection que se envía y luego llamar al método `LocalConnection.send()` con el nombre de la conexión y del método para ejecutar. Por ejemplo, para enviar el método `doQuit` al objeto LocalConnection que ha creado anteriormente, use el siguiente código:

```
sendingConnection.send("conn1", "doQuit");
```

Este código se conecta a un objeto LocalConnection existente con el nombre de conexión `conn1` e invoca al método `doMessage()` en la aplicación remota. Si desea enviar parámetros a la aplicación remota, especifique argumentos adicionales después del nombre del método en el método `send()`, como muestra el siguiente fragmento:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Conexión al contenido en diferentes dominios y a otras aplicaciones de AIR

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para permitir la comunicación solo desde dominios específicos, llame al método `allowDomain()` o `allowInsecureDomain()` de la clase LocalConnection y pase una lista de uno o más dominios que tienen autorización para acceder al objeto LocalConnection, pasando uno o más nombres de dominios.

En versiones anteriores de ActionScript, `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()` eran métodos callback que implementaban los desarrolladores y que tenían que devolver un valor booleano. En ActionScript 3.0, `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()` son métodos incorporados a los que los desarrolladores llaman del mismo modo que `Security.allowDomain()` y `Security.allowInsecureDomain()`, pasando uno o varios nombres de dominios que deben autorizarse.

Flash Player 8 introdujo restricciones de seguridad en los archivos SWF locales. Un archivo SWF al que se le permite acceder a Internet no puede tener también acceso al sistema de archivos local. Si se especifica `localhost`, cualquier archivo SWF local podrá acceder al archivo SWF. Si el método `LocalConnection.send()` intenta comunicarse con un archivo SWF desde un entorno limitado de seguridad al que no puede acceder el código que realiza la llamada, se distribuye un evento `securityError(SecurityErrorEvent.SECURITY_ERROR)`. Para solucionar este error, puede especificar el dominio del origen de llamada en el método`LocalConnection.allowDomain()` del receptor.

Hay dos valores especiales que se pueden pasar a los métodos `LocalConnection.allowDomain()` y `LocalConnection.allowInsecureDomain()`: `*` y `localhost`. El valor de asterisco `(*)` permite el acceso desde todos los dominios. La cadena `localhost` permite llamadas a la aplicación desde el contenido localmente instalado, pero fuera del directorio de recursos de la aplicación.

Si el método `LocalConnection.send()` intenta comunicarse con una aplicación desde un entorno limitado de seguridad al que el código de llamada no tiene acceso, se distribuye un evento `securityError`(`SecurityErrorEvent.SECURITY_ERROR`). Para solucionar este error, puede especificar el dominio del origen de llamada en el método`LocalConnection.allowDomain()` del receptor.

Sí solo va a implementar la comunicación entre el contenido del mismo dominio, especifique un parámetro `connectionName` que no comience por un guion bajo (_) 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 contenidos en diferentes dominios, especifique una parámetro `connectionName` que comienza con un guion bajo. La especificación de los guiones bajos permite que el contenido con el objeto LocalConnection receptor tenga una mayor portabilidad entre dominios. Estos son los dos casos posibles:

- Si la cadena para `connectionName` no comienza por un guion bajo, el motor de ejecución añade un prefijo con el nombre del superdominio y dos puntos (por ejemplo, `myDomain:connectionName`). Aunque esto garantiza que la conexión no entre en conflicto con conexiones de otros dominios que tengan el mismo nombre, los objetos LocalConnection emisores deben especificar este superdominio (por ejemplo, `myDomain:connectionName`). Si el archivo HTML o SWF que contiene el objeto LocalConnection receptor se traslada a otro dominio, el motor de ejecución cambia el prefijo para reflejar el nuevo superdominio (por ejemplo, `anotherDomain:connectionName`). Se deben editar manualmente todos los objetos LocalConnection emisores para que apunten al nuevo superdominio.
- Si la cadena de `connectionName` comienza por un guion bajo (por ejemplo, `_connectionName"`), el motor de ejecución no añade un prefijo a la cadena. Esto significa que los objetos LocalConnection receptores y emisores utilizan idénticas cadenas para `connectionName`. Si el objeto receptor utiliza `LocalConnection.allowDomain()` para especificar que se acepten las conexiones de cualquier dominio, el SWF que contiene el objeto LocalConnection receptor se puede trasladar a otro dominio sin modificar ningún objeto LocalConnection emisor.

  La desventaja de utilizar nombres con guion bajo en `connectionName` es la posibilidad de interferencias, como por ejemplo cuando dos aplicaciones intentan conectarse utilizando el mismo `connectionName`. Una segunda desventaja relacionada se refiere a la seguridad. Los nombres de conexión que usan la sintaxis de guion bajo no identifican el dominio de la aplicación que se detecta. Por estas razones, se prefieren los nombres completos de dominio.

Adobe AIR

Para comunicarse con el contenido ejecutado en el entorno limitado de seguridad de la aplicación de AIR (contenido instalado con la aplicación de AIR), debe añadir un prefijo al nombre de conexión con un superdominio que identifique la aplicación de AIR. La cadena de superdominio comienza con `app#` seguida del ID de aplicación y un punto (.), y el ID del editor (si está definido). Por ejemplo, el superdominio correcto para utilizar en el parámetro `connectionName` de una aplicación con ID de aplicación `com.example.air.MyApp` y sin ID de editor sería: `"app#com.example.air.MyApp"`. Así, si el nombre de conexión base es “appConnection”, toda la cadena que se utilizará en el parámetro `connectionName` es: `"app#com.example.air.MyApp:appConnection"`. Si la aplicación tiene ID de editor, este también debe incluirse en la cadena de superdominio: `"app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1"`.

Cuando se permite que otra aplicación de AIR se comunique con la aplicación a través de la conexión local, se debe llamar a `allowDomain()` del objeto LocalConnection, pasando el nombre del dominio de conexión local. Para una aplicación de AIR, este nombre de dominio se forma del ID de editor y de la aplicación de la misma manera que la cadena de conexión. Por ejemplo, si la aplicación de AIR de envío tiene un ID de aplicación `com.example.air.FriendlyApp` y un ID de editor `214649436BD677B62C33D02233043EA236D13934.1`, entonces el la cadena de dominio que se usa para permitir la conexión de esta aplicación es: `app##com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1`. (En AIR 1.5.3, no todas las aplicaciones de AIR tienen ID de editor.)

Capítulo 46: Comunicación con procesos nativos en AIR

Adobe AIR 2 y posterior

Tal y como sucede con Adobe AIR 2, las aplicaciones de AIR se pueden ejecutar y comunicarse con otros procesos nativos mediante la línea de comandos. Por ejemplo, una aplicación de AIR puede ejecutar un proceso y comunicarse con él a través de los flujos de entrada y salida.

Para comunicarse con procesos nativos, empaquete una aplicación de AIR para que se instale a través de un archivo de instalación nativo. El tipo de archivo de instalación nativo es específico del sistema operativo para el cual se crea:

- En Mac OS es un archivo DMG.
- En Windows es un archivo EXE.
- En Linux es un paquete RPM o DEB.

Estas aplicaciones se conocen como aplicaciones de perfil de escritorio ampliado. Un archivo de instalación nativo se puede crear especificando la opción `-target native` al llamar al comando `-package` con el uso de ADT.

Más temas de ayuda

flash.filesystem.File.openWithDefaultApplication()

flash.desktop.NativeProcess

Información general sobre las comunicaciones de proceso nativo

Adobe AIR 2 y posterior

Una aplicación de AIR del perfil de escritorio ampliado puede ejecutar un archivo como si se invocase desde la línea de comandos. Se puede comunicar con los flujos estándar del proceso nativo. Entre los flujos estándar se incluyen el flujo de entrada estándar (stdin), el flujo de salida (stdout) y el flujo de error estándar (stderr).

Nota: *las aplicaciones del perfil de escritorio ampliado también pueden iniciar archivos y aplicaciones utilizando el método* `File.openWithDefaultApplication()`*. Sin embargo, el uso de este método no proporciona a la aplicación de AIR acceso a los flujos estándar. Para obtener más información, consulte "Apertura de archivos con la aplicación del sistema predeterminada" en la página 710*

En el siguiente ejemplo de código se muestra cómo iniciar una aplicación test.exe en el directorio de la aplicación. La aplicación transmite `"hello"` como argumento de la línea de comandos y añade un detector de eventos al flujo de salida estándar del proceso:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs.push("hello");
nativeProcessStartupInfo.arguments = processArgs;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, onOutputData);
process.start(nativeProcessStartupInfo);
public function onOutputData(event:ProgressEvent):void
{
    var stdOut:ByteArray = process.standardOutput;
    var data:String = stdOut.readUTFBytes(process.standardOutput.bytesAvailable);
    trace("Got: ", data);
}
```

Inicio y cierre de un proceso nativo

Adobe AIR 2 y posterior

Para iniciar un proceso nativo, configure un objeto NativeProcessInfo para que realice lo siguiente:

- Seleccione el archivo que desee iniciar.
- Almacene argumentos de la línea de comandos para transmitir el proceso cuando se inicie. (opcional)
- Establezca el directorio de trabajo del proceso (opcional)

Para iniciar el proceso nativo, transmita el objeto NativeProcessInfo como parámetro del método `start()` de un objeto NativeProcess.

Por ejemplo, el siguiente código muestra cómo iniciar una aplicación test.exe en el directorio de la aplicación. La aplicación transmite el argumento `"hello"` y establece el directorio de documentos del usuario como directorio de trabajo:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs[0] = "hello";
nativeProcessStartupInfo.arguments = processArgs;
nativeProcessStartupInfo.workingDirectory = File.documentsDirectory;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
```

Para terminar el proceso, llame al método `exit()` del objeto NativeProcess.

Si quiere que un archivo sea ejecutable en su aplicación instalada, asegúrese de que lo es en el sistema de archivos en el momento de empaquetar la aplicación. (En Mac y Linux, puede utilizar chmod para establecer el indicador de ejecutable, si es necesario.)

Comunicación con un proceso nativo

Adobe AIR 2 y posterior

Una vez que una aplicación de AIR se ha iniciado como proceso nativo, puede comunicarse con los flujos de error y de entrada y salida estándar del proceso.

Los datos se leen y se escriben en los flujos utilizando las siguientes propiedades del objeto NativeProcess:

- `standardInput`: contiene acceso a los datos del flujo de entrada estándar.
- `standardOutput`: contiene acceso a los datos del flujo de salida estándar.
- `standardError`: contiene acceso a los datos del flujo de error estándar.

Escritura en el flujo de entrada estándar

Los datos se pueden escribir en el flujo de entrada estándar utilizando los métodos de escritura de la propiedad `standardInput` del objeto NativeProcess. A medida que la aplicación de AIR escribe datos en el proceso, el objeto NativeProcess distribuye eventos `standardInputProgress`.

Si se produce un error al escribir en el flujo de entrada estándar, el objeto NativeProcess distribuye un evento `ioErrorStandardInput`.

El flujo de entrada se puede cerrar llamando al método `closeInput()` del objeto NativeProcess. Cuando el flujo de entrada se cierra, el objeto NativeProcess distribuye un evento `standardInputClose`.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
process.standardInput.writeUTF("foo");
if(process.running)
{
    process.closeInput();
}
```

Lectura en el flujo de salida estándar

Los datos se pueden leer en el flujo de salida estándar utilizando los métodos de lectura de esta propiedad. A medida que la aplicación de AIR obtiene datos del flujo de salida del proceso, el objeto NativeProcess distribuye eventos `standardOutputData`.

Si se produce un error al escribir en el flujo de salida estándar, el objeto NativeProcess distribuye un evento `standardOutputError`.

Cuando el proceso cierra el flujo de salida, el objeto NativeProcess distribuye el evento `standardOutputClose`.

Cuando se leen datos del flujo de entrada estándar, es muy importante leerlos a medida que se generan. Dicho de otro modo, añada un detector de eventos para el evento `standardOutputData`. En el detector de eventos `standardOutputData`, lea los datos de la propiedad `standardOutput` del objeto NativeProcess. No se limite a esperar a que el evento `standardOutputClose` o `exit` lea todos los datos. Si no lee los datos a medida que el proceso nativo los genera, el búfer podría llenarse y los datos podrían perderse. Un búfer lleno puede provocar que el proceso nativo se bloquee al intentar escribir más datos. Sin embargo, si no registra un detector de eventos para el evento `standardOutputData`, el búfer no se llenará y el proceso no se bloqueará. En ese caso, no podrá acceder a los datos.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, dataHandler);
process.start(nativeProcessStartupInfo);
var bytes:ByteArray = new ByteArray();
function dataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardOutput.readBytes(process.standardOutput.bytesAvailable);
}
```

Lectura del flujo de error estándar

Los datos se pueden leer en el flujo de error estándar utilizando los métodos de lectura de esta propiedad. A medida que la aplicación de AIR lee los datos del flujo de error del proceso, el objeto NativeProcess distribuye eventos `standardErrorData`.

Si se produce un error al escribir en el flujo de error estándar, el objeto NativeProcess distribuye un evento `standardErrorIoError`.

Cuando el proceso cierra el flujo de error, el objeto NativeProcess distribuye un evento `standardErrorClose`..

Cuando se leen datos del flujo de error estándar, es muy importante leerlos a medida que se generan. Dicho de otro modo, añada un detector de eventos para el evento `standardErrorData`. En el detector de eventos `standardErrorData`, lea los datos de la propiedad `standardError` del objeto NativeProcess. No se limite a esperar a que el evento `standardErrorClose` o `exit` lea todos los datos. Si no lee los datos a medida que el proceso nativo los genera, el búfer podría llenarse y los datos podrían perderse. Un búfer lleno puede provocar que el proceso nativo se bloquee al intentar escribir más datos. Sin embargo, si no registra un detector de eventos para el evento `standardErrorData`, el búfer no se llenará y el proceso no se bloqueará. En ese caso, no podrá acceder a los datos.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_ERROR_DATA, errorDataHandler);
process.start(nativeProcessStartupInfo);
var errorBytes:ByteArray = new ByteArray();
function errorDataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardError.readBytes(process.standardError.bytesAvailable);
}
```

Seguridad en la comunicación del proceso nativo

Adobe AIR 2 y posterior

La API del proceso nativo puede ejecutar cualquier ejecutable en el sistema del usuario. Debe tener mucho cuidado al construir y ejecutar comandos. Si algún fragmento de un comando que se ejecuta viene desde un origen externo, compruebe atentamente que es seguro ejecutar dicho comando. Asimismo, la aplicación de AIR debe validar datos transferidos a un proceso en ejecución.

Sin embargo, validar los datos entrantes puede resultar difícil. Para evitar esta dificultad, es mejor escribir una aplicación nativa (como un archivo EXE en Windows) con API específicas. Estas API deben procesar solo dichos comandos definidos por la aplicación. Por ejemplo, la aplicación solo puede aceptar una serie limitada de instrucciones a través de la secuencia de entrada estándar.

En Windows, AIR no le permite ejecutar archivos .bat directamente. La aplicación del interpretador de comandos (cmd.exe) ejecuta archivos .bat de Windows. Cuando se invoca un archivo .bat, esta aplicación de comandos puede interpretar argumentos que se transfieren al comando como aplicaciones adicionales para iniciar. La presencia maliciosa de caracteres adicionales en la cadena del argumento puede provocar que un cmd.exe ejecute una aplicación insegura o dañina. Por ejemplo, sin una validación adecuada de datos, la aplicación de AIR puede llamar a `myBat.bat myArguments c:/diablo.exe`. La aplicación de comandos iniciaría la aplicación diablo.exe además de ejecutar el archivo por lotes.

Capítulo 47: Uso de la API externa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La API externa de ActionScript 3.0 (flash.external.ExternalInterface) permite una comunicación directa entre ActionScript y la aplicación contenedora en la que se ejecuta Adobe Flash Player. Utilice la API ExternalInterface para crear interacción entre un documento SWF y JavaScript en una página HTML.

Puede utilizar la API externa para interactuar con una aplicación contenedora o transferir datos entre ActionScript y JavaScript en una página HTML.

Algunas tareas habituales de la API externa son:

- Obtener información acerca de la aplicación contenedora
- Usar ActionScript para llamar al código en una página web mostrada en un navegador o en una aplicación de AIR para escritorio
- Llamar al código ActionScript desde una página web
- Crear un proxy para simplificar las llamadas al código de ActionScript desde una página web

Nota: *esta sección sobre la interfaz externa solo describe la comunicación entre el código ActionScript de un archivo SWF y la aplicación contenedora que incluye una referencia a la instancia de Flash Player en la que se carga el archivo SWF. En esta documentación no se tratan otros usos de Flash Player en una aplicación. Flash Player se ha diseñado para utilizarse como un complemento de navegador o un proyector (aplicación autónoma). Puede haber compatibilidad limitada con otros escenarios de uso.*

Uso de la API externa en AIR

Dado que una aplicación de AIR no tiene ningún contenedor externo, esta interfaz externa no suele activarse (ni suele necesitarse). Si su aplicación de AIR carga un archivo SWF directamente, el código de la aplicación puede comunicarse directamente con el código ActionScript del SWF (sujeto a restricciones del entorno limitado de seguridad).

No obstante, cuando su aplicación de AIR carga un archivo SWF mediante una página HTML en un objeto HTMLLoader (o un componente HTML en Flex), el objeto HTMLLoader hace de contenedor externo. De este modo, puede utilizar la interfaz externa para comunicarse entre el código ActionScript en el archivo SWF cargado y el código JavaScript en la página HTML cargada en el objeto HTMLLoader.

Fundamentos de la utilización de la API externa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si bien en algunos casos un archivo SWF se puede ejecutar de forma independiente (por ejemplo, si se utiliza Adobe® Flash® Professional para crear un proyector SWF), en la mayor parte de las ocasiones las aplicaciones SWF se ejecutan como elementos incorporados dentro de otras aplicaciones. Normalmente, el contenedor que incluye el archivo SWF es un archivo HTML. Menos frecuente es usar un archivo SWF de modo que constituya total o parcialmente la interfaz de usuario de una aplicación de escritorio.

Cuando se trabaja en aplicaciones más avanzadas es posible que sea necesario establecer una comunicación entre el archivo SWF y la aplicación contenedora. Por ejemplo, es habitual que una página web muestre texto u otra información en HTML e incluya un archivo SWF en el que aparezca contenido visual dinámico, como un diagrama o un vídeo. En esos casos podría resultar útil que cuando los usuarios hicieran clic en un botón de la página web, algo cambiase en el archivo SWF. ActionScript contiene un mecanismo, conocido como la API externa, que facilita este tipo de comunicación entre el código ActionScript de un archivo SWF y el código de la aplicación contenedora.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes que son relevantes para esta función:

Aplicación contenedora La aplicación en la que Flash Player está ejecutando un archivo SWF, como un navegador web, y la página HTML que incluye contenido de Flash Player o una aplicación de AIR que carga el SWF en una página web.

Proyector Archivo ejecutable que incluye contenido SWF y una versión incorporada de Flash Player. Es posible crear un archivo de proyector mediante Flash Professional o la versión autónoma de Flash Player. Los proyectores suelen usarse para distribuir archivos SWF en CD-ROM o en otras situaciones similares en las que el tamaño de la descarga no es importante y el autor del SWF desea estar seguro de que los usuarios podrán ejecutar el archivo independientemente de que tengan instalado Flash Player en el equipo.

Proxy Aplicación o código intermedio que llama a código de una aplicación (la "aplicación externa") en nombre de otra aplicación (la "aplicación que llama") y devuelve valores a esta última. Un proxy se puede usar por muy diversas razones, entre las que se encuentran las siguientes:

- Para simplificar el proceso de realización de llamadas a funciones externas, convirtiendo las llamadas a funciones nativas de la aplicación que llama en un formato comprensible por la aplicación externa..
- Para evitar limitaciones de seguridad u otras restricciones que impiden a la aplicación que llama comunicarse directamente con la aplicación externa.

Serializar Convertir objetos o valores de datos a un formato que se pueda utilizar para transmitir los valores en mensajes entre dos sistemas de programación, como a través de Internet o entre dos aplicaciones diferentes que se ejecutan en un mismo equipo.

Ejecución de los ejemplos del capítulo

Muchos de los ejemplos de código proporcionados son pequeños fragmentos de código incluidos con fines de demostración, no ejemplos completos o código de comprobación de valores. Como el uso de la API externa requiere (por definición) escribir código ActionScript así como código en una aplicación contenedora, para probar los ejemplos hay que crear un contenedor (por ejemplo, una página web que contenga el archivo SWF) y utilizar los listados de código para interactuar con el contenedor.

Para probar un ejemplo de comunicación entre ActionScript y JavaScript:

1. Cree un nuevo documento utilizando Flash Professional y guárdelo en su equipo.
2. En el menú principal, elija Archivo > Configuración de publicación.
3. En el cuadro de diálogo Configuración de publicación, en la ficha Formatos, compruebe que solo están activadas las casillas de verificación HTML y Flash.
4. Haga clic en el botón Publicar. Esto genera un archivo SWF y un archivo HTML en la misma carpeta y con el mismo nombre que utilizó para guardar el documento de Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de publicación.
5. Desactive la casilla de verificación HTML. Una vez generada la página HTML, va a modificarla para añadir el código JavaScript apropiado. Si desactiva la casilla de verificación HTML, cuando modifique la página HTML, Flash no sobrescribirá los cambios con una nueva página HTML al publicar el archivo SWF.

6 Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración de publicación.

7 Con un editor de HTML o texto, abra el archivo HTML creado por Flash al publicar el archivo SWF. En el código fuente HTML, agregue etiquetas de apertura y cierre `script` y cópielas en el código JavaScript del listado de código de ejemplo:

```
<script>
// add the sample JavaScript code here
</script>
```

8 Guarde el archivo HTML y vuelva a Flash.

9 Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.

10 Copie el listado de código ActionScript en el panel Script.

11 En el menú principal, elija Archivo > Publicar para actualizar el archivo SWF con los cambios realizados.

12 Abra la página HTML editada en un navegador web para verla y probar la comunicación entre ActionScript y la página HTML.

Para probar un ejemplo de comunicación entre ActionScript y un contenedor ActiveX:

1 Cree un nuevo documento utilizando Flash Professional y guárdelo en su equipo. Puede guardarlo en cualquier carpeta en la que la aplicación contenedora espere encontrar el archivo SWF.

2 En el menú principal, elija Archivo > Configuración de publicación.

3 En el cuadro de diálogo Configuración de publicación, en la ficha Formatos, compruebe que solo está activada la casilla de verificación Flash.

4 En el campo Archivo situado junto a la casilla de verificación Flash, haga clic en el icono de carpeta para seleccionar la carpeta en la que desea que se publique el archivo SWF. Al establecer la ubicación del archivo SWF se puede, por ejemplo, mantener el documento en una carpeta y colocar el archivo SWF publicado en otra carpeta, como la carpeta que contiene el código fuente de la aplicación contenedora.

5 Seleccione el fotograma clave del Fotograma 1 de la línea de tiempo y abra el panel Acciones.

6 Copie el listado de código ActionScript del ejemplo en el panel Script.

7 En el menú principal, elija Archivo > Publicar para volver a publicar el archivo SWF.

8 Cree y ejecute la aplicación contenedora para probar la comunicación entre ActionScript y dicha aplicación.

Para ver ejemplos completos del uso de la API externa para comunicarse con una página HTML, consulte el siguiente tema:

- "Ejemplo de API externa: Comunicación entre ActionScript y JavaScript en un navegador web" en la página 883

Estos ejemplos incluyen el código completo, incluido el código de comprobación de errores de ActionScript y de la aplicación contenedora que se debe utilizar al escribir código con la API externa. Otro ejemplo completo de uso de la API externa es el ejemplo de la clase ExternalInterface de la Referencia de ActionScript 3.0.

Requisitos y ventajas de la API externa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La API externa es la parte de ActionScript que proporciona un mecanismo de comunicación entre ActionScript y el código que se ejecuta en una "aplicación externa", que actúa a modo de contenedor de Flash Player (normalmente un navegador web o una aplicación de proyector independiente). En ActionScript 3.0, la funcionalidad de la API externa viene dada por la clase ExternalInterface. En las versiones de Flash Player anteriores a Flash Player 8, se usaba la acción `fscommand()` para llevar a cabo la comunicación con la aplicación contenedora. La clase ExternalInterface es una sustitución de `fscommand()`.

Nota: *la antigua función `fscommand()` aún está disponible como función a nivel de paquete en el paquete flash.system para los casos en los que resulte necesario usarla (por ejemplo para mantener la compatibilidad con aplicaciones anteriores o para interactuar con una aplicación contenedora de SWF de un tercero o con la versión autónoma de Flash Player).*

La clase ExternalInterface es un subsistema que permite comunicarse fácilmente entre ActionScript y Flash Player y JavaScript en una página HTML.

La clase ExternalInterface solo está disponible en los siguientes casos:

- En todas las versiones compatibles de Internet Explorer para Windows (5.0 y versiones posteriores)
- En cualquier navegador que admita la interfaz NPRuntime, que actualmente incluye Firefox 1.0 y posterior, Mozilla 1.7.5 y posterior, Netscape 8.0 y posterior y Safari 1.3 y posterior.
- En una aplicación de AIR cuando se incorpora el SWF en una página HTML visualizada por el control HTMLLoader.

En todas las demás situaciones (como al ejecutarse en un reproductor autónomo), la propiedad `ExternalInterface.available` devuelve el valor `false`.

Desde ActionScript se puede llamar a una función de JavaScript en la página HTML. La API externa ofrece las siguientes mejoras con respecto a `fscommand()`:

- Se puede utilizar cualquier función de JavaScript, no solo que se usan con la función `fscommand()`.
- Se puede pasar un número arbitrario de argumentos y los argumentos pueden tener el nombre que se desee; no existe la limitación de pasar un comando con un único argumento de tipo cadena. Esto hace que la API externa sea mucho más flexible que `fscommand()`.
- Se pueden pasar diversos tipos de datos (como Boolean, Number y String); ya no se está limitado a los parámetros de tipo String.
- Se puede recibir el valor de una llamada y ese valor vuelve inmediatamente a ActionScript (como valor devuelto de la llamada que se realiza).

Importante: *si el nombre que se asigna a la instancia de Flash Player en una página HTML (el atributo `id` de la etiqueta `object`) incluye un guion (`-`) u otros caracteres que se definen como operadores en JavaScript (por ejemplo, `+`, `*`, `/`, `\`, `.`, etc), las llamadas a ExternalInterface desde ActionScript no funcionarán cuando la página web contenedora se visualice en Internet Explorer. Asimismo, si las etiquetas HTML que definen la instancia de Flash Player (etiquetas `object` y `embed`) se anidan en una etiqueta HTML `form`, las llamadas a ExternalInterface desde ActionScript no funcionarán.*

Uso de la clase ExternalInterface

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La comunicación entre ActionScript y la aplicación contenedora puede adoptar una de las siguientes formas: ActionScript puede llamar al código (por ejemplo, una función de JavaScript) definido en el contenedor, o bien, el código del contenedor puede llamar una función de ActionScript que se haya designado como función que es posible llamar. En cualquiera de los casos, se puede enviar la información al código que se está llamando y se pueden devolver los resultados al código que realiza la llamada.

Para facilitar esta comunicación, la clase ExternalInterface incluye dos propiedades estáticas y dos métodos estáticos. Estas propiedades y métodos se utilizan para obtener información acerca de la conexión de la interfaz externa, para ejecutar código en el contenedor desde ActionScript y para hacer que las funciones ActionScript estén disponibles para ser llamadas desde el contenedor.

Obtención de información sobre el contenedor externo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La propiedad `ExternalInterface.available` indica si Flash Player se encuentra en esos momentos en un contenedor que ofrece una interfaz externa. Si la interfaz externa está disponible, esta propiedad es `true`; en caso contrario, es `false`. Antes de utilizar cualquier otra funcionalidad de la clase ExternalInterface, es recomendable comprobar que el contenedor que se está utilizando en esos momentos permite la comunicación a través de la interfaz externa del modo siguiente:

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

Nota: *la propiedad `ExternalInterface.available` indica si el contenedor que se está utilizando en esos momentos es de un tipo compatible con la conectividad mediante ExternalInterface. No indicará si JavaScript está habilitado en el navegador.*

La propiedad `ExternalInterface.objectID` permite determinar el identificador exclusivo de la instancia de Flash Player (específicamente, el atributo `id` de la etiqueta `object` en Internet Explorer o el atributo `name` de la etiqueta `embed` en los navegadores que usan la interfaz NPRuntime). Este identificador exclusivo representa al documento SWF actual en el navegador y se puede utilizar para hacer referencia al documento SWF (por ejemplo, al llamar a una función de JavaScript en una página HTML contenedora). Cuando el contenedor de Flash Player no es un navegador web, esta propiedad es `null`.

Llamada a código externo desde ActionScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `ExternalInterface.call()` ejecuta código en la aplicación contenedora. Como mínimo requiere un parámetro, una cadena que contenga el nombre de la función que se va a llamar en la aplicación contenedora. Todos los parámetros adiciones que se pasen al método `ExternalInterface.call()` se pasarán a su vez al contenedor como parámetros de la llamada a la función.

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Si el contenedor es una página HTML, este método invocará a la función de JavaScript con el nombre especificado, que debe estar definida en un elemento `script` en la página HTML contenedora. El valor devuelto por la función de JavaScript se devuelve a ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Si el contenedor es algún otro contenedor ActiveX, este método hace que el control ActiveX de Flash Player distribuya su evento `FlashCall`. Flash Player serializa en una cadena XML el nombre de la función especificada y todos los parámetros. El contenedor puede acceder a esa información de la propiedad `request` del objeto de evento y utilizarla para determinar el modo en que se ejecutará su propio código. Para devolver un valor a ActionScript, el código del contenedor llama al método `SetReturnValue()` del objeto ActiveX y pasa el resultado (serializado en una cadena XML) como parámetro de ese método. Para obtener más información sobre el formato XML utilizado para esta comunicación, consulte "Formato XML de la API externa" en la página 881.

Tanto si el contenedor es un navegador web como si otro contenedor ActiveX, en el caso de que se produzca un error en la llamada o el método del contenedor no especifique un valor devuelto, se devolverá `null`. El método `ExternalInterface.call()` emite una excepción SecurityError si el entorno contenedor pertenece a un entorno limitado de seguridad al que no tiene acceso el código que realiza la llamada. Se puede solucionar esta limitación asignando un valor adecuado a `allowScriptAccess` en el entorno contenedor. Por ejemplo, para cambiar el valor de `allowScriptAccess` en una página HTML, es necesario editar el atributo adecuado en las etiquetas `object` y `embed`.

Llamadas a código ActionScript desde el contenedor

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un contenedor solo puede llamar a código ActionScript que se encuentre en una función; no puede llamar a ningún otro código ActionScript. Para llamar una función de ActionScript desde la aplicación contenedora, debe realizar dos operaciones: registrar la función con la clase ExternalInterface y posteriormente llamarla desde el código del contenedor.

En primer lugar se debe registrar la función de ActionScript para indicar que debe ponerse a disposición del contenedor. Para ello se usa el método `ExternalInterface.addCallback()` del modo siguiente:

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

El método `addCallback()` utiliza dos parámetros: El primero, un nombre de función con formato String, es el nombre por el que se conocerá a la función en el contenedor. El segundo parámetro es la función real de ActionScript que se ejecutará cuando el contenedor llame al nombre de función definido. Dado que estos dos nombres son distintos, se puede especificar un nombre de función que se utilizará en el contenedor incluso si la función real de ActionScript tiene un nombre distinto. Esto resulta especialmente útil si no se conoce el nombre de la función (por ejemplo, cuando se especifica una función anónima o cuando la función que ha de llamarse se determina en tiempo de ejecución).

Una vez que la función de ActionScript se registra en la clase ExternalInterface, el contenedor puede llamar a la función. La forma de realizar esto varía según el tipo de contenedor. Por ejemplo, en el código JavaScript en un navegador web, la función de ActionScript se llama utilizando el nombre de función registrado como si fuese un método del objeto del navegador de Flash Player (es decir, un método del objeto JavaScript que representa a la etiqueta `object` o `embed)`. Dicho de otro modo, se pasan los parámetros y se devuelve el resultado como si se llamase a una función local.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

Por otra parte, al llamar a una función de ActionScript en un archivo SWF que se ejecuta en una aplicación de escritorio, el nombre de función registrado y los parámetros deben serializarse en una cadena con formato XML. A continuación, se efectúa la llamada llamando al método `CallFunction()` del control ActiveX con la cadena XML como parámetro. Para obtener más información sobre el formato XML utilizado para esta comunicación, consulte "Formato XML de la API externa" en la página 881.

En cualquiera de los dos casos, el valor devuelto por la función de ActionScript se pasa al código del contenedor, ya sea directamente como un valor si la llamada se origina en el código JavaScript de un navegador o serializado como una cadena con formato XML si la llamada se origina en un contenedor ActiveX.

Formato XML de la API externa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En la comunicación entre ActionScript y la aplicación que aloja el control ActiveX de Shockwave Flash se utiliza un formato XML específico para codificar las llamadas a las funciones y los valores. El formato XML usado por la API externa tiene dos partes. Se utiliza un formato para representar las llamadas a funciones. El otro se utiliza para representar valores individuales; este formato se emplea para parámetros de funciones y para los valores devueltos por las funciones. El formato XML para llamadas a funciones se utiliza para llamadas desde y hacia ActionScript. En el caso de una llamada a una función desde ActionScript, Flash Player pasa los datos XML al contenedor; cuando se trata de una llamada desde el contenedor, Flash Player espera que la aplicación contenedora pase una cadena XML con este formato. El siguiente fragmento XML muestra un ejemplo de llamada a función con formato XML:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

El nodo raíz es `invoke`, Cuenta con dos atributos: `name` indica el nombre de la función que se va a llamar y `returntype` siempre es `xml`. Si la llamada a la función incluye parámetros, el nodo `invoke` tendrá un nodo secundario `arguments`, cuyos nodos secundarios serán los valores de los parámetros a los que se ha dado el formato de valor individual que se explica a continuación.

Los valores individuales, incluidos los parámetros de las funciones y los valores devueltos por estas, utilizan un esquema de formato que, además de los valores reales, incluye información sobre los tipos de datos. En la siguiente tabla se indican las clases de ActionScript y el formato XML utilizado para codificar los valores de ese tipo de datos:

Clase/Valor de ActionScript	**Clase/Valor de C#**	**Formato**	**Comentarios**
null	null	<null/>	
Boolean `true`	bool true	<true/>	
Boolean `false`	bool false	<false/>	
String	cadena	<string>valor de cadena</string>	
Number, int, uint	single, double, int, uint	`<number>27.5</number>` `<number>-12</number>`	

Array (los elementos pueden ser de varios tipos)	Un conjunto que admite elementos de varios tipos, como ArrayList u object[]	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			El nodo `property` define elementos individuales y el atributo `id` es el índice numérico de base cero.

Object	Un diccionario con claves de tipo cadena y valores de objeto, como un objeto HashTable con claves de tipo cadena	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			El nodo `property` define propiedades individuales y el atributo `id` es el nombre de la propiedad (una cadena).

Otras clases incorporadas o personalizadas		`<null/> or` `<object></object>`	ActionScript codifica otros objetos como null o como un objeto vacío. En ambos casos, los valores de la propiedad se pierden.

Nota: *a modo de ejemplo, esta tabla muestra las clases de C# equivalentes además de las clases de ActionScript; no obstante, la API externa se puede usar para comunicarse con cualquier lenguaje de programación o entorno de tiempo de ejecución compatible con los controles ActiveX y no se limita a las aplicaciones C#.*

Cuando se crean aplicaciones propias utilizando la API externa con una aplicación contenedora ActiveX, suele resultar cómodo escribir un proxy encargado de la tarea de convertir las llamadas a funciones nativas en el formato XML serializado. Para ver un ejemplo de una clase proxy escrita en C#, consulte Dentro de la clase ExternalInterfaceProxy.

Ejemplo de API externa: Comunicación entre ActionScript y JavaScript en un navegador web

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En esta aplicación de ejemplo se muestran técnicas apropiadas para establecer una comunicación entre ActionScript y JavaScript en un navegador web en el contexto de una aplicación de mensajería instantánea que permite a una persona charlar consigo misma (de ahí el nombre de la aplicación: Introvert IM). Los mensajes se envían entre un formulario HTML en la página web y una interfaz SWF mediante la API externa. Este ejemplo ilustra las siguientes técnicas:

- Iniciación adecuada de la comunicación comprobando que el navegador está listo para ello antes de establecer la comunicación.
- Comprobación de que el contenedor es compatible con la API externa.
- Llamada a funciones de JavaScript desde ActionScript, paso de parámetros y recepción de valores como respuesta.
- Preparación de métodos de ActionScript para que puedan ser llamados y ejecución de dichas llamadas.

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Introvert IM se encuentran en la carpeta Samples/IntrovertIM_HTML. La aplicación consta de los siguientes archivos:

Archivo	Descripción
IntrovertIMApp.fla o IntrovertIMApp.mxml	El archivo de aplicación principal para Flash (FLA) o Flex (MXML).
com/example/programmingas3/introvertIM/IMManager.as	La clase que establece y gestiona la comunicación entre ActionScript y el contenedor.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Tipo de evento personalizado distribuido por la clase IMManager cuando se recibe un mensaje del contenedor.
com/example/programmingas3/introvertIM/IMStatus.as	Enumeración cuyos valores representan los distintos estados de "disponibilidad" que se pueden seleccionar en la aplicación.
html-flash/IntrovertIMApp.html o html-template/index.template.html	La página HTML de la aplicación para Flash (html-flash/IntrovertIMApp.html) o la plantilla que se utiliza para crear la página HTML contenedora de la aplicación para Adobe Flex (html-template/index.template.html). Este archivo contiene todas las funciones de JavaScript que constituyen la parte contenedora de la aplicación.

Preparación de la comunicación ActionScript-navegador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Uno de los usos más frecuentes de la API externa es el de permitir a las aplicaciones ActionScript comunicarse con un navegador web. Gracias a la API externa, los métodos de ActionScript pueden llamar a código escrito en JavaScript y viceversa. A causa de la complejidad de los navegadores y al modo en el que representan internamente las páginas, no hay manera de garantizar que un documento SWF registrará sus funciones de repetición de llamada antes de que se ejecute el primer fragmento de código JavaScript de la página HTML. Por ese motivo, antes de llamar a las funciones del documento SWF desde JavaScript, es aconsejable que el documento SWF llame siempre a la página HTML para notificarle que está listo para aceptar conexiones.

Por ejemplo, mediante una serie de pasos efectuados por la clase IMManager, Introvert IM determina si el navegador está listo para la comunicación y prepara el archivo SWF para la comunicación. El último paso, que consiste en determinar cuándo está listo el navegador para la comunicación, ocurre en el constructor de IMManager del modo siguiente:

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

En primer lugar, el código comprueba si la API externa está disponible en el contendor actual utilizando la propiedad `ExternalInterface.available`. En caso afirmativo, inicia el proceso de establecimiento de la comunicación. Dado que pueden producirse excepciones de seguridad y otros errores al intentar comunicarse con una aplicación externa, el código se escribe dentro de un bloque `try` (los bloques `catch` correspondientes se han omitido del listado en aras de la brevedad).

A continuación, el código llama al método `isContainerReady()`, que se muestra aquí:

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

El método `isContainerReady()` usa a su vez el método `ExternalInterface.call()` para llamar a la función de JavaScript `isReady()` del modo siguiente:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

La función `isReady()` simplemente devuelve el valor de la variable `jsReady` . Inicialmente, la variable tiene el valor `false`; una vez que el evento `onload` de la página web se ha activado, el valor de la variable cambia a `true`. Dicho de otro modo, si ActionScript llama a la función `isReady()` antes de que se cargue la página, JavaScript devuelve `false` a `ExternalInterface.call("isReady")` y, por lo tanto, el método `isContainerReady()` de ActionScript devuelve `false`. Una vez que la página se ha cargado, la función `isReady()` de JavaScript devuelve `true`, de modo que el método `isContainerReady()` de ActionScript también devuelve `true`.

De vuelta en el constructor de IMManager ocurre una de las dos cosas siguientes, en función de la disponibilidad del contenedor. Si `isContainerReady()` devuelve `true`, el código simplemente llama al método `setupCallbacks()`, que completa el proceso de configuración de la comunicación con JavaScript. Por otra parte, si `isContainerReady()` devuelve `false`, el proceso se pone en espera. Se crea un objeto Timer y se le indica que llame al método `timerHandler()` cada 100 milisegundos del modo siguiente:

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

Cada vez que se llama al método `timerHandler()`, este vuelve a comprobar el resultado del método `isContainerReady()`. Una vez que se inicializa el contenedor, el método devuelve `true`. En ese momento, el código detiene el temporizador y llama al método `setupCallbacks()` para terminar el proceso de configuración de las comunicaciones con el navegador.

Exposición de los métodos de ActionScript a JavaScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Como se muestra en el ejemplo anterior, una vez que el código determina que el navegador está listo, se llama al método `setupCallbacks()`. Este método prepara a ActionScript para recibir llamadas desde JavaScript tal y como se muestra a continuación:

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

El método `setCallBacks()` finaliza la tarea de preparar la comunicación con el contenedor llamando a `ExternalInterface.addCallback()` para registrar los dos métodos que estarán disponibles para ser llamados desde JavaScript. En este código, el primer parámetro (el nombre por el que el método se conoce en JavaScript, `"newMessage"` y `"getStatus"`) es el mismo que el nombre del método en ActionScript (en este caso no había ninguna ventaja en usar nombres distintos, así que se reutilizó el mismo nombre por simplificar). Por último, el método `ExternalInterface.call()` se utiliza para llamar a la función `setSWFIsReady()` de JavaScript, que notifica al contenedor que las funciones ActionScript se han registrado.

Comunicación desde ActionScript al navegador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La aplicación Introvert IM presenta toda una gama de ejemplos de llamadas a funciones JavaScript en la página contenedora. En el caso más simple (un ejemplo del método `setupCallbacks()`), se llama a la función `setSWFIsReady()` de JavaScript sin pasarle ningún parámetro ni recibir ningún valor devuelto:

```
ExternalInterface.call("setSWFIsReady");
```

En otro ejemplo del método `isContainerReady()`, ActionScript llama a la función `isReady()` y recibe un valor Boolean como respuesta:

```
var result:Boolean = ExternalInterface.call("isReady");
```

También se puede pasar parámetros a funciones de JavaScript a través de la API externa. Esto puede observarse, por ejemplo, en el método `sendMessage()` de la clase IMManager, al que se llama cuando el usuario envía un mensaje nuevo a su "interlocutor":

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

De nuevo se utiliza `ExternalInterface.call()` para llamar a la función de JavaScript designada, que notifica al navegador que hay un nuevo mensaje. Además, el mensaje en sí se pasa como un parámetro adicional a `ExternalInterface.call()` y, por lo tanto, se pasa como un parámetro a la función de JavaScript `newMessage()`.

Llamadas a código de ActionScript desde JavaScript

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La comunicación suele ser bidireccional y, en este sentido, la aplicación Introvert IM no es ninguna excepción. No solo el cliente de mensajería instantánea de Flash Player llama a JavaScript para enviar mensajes, sino que también el formulario HTML llama al código JavaScript para enviar mensajes y recibir información del archivo SWF. Por ejemplo, cuando el archivo SWF notifica al contenedor que ha terminado de establecer contacto y que está listo para empezar a comunicarse, lo primero que hace el navegador es llamar al método `getStatus()` de la clase IMManager para recuperar el estado de disponibilidad inicial del usuario desde el cliente de mensajería instantánea de SWF. Esto se lleva a cabo en la página web, en la función `updateStatus()`, del modo siguiente:

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

El código comprueba el valor de la variable `swfReady`, que determina si el archivo SWF ha notificado al navegador que este ha registrado sus métodos en la clase ExternalInterface. Si el archivo SWF está listo para recibir comunicaciones, la siguiente línea (`var currentStatus =` ...) llama al método `getStatus()` de la clase IMManager. En esta línea de código ocurren tres cosas:

- Se llama a la función `getSWF()` de JavaScript, que devuelve una referencia al objeto JavaScript que representa al archivo SWF. El parámetro pasado a `getSWF()` determina qué objeto de navegador se devuelve en caso de que haya más un archivo SWF en una página HTML. El valor pasado a ese parámetro debe coincidir con el atributo `id` de la etiqueta `object` y el atributo `name` de la etiqueta `embed` utilizada para incluir el archivo SWF.
- Se utiliza la referencia al archivo SWF para llamar al método `getStatus()` como si fuese un método del objeto SWF. En este caso, se utiliza el nombre de función "`getStatus`", ya que ese el nombre con el que se ha registrado la función de ActionScript mediante `ExternalInterface.addCallback()`.

- El método `getStatus()` de ActionScript devuelve un valor, que se asigna a la variable `currentStatus`, que se asigna a su vez como contenido (la propiedad `value`) del campo de texto `status`.

Nota: *si sigue el código, probablemente se habrá dado cuenta de que en el código fuente de la función `updateStatus()`, la línea de código que llama a la función `getSWF()` en realidad se escribe del modo siguiente: var currentStatus = getSWF("${application}").getStatus(); El texto `${application}` es un marcador de posición de la plantilla de la página HTML; cuando Adobe Flex Builder genera la página HTML real de la aplicación, este texto se sustituye por el texto utilizado como atributo `id` de la etiqueta `object` y como atributo `name` de la etiqueta `embed` (`IntrovertIMApp` en el ejemplo). Ese es el valor que espera la función `getSWF()`.*

En la función `sendMessage()` de JavaScript se muestra el modo de pasar un parámetro a una función de ActionScript `sendMessage()` es la función llamada cuando el usuario pulsa el botón Send de la página HTML).

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

El método `newMessage()` de ActionScript espera un parámetro, de modo que la variable `message` de JavaScript se pasa a ActionScript usándola como parámetro en la llamada al método `newMessage()` en el código JavaScript.

Detección del tipo de navegador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

A causa de las diferencias en el modo en el que los navegadores acceden al contenido, es importante usar siempre JavaScript para detectar qué navegador está utilizando el usuario y para acceder a la película de acuerdo con su sintaxis específica, empleando el objeto window o document, según se muestra en la función `getSWF()` de JavaScript en este ejemplo:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Si el script no detecta el tipo de navegador del usuario, puede producirse un comportamiento inesperado al reproducir archivos SWF en un contenedor HTML.

Capítulo 48: Validación de la firma XML en AIR

Adobe AIR 1.5 y posterior

Utilice las clases en la API XMLSignatureValidator de Adobe® AIR® para validar firmas digitales que cumplan con un subconjunto de la recomendación W3C para el procesamiento y la sintaxis de la firma XML (XML-Signature Syntax and Processing) (http://http://www.w3.org/TR/xmldsig-core/). Las firmas XML se pueden utilizar para ayudar a verificar la integridad y la identidad del firmante de los datos o información.

Las firmas XML se pueden utilizar para validar mensajes o recursos descargados por la aplicación. Por ejemplo, si la aplicación proporciona servicios en función de una suscripción, se pueden encapsular los términos de suscripción en un documento XML firmado. Si alguien intentara modificar el documento de suscripción, se produciría un error de validación.

También se puede utilizar una firma XML para ayudar a validar los recursos descargados utilizados por la aplicación, mediante la inclusión de un manifiesto firmado que contenga resúmenes de esos recursos. La aplicación podría verificar que los recursos no se han modificado mediante la comparación del resumen en el archivo firmado con un resumen calculado a partir de los bytes cargados. Esto resulta especialmente útil cuando el recurso descargado es un archivo SWF u otro contenido activo que se desea ejecutar en el entorno limitado de seguridad de la aplicación.

Aspectos básicos de la validación de firmas XML

Adobe AIR 1.5 y posterior

Para ver una explicación rápida y ejemplos de código de la validación de firmas XML, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Creating and validating XML signatures (Creación y validación de firmas XML, en inglés) (Flex)
- Creating and validating XML signatures (Creación y validación de firmas XML, en inglés) (Flash)

Adobe® AIR® proporciona la clase XMLSignatureValidator y la interfaz IURIDereferencer para validar firmas XML. La sintaxis XML aceptada por la clase XMLSignatureValidator es un subconjunto de la recomendación W3C para el procesamiento y la sintaxis de la firma XML (XML Signature Syntax and Processing). (Debido a que solo se admite un subconjunto de la recomendación, no todas las firmas legales se pueden validar.) AIR no proporciona ninguna API para crear firmas XML.

Clases de validación de firma XML

Adobe AIR 1.5 y posterior

La API de validación de firma XML incluye las siguientes clases:

Paquete	Clases
flash.security	• XMLSignatureValidator • IURIDereferencer (interfaz) Las constantes de la cadena XMLSignatureValidator se definen en las siguientes clases: • ReferencesValidationSetting • RevocationCheckSettings • SignatureStatus • SignerTrustSettings
flash.events	• Evento • ErrorEvent

Uso de las clases de validación de firma XML

Adobe AIR 1.5 y posterior

Para utilizar la clase XMLSignatureValidator para validar una firma XML, se debe realizar lo siguiente:

- Cree un objeto XMLSignatureValidator.
- Proporcione una implementación de la interfaz IURIDereferencer. El objeto XMLSignatureValidator llama al método `dereference()` de IURIDereferencer, transmitiendo el identificador URI para cada referencia de una firma. El método `dereference()` debe resolver el URI y devolver los datos a los que se hace referencia (que pueden estar en el mismo documento como firma o encontrarse en un recurso externo).
- Defina la configuración de la validación de referencia, comprobación de revocación y confianza de certificado del objeto XMLSignatureValidator según sea conveniente para su aplicación.
- Añada detectores de eventos para los eventos `complete` y `error`.
- Llame al método `verify()`, transmitiendo la firma para validar.
- Controle los eventos `complete` y `error` e interprete los resultados.

El siguiente ejemplo implementa una función `validate()` que verifica la validez de una firma XML. Las propiedades XMLSignatureValidator se establecen de tal modo que el certificado de firma debe estar en el almacén de confianza del sistema, o bien, encadenarse a un certificado del almacén de confianza. En el ejemplo también se supone que existe una clase IURIDereferencer adecuada denominada *XMLDereferencer*.

```
private function validate( xmlSignature:XML ):void
{
    var verifier:XMLSignatureValidator = new XMLSignatureValidator();
    verifier.addEventListener(Event.COMPLETE, verificationComplete);
    verifier.addEventListener(ErrorEvent.ERROR, verificationError);
    try
    {
        verifier.uriDereferencer = new XMLDereferencer();

        verifier.referencesValidationSetting =
            ReferencesValidationSetting.VALID_IDENTITY;
        verifier.revocationCheckSetting = RevocationCheckSettings.BEST_EFFORT;
        verifier.useSystemTrustStore = true;

        //Verify the signature
        verifier.verify( xmlSignature );
    }
    catch (e:Error)
        {
            trace("Verification error.\n" + e);
        }
}

//Trace verification results
private function verificationComplete(event:Event):void

    var signature:XMLSignatureValidator = event.target as XMLSignatureValidator;
    trace("Signature status: " + signature.validityStatus + "\n");
    trace("  Digest status: " + signature.digestStatus + "\n");
    trace("  Identity status: " + signature.identityStatus + "\n");
    trace("  Reference status: " + signature.referencesStatus + "\n");
}

private function verificationError(event:ErrorEvent):void
{
    trace("Verification error.\n" + event.text);
}
```

Proceso de validación de firma XML

Adobe AIR 1.5 y posterior

Cuando se llama al método `verify()` de XMLSignatureValidator, AIR realiza los siguientes pasos:

- El motor de ejecución verifica la integridad criptográfica de la firma utilizando la clave pública del certificado de firma.
- El motor de ejecución establece la integridad criptográfica, identidad y veracidad del certificado en función de la configuración actual del objeto XMLSignatureValidator.

  La confianza en el certificado de firma es fundamental para la integridad del proceso de validación. La validación de la firma se lleva a cabo mediante un proceso criptográfico bien definido, pero la veracidad del certificado de firma es un criterio que no se puede adoptar mediante algoritmos.

  En general, existen tres formas de decidir si un certificado es de confianza:

  - Confiar en las entidades emisoras de certificados y el almacén de confianza del sistema operativo.

  - Obtener, directamente del firmante, una copia del certificado, otro certificado que sirva como delimitador de confianza para el certificado, o bien, información suficiente para identificar de forma fiable el certificado como, por ejemplo, la clave pública.
  - Preguntar al usuario final de la aplicación si confía en el certificado. Esta consulta no es válida en certificados con firma automática, ya que la información de identificación del certificado no es fiable de forma inherente.
- El motor de ejecución verifica la integridad criptográfica de los datos firmados.

  La información firmada se verifica con la ayuda de la implementación de IURIDereferencer. Para cada referencia del documento de la firma, se llama al método `dereference()` de la implementación de IURIDereferencer. Los datos que devuelve el método `dereference()` se utilizan para calcular el resumen de referencia. El valor de resumen se compara con el resumen registrado en el documento de la firma. Si los resúmenes coinciden, la información no se ha modificado desde que se firmó.

  Una consideración importante al confiar en los resultados de la validación de una firma XML es que únicamente lo que se firma es seguro. Tomemos como ejemplo un manifiesto firmado que incluye los archivos en un paquete. Cuando XMLSignatureValidator verifica la firma, únicamente comprueba si el propio manifiesto está sin modificar. Los datos de los archivos no están firmados, por lo que la firma validará cuando los archivos a los que se hace referencia en el manifiesto se modifiquen o se eliminen.

  Nota: *para verificar archivos en este manifiesto, puede calcular el resumen de los datos del archivo (con el mismo algoritmo hash utilizado en el manifiesto) y comparar el resultado con el resumen almacenado en el manifiesto firmado. En algunos casos, también se debe comprobar la presencia de archivos adicionales.*

Interpretación de los resultados de validación

Adobe AIR 1.5 y posterior

Los resultados de validación se notifican mediante las propiedades de estado del objeto XMLSignatureValidator. Estas propiedades se pueden leer una vez que el objeto validador distribuya el evento*complete*. Entre las cuatro propiedades de estado se incluyen: `validityStatus`, `digestStatus`, `identityStatus` y `referencesStatus`.

Propiedad validityStatus

Adobe AIR 1.5 y posterior

La propiedad `validityStatus` notifica la validez general de la firma. `validityStatus` depende del estado de las otras tres propiedades y puede disponer de los siguientes valores:

- `valid`: si las propiedades `digestStatus`, `identityStatus` y `referencesStatus` son todas `válidas`.
- `invalid`: si una o varias de las propiedades de estado individuales son `no válidas`.
- `unknown`: si una o varias de las propiedades de estado individuales son `unknown` y ningún estado individual es `invalid`.

Propiedad digestStatus

Adobe AIR 1.5 y posterior

La propiedad `digestStatus` indica los resultados de la verificación criptográfica del resumen del mensaje. La propiedad `digestStatus` puede tener uno de los siguientes valores:

- `valid`: si el propio documento no se ha modificado desde la firma.
- `invalid`: si el documento de firma se ha modificado o presenta un formato incorrecto.

- `unknown`: si el método `verify()` no se ha completado sin error.

Propiedad identityStatus

Adobe AIR 1.5 y posterior

La propiedad `identityStatus` notifica el estado del certificado de firma. El valor de esta propiedad depende de varios factores, entre los que se incluyen:

- La integridad criptográfica del certificado.
- Si el certificado ha caducado o se ha revocado.
- Si el certificado es de confianza en el equipo actual.
- El estado del objeto XMLSignatureValidator (por ejemplo, si los certificados adicionales se han agregado para crear la cadena de confianza, si dichos certificados son de confianza y los valores de las propiedades `useSystemTrustStore` y `revocationCheckSettings`).

La propiedad `identityStatus` puede contar con los siguientes valores:

- `valid`: para considerarse válida, el certificado de firma debe cumplir las siguientes condiciones:
  - El certificado no debe presentar modificaciones.
  - El certificado de firma no debe haber caducado ni haber sido revocado, excepto cuando una marca de hora válida está presente en la firma. Si la firma incluye una marca de hora, el certificado se considerará válido si ya lo era en el momento en que se firmó el documento. (El certificado utilizado por el servicio de marca de hora para firmar la marca de hora debe encadenarse con un certificado raíz de confianza en el equipo del usuario.)
  - El certificado es de confianza. Se puede confiar en el certificado si este se encuentra en el almacén de confianza del sistema o si se encadena a otro certificado del almacén y la propiedad `useSystemTrustStore` se establece en true. También puede designar un certificado como de confianza con el método `addCertificate()` del objeto XMLSignatureValidator.
  - De hecho, el certificado es el certificado de firma.
- `invalid`: el certificado ha caducado o se ha revocado y no existe ninguna marca de hora que pruebe la validez en el momento de la firma, o bien, el certificado se ha modificado.
- `unknown`: si el certificado no es válido y tampoco es de confianza. Por ejemplo, los certificados con firma automática se notificarán como `unknown` (a no ser que sean de confianza de forma explícita). `identityStatus` también se notifica como `unknown` si el método `verify()` no se ha completado sin error o si la identidad no se ha comprobado porque el resumen de la firma no es válido.

Propiedad referencesStatus

Adobe AIR 1.5 y posterior

La propiedad `referencesStatus` indica la integridad criptográfica de las referencias en el elemento SignedData de la firma.

- `valid`: si el resumen calculado de todas las referencias de la firma coincide con el resumen correspondiente registrado en la firma XML. Un estado `valid` indica que los datos firmados no se han modificado.
- `invalid`: si algún resumen calculado no coincide con el resumen correspondiente en la firma.
- `unknown`: si los resúmenes de referencia no se han comprobado. Las referencias no se comprueban si el resumen de firma general es `invalid` o el certificado de firma no es válido. Si `identityStatus` es `unknown`, las referencias solo se comprueban cuando `referencesValidationSetting` es `validOrUnknown`.

Firmas XML

Adobe AIR 1.5 y posterior

Una firma XML es una firma digital representada mediante sintaxis XML. Los datos de una firma XML se pueden utilizar para validar la información firmada que no se haya modificado desde la firma. Asimismo, cuando una entidad emisora de certificados de confianza ha proporcionado el certificado de firma, la identidad del firmante se puede verificar a través de la infraestructura de clave pública.

Una firma XML se puede aplicar a cualquier tipo de datos digitales (en formato binario o XML). Las firmas XML se suelen emplear en los siguientes escenarios:

- Comprobación de la modificación de recursos descargados o externos.
- Verificación para conocer si los mensajes proceden de un origen seguro.
- Validación de la licencia de la aplicación o de los privilegios de suscripción.

Sintaxis admitida de firma XML

Adobe AIR 1.5 y posterior

AIR admite los siguientes elementos de la recomendación W3C para el procesamiento y la sintaxis de firmas XML:

- Todos los elementos de la sintaxis de firma fundamentales (sección 4 del documento de la recomendación W3C), excepto el elemento KeyInfo que no se admite por completo.
- El elemento KeyInfo solo debe contener un elemento X509Data.
- Un elemento X509Data únicamente debe incluir un elemento X509Certificate.
- Método de resumen SHA256.
- Algoritmo de firma RSA-SHA1 (PKCS1).
- El método de canónico "Canonical XML without comments" (XML canónico sin comentarios) y la transformación.
- La transformación de firma protegida.
- marcas de hora

El siguiente documento muestra una firma XML estándar (la mayor parte de los datos criptográficos se han eliminado para simplificar el ejemplo):

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="URI_to_signed_data">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/></Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>uoo...vY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>Ked...w==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>i7d...w==</X509Certificate>
        </X509Data>
    </KeyInfo>
</Signature>
```

Los elementos fundamentales de una firma son los siguientes:

- SignedInfo: contiene referencias a los datos firmados y los valores de resumen calculados en el momento de la firma. Los propios datos firmados se pueden incluir en el mismo documento que la firma XML o pueden ser externos.
- SignatureValue: contiene un resumen del elemento SignedInfo cifrado con la clave privada del firmante.
- KeyInfo: incluye el certificado de firma, así como los certificados adicionales necesarios para establecer la cadena de confianza. Se debe tener en cuenta que aunque el elemento KeyInfo es opcional desde el punto de vista técnico, AIR no puede validar la firma si no se incluye.

Existen tres tipos generales de firmas XML:

- Protegidas: la firma se inserta dentro de los datos XML que se están firmando.
- Con protección: los datos XML firmados se incluyen en un elemento Object en el elemento Signature.
- Separadas: los datos firmados son externos a la firma XML. La información firmada puede estar en un archivo externo. También puede estar en el mismo documento XML que la firma y no solo como elemento principal o secundario del elemento Signature.

Las firmas XML utilizan los URI para hacer referencia a los datos firmados. Las aplicaciones de validación y firma deben usar las mismas convenciones para resolver estos URI. Cuando se utiliza la clase XMLSignatureValidator, se debe proporcionar una implementación de la interfaz IURIDereferencer. Esta implementación es responsable de resolver el URI y devolver los datos firmados como objeto ByteArray. El objeto ByteArray devuelto se resume utilizando el mismo algoritmo que generó el resumen en la firma.

Certificados y confianza

Adobe AIR 1.5 y posterior

Un certificado consta de una clave pública, información de identificación y posiblemente uno o varios certificados que pertenecen a la entidad emisora de certificados.

Existen dos modos de establecer la confianza en un certificado. Se puede obtener una copia del certificado directamente del firmante; por ejemplo, en soportes físicos o a través de una transmisión digital segura como, por ejemplo, una transacción SSL. También se puede confiar en una entidad emisora de certificados para determinar si el certificado de firma es de confianza.

Para confiar en una entidad de este tipo, el certificado debe proceder de una autoridad de confianza para el equipo en el que se valida la firma. La mayor parte de los fabricantes de sistemas operativos sitúan los certificados raíz de una serie de entidades emisoras de certificados en el almacén de confianza del sistema operativo. Los usuarios también pueden añadir y eliminar certificados del almacén.

Aunque un certificado proceda de una entidad de confianza, aún debe decidir si el certificado pertenece a un usuario de confianza. En muchos casos esta decisión pertenece al usuario final. Por ejemplo, cuando se instala una aplicación de AIR, el archivo de instalación de AIR muestra información de identificación del certificado del editor cuando solicita al usuario que confirme si desea instalar la aplicación. En otros casos, se puede comparar la clave pública u otra información del certificado con una lista de claves aceptables. (Esta lista se debe asegurar, quizás mediante su propia firma, o bien, almacenándola en el almacén local cifrado de AIR, con el fin de que no se pueda manipular.)

Nota: *aunque puede optar por confiar en el certificado de firma sin verificación independiente (como cuando se trata de una firma automática), de este modo no se obtienen muchas garantías de nada al verificar la firma. Si no se conoce quién creó una firma, la garantía de que esta no se ha manipulado no es una opción muy eficaz, si es que esta opción sirve de algo. La firma podría ser una falsificación firmada con validez.*

Caducidad y revocación de certificados

Adobe AIR 1.5 y posterior

Todos los certificados caducan. Los certificados también se pueden revocar mediante la entidad emisora de certificados si, por ejemplo, la clave privada relacionada con el certificado se ha robado o pierde su carácter de privacidad. Si una firma aparece en un certificado caducado o revocado, la firma se notificará como no válida a no ser que se haya incluido una marca de hora como parte de la firma. Si se incluye una marca de hora, la clase XMLSignatureValidator validará la firma si el certificado era válido en el momento de la firma.

Una marca de hora es un mensaje digital firmado procedente de un servicio de marca de hora que certifica que los datos se firmaron en una fecha y horas concretas. Las marcas de hora provienen de entidades emisoras y se firman mediante el propio certificado de la entidad emisora de marcas de hora. El certificado de la entidad emisora incorporado a la marca de hora debe ser de confianza en el equipo actual para que la marca de hora se considere válida. XMLSignatureValidator no proporciona ninguna API para designar un certificado diferente para su uso en la validación de marcas de hora.

Implementación de la interfaz IURIDereferencer

Adobe AIR 1.5 y posterior

Para validar una firma XML, debe proporcionar una implementación de la interfaz IURIDereferencer. La implementación es responsable de resolver los URI en los elementos Reference de un documento de firma XML y devolver los datos para que se pueda calcular el resumen. El resumen calculado se compara con el resumen en la firma para determinar si los datos a los que se hace referencia se han modificado desde que se creó la firma.

Nota: *las aplicaciones de AIR basadas en HTML deben importar una biblioteca SWF que contenga una implementación de ActionScript para poder validar firmas XML. La interfaz IURIDereferencer no se puede implementar en JavaScript.*

La interfaz IURIDerefencer tiene un solo método, `dereference(uri:String)`, que debe implementarse. El objeto XMLSignatureValidator llama a este método para todas las referencias de la firma. El método debe devolver los datos en un objeto ByteArray.

En la mayoría de los casos, también será necesario agregar propiedades o métodos que permitan al objeto dereferencer localizar los datos a los que se hace referencia. Por ejemplo, si los datos firmados se ubican en el mismo documento que la firma, puede añadir una variable miembro que proporcione una referencia al documento XML. El método `dereference()` puede utilizar después esta variable, junto con el URI, para localizar los datos a los que se hace referencia. Del mismo modo, si los datos firmados se encuentran en un directorio del sistema de archivos local, el método `dereference()` puede necesitar una propiedad que proporcione la ruta al directorio para poder resolver los archivos a los que se hace referencia.

XMLSignatureValidator confía por completo en el objeto dereferencer para interpretar las cadenas de URI. Las reglas estándar para eliminar las referencias de los URI se incluyen en la sección 4.3.3 de la recomendación del W3C para el procesamiento y la sintaxis de la firma XML.

Eliminación de referencias de los URI en firmas protegidas

Adobe AIR 1.5 y posterior

Cuando se genera una firma XML protegida, sus elementos se insertan en los datos firmados. Por ejemplo,si se ha firmado el siguiente mensaje utilizando una estructura de firma protegida:

```
<message>
    <data>...</data>
</message>
```

El documento firmado resultante presentará el siguiente aspecto:

```
<message>
    <data>...</data>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>yv6...Z0Y=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>cCY...LQ==</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509Certificate>MII...4e</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</message>
```

Se debe tener en cuenta que la firma contiene un solo elemento Reference con una cadena vacía como su URI. Una cadena vacía en este contexto hace referencia a la raíz del documento.

También se debe observar que el algoritmo de transformación especifica que se ha aplicado una transformación de firma protegida. Cuando se aplica una transformación de este tipo, XMLSignatureValidator elimina automáticamente la firma del documento antes de calcular el resumen. Esto significa que el objeto dereferencer no necesita eliminar el elemento Signature al devolver los datos.

El siguiente ejemplo muestra un elemento dereferencer para firmas protegidas:

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class EnvelopedDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var signedMessage:XML;

        public function EnvelopedDereferencer( signedMessage:XML )
        {
            this.signedMessage = signedMessage;
        }

        public function dereference( uri:String ):IDataInput
        {
            try
            {
                if( uri.length != 0 )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var data:ByteArray = new ByteArray();
                data.writeUTFBytes( signedMessage.toXMLString() );
                data.position = 0;
            }
            catch (e:Error)
                {
                var error:ErrorEvent =
                    new ErrorEvent("Ref error " + uri + " ", false, false, e.message);
                this.dispatchEvent(error);
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }
        }
    }
}
```

Esta clase dereferencer utiliza una función constructora con un parámetro, `signedMessage`, para que el documento de firma protegida esté disponible en el método `dereference()`. Debido a que la referencia en una firma protegida siempre hace referencia a la raíz de los datos firmados, el método `dereferencer()` escribe el documento en un conjunto de bytes y lo devuelve.

Eliminación de referencias de los URI en firmas con protección y separadas

Adobe AIR 1.5 y posterior

Si los datos firmados se ubican en el mismo documento que la propia firma, los URI de las referencias suelen utilizar la sintaxis XPath o XPointer para hacer referencia a los elementos que están firmados. La recomendación W3C para el procesamiento y la sintaxis de la firma XML únicamente recomienda esta sintaxis, por lo que la implementación se debe basar en las firmas que se esperan encontrar (y añadir una comprobación de errores suficiente para administrar correctamente la sintaxis no admitida).

La firma de una aplicación de AIR es un ejemplo de una firma con protección. Los archivos de la aplicación se incluyen en un elemento Manifest. Al elemento Manifest se hace referencia en el atributo URI Reference utilizando la cadena "#PackageContents", que hace referencia al Id. del elemento Manifest:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="PackageSignature">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/TR/xmldsig-core#rsa-sha1"/>
        <Reference URI="#PackageContents">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>ZMGqQdaRKQc1HirIRsDpeBDlaElS+pPotdziIAyAYDk=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue Id="PackageSignatureValue">cQK...7Zg==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>MII...T4e</X509Certificate>
        </X509Data>
    </KeyInfo>
    <Object>
    <Manifest Id="PackageContents">
        <Reference URI="mimetype">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>0/oCb84THKMagtI0Dy0KogEu92TegdesqRr/clXct1c=</DigestValue>
        </Reference>
        <Reference URI="META-INF/AIR/application.xml">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>P9MqtqSdqcqnFgeoHCJysLQu4PmbUW2JdAnc1WLq8h4=</DigestValue>
        </Reference>
        <Reference URI="XMLSignatureValidation.swf">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>OliRHRAgc9qt3Dk0m0Bi53Ur5ur3fAweIFwju74rFgE=</DigestValue>
        </Reference>
    </Manifest>
</Object>
</Signature>
```

Una clase dereferencer para validar esta firma debe tomar la cadena URI que contiene `"#PackageContents"` del elemento Reference y devolver el elemento Manifest en un objeto ByteArray. El símbolo "#" hace referencia al valor de un atributo Id. del elemento.

El siguiente ejemplo implementa un elemento dereferencer para validar firmas de la aplicación de AIR. La implementación constituye un proceso sencillo al confiar en la estructura conocida de una firma de AIR. Un objeto dereferencer de uso general podría resultar mucho más complejo.

```
package
{
    import flash.events.ErrorEvent;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class AIRSignatureDereferencer implements IURIDereferencer {
        private const XML_SIG_NS:Namespace =
            new Namespace( "http://www.w3.org/2000/09/xmldsig#" );
        private var airSignature:XML;

        public function AIRSignatureDereferencer( airSignature:XML ) {
            this.airSignature = airSignature;
        }

        public function dereference( uri:String ):IDataInput {
            var data:ByteArray = null;
            try
            {
                if( uri != "#PackageContents" )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var manifest:XMLList =
                    airSignature.XML_SIG_NS::Object.XML_SIG_NS::Manifest;
                data = new ByteArray();
                data.writeUTFBytes( manifest.toXMLString());
                data.position = 0;
            }
            catch (e:Error)
            {
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }

        }
    }
}
```

Cuando se verifica este tipo de firma, únicamente se validan los datos del elemento Manifest. Los archivos reales del paquete no se comprueban. Para comprobar modificaciones en los archivos del paquete, debe leer los archivos, calcular el resumen SHA256 y comparar el resultado con el resumen registrado en el manifiesto. XMLSignatureValidator no comprueba automáticamente estas referencias secundarias.

Nota: *este ejemplo se incluye para mostrar el proceso de validación de firma. No resulta muy útil en una aplicación de AIR que valida su propia firma. Si la aplicación ya se ha manipulado, el agente de las alteraciones podría eliminar simplemente la comprobación de validación.*

Cálculo de valores de resumen para recursos externos

Adobe AIR 1.5 y posterior

AIR no incluye funciones incorporadas para calcular resúmenes SHA256, pero el SDK de Flex incluye una clase de utilidad SHA256. El SDK también incluye la clase de utilidad del codificador Base64 que resulta útil para comparar el resumen calculado con el resumen almacenado en una firma.

La siguiente función de ejemplo lee y valida los archivos en un manifiesto de un paquete de AIR:

```
import mx.utils.Base64Encoder;
import mx.utils.SHA256;

private function verifyManifest( sigFile:File, manifest:XML ):Boolean
{
    var result:Boolean = true;
    var message:String = '';
    var nameSpace:Namespace = manifest.namespace();

    if( manifest.nameSpace::Reference.length() <= 0 )
    {
        result = false;
        message = "Nothing to validate.";
    }
    for each (var reference:XML in manifest.nameSpace::Reference)
    {
        var file:File = sigFile.parent.parent.resolvePath( reference.@URI );
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var fileData:ByteArray = new ByteArray();
        stream.readBytes( fileData, 0, stream.bytesAvailable );

        var digestHex:String = SHA256.computeDigest( fileData );
        //Convert hexidecimal string to byte array
        var digest:ByteArray = new ByteArray();
        for( var c:int = 0; c < digestHex.length; c += 2 ){
            var byteChar:String = digestHex.charAt(c) + digestHex.charAt(c+1);
            digest.writeByte( parseInt( byteChar, 16 ));
        }
        digest.position = 0;
```

```
        var base64Encoder:Base64Encoder = new Base64Encoder();
        base64Encoder.insertNewLines = false;
        base64Encoder.encodeBytes( digest, 0, digest.bytesAvailable );
        var digestBase64:String = base64Encoder.toString();
        if( digestBase64 == reference.nameSpace::DigestValue )
        {
            result = result && true;
            message += "   " + reference.@URI + " verified.\n";
        }
        else
        {
            result = false;
            message += " ---- " + reference.@URI + " has been modified!\n";
        }
        base64Encoder.reset();
    }
    trace( message );
    return result;
}
```

La función recorre todas las referencias en el elemento Manifest. Para cada referencia, se calcula el resumen SHA256, codificado en formato base64, y se compara con el resumen en el manifiesto. Los URI de un paquete de AIR hacen referencia a las rutas relativas al directorio de la aplicación. Las rutas se resuelven en función de la ubicación del archivo de firma, que siempre está en el subdirectorio META-INF del directorio de la aplicación. Se debe tener en cuenta que la clase SHA256 de Flex devuelve el resumen como una cadena de caracteres hexadecimales. Esta cadena se debe convertir en un objeto ByteArray que contenga los bytes representados mediante la cadena hexadecimal.

Para utilizar las clases mx.utils.SHA256 y Base64Encoder en Flash CS4, puede ubicar y copiar estas clases en el directorio de desarrollo de la aplicación o compilar un archivo SWF de biblioteca que incluya las clases utilizando el SDK de Flex.

Eliminación de referencia de los URI en firmas separadas que hacen referencia a datos externos

Adobe AIR 1.5 y posterior

Cuando un URI hace referencia a un recurso externo, se debe acceder a los datos y estos deben cargarse en un objeto ByteArray. Si el URI contiene una dirección URL absoluta, simplemente se debe leer un archivo o solicitar una URL. Probablemente el caso más común es que si el URI se contiene en una ruta relativa, la implementación de IURIDereferencer debe incluir un modo de resolver las rutas a los archivos firmados.

El siguiente ejemplo utiliza un objeto File inicializado cuando la instancia de dereferencer se crea como base para resolver los archivos firmados.

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.filesystem.File;
    import flash.filesystem.FileMode;
    import flash.filesystem.FileStream;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;
    public class RelativeFileDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var base:File;

        public function RelativeFileDereferencer( base:File )
        {
            this.base = base;
        }

        public function dereference( uri:String ):IDataInput
        {
            var data:ByteArray = null;
            try{
                var referent:File = this.base.resolvePath( uri );
                var refStream:FileStream = new FileStream();
                data = new ByteArray();
                refStream.open( referent, FileMode.READ );

                refStream.readBytes( data, 0, data.bytesAvailable );

            } catch ( e:Error ) {
                data = null;
                throw new Error("Reference not resolvable: " + referent.nativePath + ", " +
e.message );
            } finally {
                return data;
            }
        }
    }
}
```

La función `dereference()` simplemente ubica el archivo al que se dirige el URI de referencia, carga el contenido del archivo en un conjunto de bytes y devuelve el objeto ByteArray.

Nota: *antes de validar las referencias externas remotas, tenga en cuenta si la aplicación podría ser vulnerable a un ataque de tipo "phone home" o ataque similar por parte de un documento de firma creado de forma malintencionada.*

Capítulo 49: Entorno del sistema del cliente

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En este capítulo se explica la manera de interactuar con el sistema del usuario. Muestra la manera de determinar qué funciones se admiten y cómo crear aplicaciones multilingües mediante el editor de método de entrada (IME) instalado en el sistema del usuario (si está disponible). También muestra usos típicos de los dominios de aplicación.

Más temas de ayuda

flash.system.System

flash.system.Capabilities

Fundamentos del entorno del sistema del cliente

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Al crear aplicaciones de más avanzadas puede que sea necesario conocer los detalles sobre los sistemas operativos de los usuarios y acceder a funciones de dichos sistemas. El paquete flash.system contiene un conjunto de clases que permiten acceder a funcionalidades de nivel del sistema, como las siguientes:

- Determinar la aplicación y el dominio de seguridad en el que se está ejecutando código SWF
- Determinar las capacidades de la instancia del motor de ejecución de Flash del usuario (como Flash® Player o Adobe® AIR™), como el tamaño de la pantalla (resolución) y si determinadas funcionalidades, tales como el audio MP3, están disponibles
- Generar sitios multilingües con el IME
- Interactuar con el contenedor del motor de ejecución de Flash (que puede ser una página HTML o una aplicación contenedora).
- Guardar información en el portapapeles del usuario

El paquete flash.system también incluye las clases IMEConversionMode y SecurityPanel. Estas clases contienen constantes estáticas que se utilizan con las clases IME y Security, respectivamente.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes:

Sistema operativo Programa principal que se ejecuta en un equipo, con el que se ejecutan todas las demás aplicaciones, como Microsoft Windows, Mac OS X o Linux®.

Clipboard Contenedor del sistema operativo para texto o elementos que se copian o cortan, y desde el que se pegan elementos en aplicaciones.

Dominio de aplicación Mecanismo para separar clases utilizadas en distintos archivos SWF de forma que, si los archivos SWF incluyen distintas clases con el mismo nombre, no se sobrescriban unas a otras.

IME (editor de método de entrada) Programa (o herramienta del sistema operativo) que se utiliza para introducir caracteres o símbolos complejos mediante un teclado estándar.

Sistema del cliente En términos de programación, un cliente es la parte de una aplicación (o una aplicación completa) que se ejecuta en un equipo individual y es utilizada por un solo usuario. El sistema del cliente es el sistema operativo subyacente en el equipo del usuario.

Uso de la clase System

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase System contiene métodos y propiedades que permiten interactuar con el sistema operativo del usuario y obtener el consumo de memoria actual del motor de ejecución. Los métodos y propiedades de la clase System también permiten detectar eventos `imeComposition`, ordenar al motor de ejecución de que cargue archivos de texto externos con la página de códigos actual del usuario o como Unicode, o establecer el contenido del portapapeles del usuario.

Obtención de datos sobre el sistema del usuario en tiempo de ejecución

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede comprobar el valor de la propiedad `System.totalMemory` para determinar la cantidad de memoria (en bytes) utilizada actualmente por el motor de ejecución de Esta propiedad permite controlar el consumo de memoria y optimizar las aplicaciones a partir de los cambios de nivel de la memoria. Por ejemplo, si un efecto visual específico provoca un gran aumento de consumo de memoria, es posible que se desee modificar el efecto o eliminarlo del todo.

La propiedad `System.ime` es una referencia al editor de método de entrada (IME) instalado actualmente. Esta propiedad permite detectar eventos `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) mediante el método `addEventListener()`.

La tercera propiedad de la clase System es `useCodePage`. Si `useCodePage` se establece en `true`, el motor de ejecución de utiliza la página de código tradicional del sistema operativo para cargar archivos de texto externos. Si se establece esta propiedad en `false`, se indica al motor de ejecución de que debe interpretar el archivo externo como Unicode.

Si se establece `System.useCodePage` en el valor `true`, hay que recordar que la página de códigos tradicional del sistema operativo en el que se ejecuta el reproductor debe incluir los caracteres utilizados en el archivo de texto externo para que se muestre el texto. Por ejemplo, si se carga un archivo de texto externo que contiene caracteres chinos, dichos caracteres no se visualizarán en un sistema que utilice la página de códigos de Windows en inglés, ya que dicha página de códigos no contiene caracteres chinos.

Para garantizar que los usuarios de todas las plataformas puedan ver archivos de texto externos que se utilizan en su aplicación, hay que codificar todos los archivos de texto externos como Unicode y establecer `System.useCodePage` como `false` de forma predeterminada. De este modo, el motor de ejecución de interpreta el texto como Unicode.

Copia de texto al portapapeles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase System incluye un método denominado `setClipboard()` que permite al motor de ejecución de Flash establecer el contenido del portapapeles del usuario con una cadena especificada. Por razones de seguridad, no hay un método `Security.getClipboard()`, ya que este método podría permitir a sitios malintencionados el acceso a los últimos datos copiados al portapapeles del usuario.

El código siguiente ilustra cómo se puede copiar un mensaje de error al portapapeles del usuario cuando se produce un error de seguridad. El mensaje de error puede ser útil si el usuario desea notificar un posible error con una aplicación.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Flash Player 10 y AIR 1.0

La clase Clipboard se puede utilizar para leer y escribir datos del portapapeles como respuesta a un evento de usuario. En AIR, no se requiere ningún evento de usuario para que el código que se ejecuta en el entorno limitado de la aplicación pueda acceder al portapeles.

Uso de la clase Capabilities

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Capabilities permite a los desarrolladores determinar el entorno en el que se está ejecutando una aplicación. Se pueden utilizar diversas propiedades de la clase Capabilities para averiguar la resolución del sistema del usuario, si dicho sistema admite software de accesibilidad y el lenguaje del sistema operativo del usuario, así como la versión del motor de ejecución de Flash instalada en ese momento.

Se pueden comprobar los valores de las propiedades de la clase Capabilities para personalizar la aplicación de forma que funcione de forma óptima con el entorno específico del usuario. Por ejemplo, si se comprueban los valores de las propiedades `Capabilities.screenResolutionX` y `Capabilities.screenResolutionY`, se puede determinar la resolución de pantalla que utiliza el sistema del usuario y decidir qué tamaño de vídeo es el más apropiado. También se puede comprobar el valor de la propiedad `Capabilities.hasMP3` para comprobar si el sistema del usuario admite la reproducción de MP3 antes de intentar cargar un archivo MP3 externo.

El código siguiente utiliza una expresión regular para analizar la versión del motor de ejecución de Flash que utiliza el cliente:

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

Si se desea enviar las características del sistema del usuario a un script de servidor de forma que se pueda almacenar la información en una base de datos, se puede utilizar el siguiente código ActionScript:

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

Ejemplo de Capabilities: Detección de las características del sistema

Flash Player 9 y posterior

En el ejemplo CapabilitiesExplorer se muestra la manera de utilizar la clase flash.system.Capabilities para determinar qué funciones admite la versión del motor de ejecución de Flash del usuario. Este ejemplo ilustra las técnicas siguientes:

- Detectar las características de la versión del motor de ejecución de Flash del usuario mediante la clase Capabilities.
- Utilizar la clase ExternalInterface para detectar la configuración del navegador del usuario

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación CapabilitiesExplorer se encuentran en la carpeta Samples/CapabilitiesExplorer. La aplicación consta de los siguientes archivos:

Archivo	Descripción
CapabilitiesExplorer.fla o CapabilitiesExplorer.mxml	El archivo de aplicación principal en Flash (FLA) o Flex (MXML).
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	La clase que proporciona la funcionalidad principal de la aplicación, como añadir las características del sistema a un conjunto, ordenar los elementos y utilizar la clase ExternalInterface para obtener las características del navegador.
capabilities.html	Un contenedor HTML que contiene el código JavaScript necesario para comunicarse con la API externa.

Información general sobre CapabilitiesExplorer

Flash Player 9 y posterior

El archivo CapabilitiesExplorer.mxml se encarga de configurar la interfaz de usuario para la aplicación CapabilitiesExplorer. Las capacidades de la versión del motor de ejecución de Flash del usuario se mostrarán en una instancia del componente DataGrid en el escenario. También se mostrarán las características del navegador si se ejecuta la aplicación desde un contenedor HTML y la API externa está disponible.

Cuando se distribuye el evento `creationComplete` del archivo de aplicación principal, se invoca el método `initApp()`. El método `initApp()` llama al método `getCapabilities()` desde la clase com.example.programmingas3.capabilities.CapabilitiesGrabber. El código del método `initApp()` es el siguiente:

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

El método `CapabilitiesGrabber.getCapabilities()` devuelve un conjunto ordenado de las capacidades del navegador y el motor de ejecución de Flash, que después se establece en la propiedad `dataProvider` de la instancia `capabilitiesGrid` del componente DataGrid en el escenario.

Información general sobre la clase CapabilitiesGrabber

Flash Player 9 y posterior

El método estático `getCapabilities()` de la clase CapabilitiesGrabber añade cada propiedad de la clase flash.system.Capabilities a un conjunto (`capDP`). Después llama al método estático `getBrowserObjects()` de la clase CapabilitiesGrabber. El método `getBrowserObjects()` utiliza la API externa para recorrer el objeto navigator del navegador, que contiene las características del navegador. El código del método `getCapabilities()` es:

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

El método `getBrowserObjects()` devuelve un conjunto que contiene cada una de las propiedades del objeto navigator del navegador. Si este conjunto tiene una longitud de uno o más elementos, el conjunto de características del navegador (`navArr`) se añade a la matriz de características de Flash Player (`capDP`) y se ordena alfabéticamente el conjunto completo. Por último, el conjunto ordenado se devuelve al archivo de aplicación principal, que llena a continuación la cuadrícula de datos. El código del método `getBrowserObjects()` es el siguiente:

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

Si la API externa está disponible en el entorno de usuario actual, el motor de ejecución de Flash llama al método `JS_getBrowserObjects()` de JavaScript, que recorre el objeto navigator del navegador y devuelve a ActionScript una cadena de valores codificados en URL. Esta cadena se convierte en un objeto URLVariables (`itemVars`) y se añade al conjunto `itemArr`, que se devuelve al script que hizo la llamada.

Comunicación con JavaScript

Flash Player 9 y posterior

La parte final de la creación de la aplicación CapabilitiesExplorer consiste en escribir el código JavaScript necesario para recorrer cada uno de los elementos del objeto navigator del navegador y añadir un par nombre-valor a un conjunto temporal. El código del método `JS_getBrowserObjects()` de JavaScript del archivo container.html es:

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
```

```
                    // escape() function.
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
                }
            }
            // Loop over each item in the browser's screen object.
            for (var name in screen)
            {
                var value = screen[name];

                // If the current value is a number, add it to the array, otherwise
                // ignore the item.
                switch (typeof(value))
                {
                    case "number":
                        var tempStr = "screen." + name + "=" + escape(value);
                        tempArr.push(tempStr);
                        break;
                }
            }
            // Return the array as a URL-encoded string of name-value pairs.
            return tempArr.join("&");
        }
</script>
```

El código empieza creando un conjunto temporal que contendrá todos los pares nombre-valor del objeto navigator. A continuación, el objeto navigator se somete a un bucle `for..in` y se evalúa el tipo de datos del valor actual para filtrar y omitir los valores no deseados. En esta aplicación, solo estamos interesados en valores de cadena o booleanos, y los otros tipos de datos (como funciones o conjuntos) se omiten. Cada valor de tipo cadena o booleano del objeto navigator se añade al conjunto `tempArr`. A continuación, el objeto screen del navegador se somete a un bucle `for..in` y se añaden los valores numéricos encontrados al conjunto `tempArr`. Por último, el conjunto se convierte en una cadena mediante el método `Array.join()`. El conjunto usa un carácter ampersand (&) como delimitador, lo que permite a ActionScript analizar fácilmente los datos con la clase URLVariables.

Capítulo 50: Cierre e invocación de una aplicación de AIR

Adobe AIR 1.0 y posterior

En esta sección se analizan las formas en que se puede invocar una aplicación de Adobe® AIR® instalada, así- como las opciones y consideraciones para cerrar una aplicación en ejecución.

Nota: *los objetos NativeApplication, InvokeEvent y BrowserInvokeEvent solo están disponibles para el contenido SWF que se ejecuta en el entorno limitado de la aplicación de AIR. El contenido SWF que se ejecuta en el motor de ejecución de Flash Player, en el navegador o reproductor autónomo (proyector), o en una aplicación de AIR fuera del entorno limitado de la aplicación, no puede acceder a estas clases.*

Para ver una explicación rápida y ejemplos de código de la invocación y finalización de aplicaciones de AIR, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Startup Options (Opciones de inicio, en inglés)

Más temas de ayuda

flash.desktop.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

Invocación de aplicaciones

Adobe AIR 1.0 y posterior

Se invoca una aplicación se AIR cuando el usuario (o el sistema operativo):

- inicia la aplicación desde el shell del escritorio;
- utiliza la aplicación como comando en un shell de línea de comandos;
- abre un tipo de archivo para el que la aplicación es la aplicación de apertura predeterminada;
- (Mac OS X) hace clic en el icono de la aplicación en el Dock (esté ejecutándose en ese momento o no la aplicación);
- elige iniciar la aplicación desde el programa de instalación (al finalizar un nuevo proceso de instalación o después de hacer doble clic en el archivo de AIR para una aplicación ya instalada);
- inicia una actualización de una aplicación de AIR cuando la versión instalada ha señalado que está gestionando las actualizaciones por su cuenta (al incluir la declaración `<customUpdateUI>true</customUpdateUI>` en el archivo descriptor de la aplicación);
- (iOS) Recibe una notificación del servicio de notificaciones push de Apple (APN).
- Invoca la aplicación a través de una URL.

- visita una página web que contiene un logotipo o una aplicación de Flash que llama al método `com.adobe.air.AIRlaunchApplication()` especificando la información de identificación para la aplicación de AIR. (Para que la invocación desde el navegador funcione, el descriptor de la aplicación debe incluir además una declaración `<allowBrowserInvocation>true</allowBrowserInvocation>`).

Siempre que se invoca una aplicación de AIR, AIR distribuye un objeto InvokeEvent del tipo `invoke` a través del objeto NativeApplication de instancia única. Para que la aplicación tenga tiempo de inicializarse y registrar un detector de eventos, los eventos `invoke` pasan a la cola en lugar de ser desechados. En cuanto se haya registrado el detector, se entregan todos los eventos que están en la cola.

Nota: *cuando se invoca una aplicación con la función de invocación desde el navegador, el objeto NativeApplication solo distribuye un evento* `invoke` *si la aplicación no está ya ejecutándose.*

Para recibir eventos `invoke`, llame al método `addEventListener()` del objeto NativeApplication (`NativeApplication.nativeApplication)`. Cuando un detector de eventos se registra para un evento `invoke`, también recibe todos los eventos `invoke` que se produjeron antes de haberse registrado el detector. Los eventos `invoke` que están en la cola se distribuyen uno a la vez en un breve intervalo tras la devolución de la llamada a `addEventListener()`. Si se produce un nuevo evento `invoke` durante este proceso, puede que se distribuya antes de uno o varios de los eventos de la cola. Esta cola de eventos le permite controlar cualquier evento `invoke` que se haya producido antes de ejecutarse el código de inicialización. Tenga en cuenta que si se añade un detector de eventos más adelante en la ejecución (después de inicializada la aplicación), aún recibirá todos los eventos `invoke` que se hayan producido desde que se inició la aplicación.

Solo se inicia una instancia de una aplicación de AIR. Si se vuelve a invocar una aplicación que ya está en curso, AIR distribuye un nuevo evento `invoke` a la instancia que se está ejecutando. Es responsabilidad de una aplicación de AIR responder a un evento `invoke` y tomar las medidas correspondientes (por ejemplo, abrir una nueva ventana para documentos).

Un objeto `InvokeEvent` contiene los argumentos que se pasen a la aplicación, además del directorio desde el cual se invocó la aplicación. Si la aplicación se invocó debido a una asociación de tipo de archivo, los argumentos de la línea de comandos incluirán la ruta completa al archivo. Asimismo, si la aplicación se invocó a raíz de una actualización de la misma, se indica la ruta completa al archivo de actualización de AIR.

Cuando se abren varios archivos en una operación, se distribuye un solo objeto InvokeEvent en Mac OS X. Cada archivo se incluye en el conjunto `arguments`. En Windows y Linux, se distribuye un objeto InvokeEvent independiente para cada archivo.

La aplicación puede controlar eventos `invoke` registrando un detector con su objeto NativeApplication:

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
```

Y mediante la definición de un detector de eventos:

```
 var arguments:Array;
var currentDir:File;
public function onInvokeEvent(invocation:InvokeEvent):void {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

Captura de argumentos de la línea de comandos

Adobe AIR 1.0 y posterior

Los argumentos de la línea de comandos asociados con la invocación de una aplicación de AIR se transmiten en el objeto InvokeEvent distribuido por el objeto NativeApplication. La propiedad `arguments` de InvokeEvent contiene un conjunto de los argumentos que pasa el sistema operativo cuando se invoca una aplicación de AIR. Si los argumentos contienen rutas a archivos relacionados, normalmente se pueden resolver las rutas con la propiedad `currentDirectory`.

Los argumentos que se pasan a un programa de AIR se tratan como cadenas delimitadas por espacios en blanco, a menos que estén entre comillas dobles:

Argumentos	Array
tick tock	{tick,tock}
tick "tick tock"	{tick,tick tock}
"tick" “tock”	{tick,tock}
\"tick\" \"tock\"	{"tick","tock"}

La propiedad `currentDirectory` contiene un objeto File que representa el directorio desde el cual se inició la aplicación.

Cuando se invoca una aplicación porque se abre un archivo del tipo registrado por la aplicación, la ruta nativa al archivo se incluye como cadena en los argumentos de la línea de comandos. (La aplicación se encarga de abrir el archivo y realizarle la operación prevista). Asimismo, si una aplicación está programada para actualizarse ella misma (en lugar de depender de la interfaz de usuario de actualización de AIR estándar), la ruta nativa al archivo de AIR se incluye cuando el usuario hace doble clic en un archivo de AIR que contenga una aplicación con un ID de aplicación igual.

Se puede acceder al archivo con el método `resolve()` del objeto File `currentDirectory`:

```
 if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

También debe validar que un argumento es realmente una ruta a un archivo.

Ejemplo: Historial de eventos de invocación

Adobe AIR 1.0 y posterior

El siguiente ejemplo muestra cómo registrar detectores para el evento `invoke` y cómo se controla este evento. En el ejemplo se registran todos los eventos de invocación recibios y se muestran el directorio y los argumentos de la línea de comandos actuales.

Ejemplo de ActionScript

```
package
{
    import flash.display.Sprite;
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;
    import flash.text.TextField;

    public class InvokeEventLogExample extends Sprite
    {
        public var log:TextField;

        public function InvokeEventLogExample()
        {
            log = new TextField();
            log.x = 15;
            log.y = 15;
            log.width = 520;
            log.height = 370;
            log.background = true;

            addChild(log);

            NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvoke);
        }

        public function onInvoke(invokeEvent:InvokeEvent):void
        {
            var now:String = new Date().toTimeString();
            logEvent("Invoke event received: " + now);

            if (invokeEvent.currentDirectory != null)
            {
                logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
            }
            else
            {
```

```
                logEvent("--no directory information available--");
            }

            if (invokeEvent.arguments.length > 0)
            {
                logEvent("Arguments: " + invokeEvent.arguments.toString());
            }
            else
            {
                logEvent("--no arguments--");
            }
        }

        public function logEvent(entry:String):void
        {
            log.appendText(entry + "\n");
            trace(entry);
        }
    }
}
```

Ejemplo de Flex

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    invoke="onInvoke(event)" title="Invocation Event Log">
    <mx:Script>
    <![CDATA[
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;

    public function onInvoke(invokeEvent:InvokeEvent):void {
        var now:String = new Date().toTimeString();
        logEvent("Invoke event received: " + now);

        if (invokeEvent.currentDirectory != null){
            logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
        } else {
            logEvent("--no directory information available--");
        }

        if (invokeEvent.arguments.length > 0){
            logEvent("Arguments: " + invokeEvent.arguments.toString());
        } else {
            logEvent("--no arguments--");
        }
    }

    public function logEvent(entry:String):void {
        log.text += entry + "\n";
        trace(entry);
    }
    ]]>
    </mx:Script>
    <mx:TextArea id="log" width="100%" height="100%" editable="false"
        valueCommit="log.verticalScrollPosition=log.textHeight;"/>
</mx:WindowedApplication>
```

Invocación de una aplicación de AIR en el inicio de sesión de usuario

Adobe AIR 1.0 y posterior

Para configurar una aplicación de AIR de modo que se inicie automáticamente cuando el usuario actual inicia sesión, establezca la propiedad `startAtLogin` en `true`. Una vez configurada de esta forma, la aplicación se iniciará automáticamente cada vez que el usuario inicie sesión. Continúa iniciándose al iniciar sesión hasta que se cambie la configuración a `false`, el usuario modifique manualmente la configuración a través del sistema operativo o se desinstale la aplicación. El inicio de una aplicación al iniciar sesión es una opción del motor de ejecución. Esta configuración solo se aplica al usuario actual. Para lograr configurar la propiedad `startAtLogin` en `true` la aplicación debe estar ya instalada. Si se configura esta propiedad sin estar instalada la aplicación (cuando se inicia con ADL, por ejemplo), se emite un error.

Nota: *la aplicación no se inicia cuando arranca el ordenador, sino que lo hace al iniciar sesión el usuario.*

Para determinar si una aplicación se ha iniciado automáticamente o como resultado de una acción de usuario, puede examinarse la propiedad `reason` del objeto InvokeEvent. Si la propiedad es igual a `InvokeEventReason.LOGIN`, la aplicación se inicia automáticamente. Para otras rutas de invocación, la propiedad `reason` se define del modo siguiente:

- `InvokeEventReason.NOTIFICATION` (solo iOS) - La aplicación se ha invocado a través de APN. Para obtener más información sobre APN, consulte Uso de notificaciones push.
- `InvokeEventReason.OPEN_URL` - La aplicación se ha invocado a través de otra aplicación o por el sistema.
- `InvokeEventReason.Standard` - Todos los demás casos.

Para acceder a la propiedad `reason`, la aplicación debe especificar como destino AIR 1.5.1 o versión superior (estableciendo el valor de espacio de nombres concreto en el archivo descriptor de la aplicación).

La siguiente aplicación simplificada utiliza la propiedad reason de InvokeEvent para decidir cómo comportarse cuando se produce un evento invoke. Si la propiedad reason es "login", la aplicación permanece en segundo plano. De lo contrario, hace que la aplicación pueda verse. Una aplicación que utilice este patrón suele comenzar en el inicio de sesión de modo que puede llevar a cabo el procesamiento en segundo plano o el control de eventos y abre una ventana como respuesta a un evento invoke desencadenado por el usuario.

```
package {
    import flash.desktop.InvokeEventReason;
    import flash.desktop.NativeApplication;
    import flash.display.Sprite;
    import flash.events.InvokeEvent;

    public class StartAtLogin extends Sprite
    {
        public function StartAtLogin()
        {
            try
            {
                NativeApplication.nativeApplication.startAtLogin = true;
            }
            catch ( e:Error )
            {
                trace( "Cannot set startAtLogin:" + e.message );
            }

            NativeApplication.nativeApplication.addEventListener( InvokeEvent.INVOKE, onInvoke );
        }

        private function onInvoke( event:InvokeEvent ):void
        {
            if( event.reason == InvokeEventReason.LOGIN )
            {
                //do background processing...
                trace( "Running in background..." );
            }
            else
            {
                this.stage.nativeWindow.activate();
            }
        }
    }
}
```

Nota: *para ver la diferencia de comportamiento, empaquete e instale la aplicación. La propiedad* `startAtLogin` *solo se puede definir para aplicaciones instaladas.*

Invocación de una aplicación de AIR desde el navegador

Adobe AIR 1.0 y posterior

La función de invocación desde el navegador permite que un sitio web inicie una aplicación de AIR instalada desde el navegador. La invocación desde el navegador solo se admite si el archivo descriptor de la aplicación define `allowBrowserInvocation` en `true`:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Cuando se invoca la aplicación a través del navegador, el objeto NativeApplication de la aplicación distribuye un objeto BrowserInvokeEvent.

Para recibir eventos BrowserInvokeEvent, llame al método `addEventListener()` del objeto NativeApplication (`NativeApplication.nativeApplication`) en la aplicación de AIR. Cuando un detector de eventos se registra para un evento BrowserInvokeEvent, también recibe todos los eventos BrowserInvokeEvent que se produjeron antes de haberse registrado el detector. Estos eventos se distribuyen tras la devolución de la llamada a `addEventListener()`, pero no necesariamente antes de otros eventos BrowserInvokeEvent que quizá se reciban después de registrar el detector. Esto le permite controlar los eventos BrowserInvokeEvent que se hayan producido antes de ejecutarse el código de inicialización (por ejemplo, cuando la aplicación se invocó desde el navegador). Tenga en cuenta que si se añade un detector de eventos más adelante en la ejecución (después de inicializada la aplicación), aún recibirá todos los eventos BrowserInvokeEvent que se hayan producido desde que se inició la aplicación.

El objeto BrowserInvokeEvent incluye las siguientes propiedades:

Propiedad	**Descripción**
arguments	Un conjunto de argumentos (cadenas) que se pasan a la aplicación.
isHTTPS	Indica si el contenido en el navegador utiliza el esquema https de la URL (`true`) o no (`false`).
isUserEvent	Indica si la invocación desde el navegador produjo un evento de usuario (hacer clic, por ejemplo). En AIR 1.0 siempre está definido en `true`; AIR requiere un evento de usuario para la función de invocación desde el navegador.
sandboxType	El tipo de entorno limitado para el contenido en el navegador. Los valores válidos se definen igual que los que pueden utilizarse en la propiedad `Security.sandboxType` y pueden ser uno de los siguientes: • `Security.APPLICATION`: el contenido se encuentra en el entorno limitado de seguridad de la aplicación. • `Security.LOCAL_TRUSTED`: el contenido se encuentra en el entorno limitado de seguridad local de confianza. • `Security.LOCAL_WITH_FILE`: el contenido se encuentra en el entorno limitado de seguridad local con sistema de archivos. • `Security.LOCAL_WITH_NETWORK`: el contenido se encuentra en el entorno limitado de seguridad local de red. • `Security.REMOTE`: el contenido se encuentra en un dominio remoto (en la red).
securityDomain	El dominio de seguridad para el contenido del navegador, por ejemplo `"www.adobe.com"` o `"www.example.org"`. Esta propiedad solo se define para contenido en el entorno limitado de seguridad remota (para contenido procedente de un dominio de la red). No se define para contenido en un entorno limitado de seguridad local o de la aplicación.

Si utiliza la función de invocación desde el navegador, asegúrese de tener en cuenta las posibles consecuencias para la seguridad. Cuando un sitio web inicia una aplicación de AIR, puede enviar datos a través de la propiedad `arguments` del objeto BrowserInvokeEvent. Tenga cuidado al utilizar estos datos en operaciones sensibles, como las API de carga de archivos o código. El nivel de riesgo dependerá de lo que la aplicación haga con los datos. Si se espera que un solo sitio web en particular invoque la aplicación, esta debería comprobar la propiedad `securityDomain` del objeto BrowserInvokeEvent. También se puede exigir que el sitio web que invoca la aplicación utilice HTTPS, lo cual se puede verificar comprobando la propiedad `isHTTPS` del objeto BrowserInvokeEvent.

La aplicación debe validar los datos que le llegan. Por ejemplo, si una aplicación espera recibir unas URL a un dominio concreto, debería validar que las URL apunten, efectivamente, a ese dominio. Esto puede evitar que un atacante logre engañar la aplicación para que esta le mande datos sensibles.

Ninguna aplicación debería utilizar argumentos BrowserInvokeEvent que puedan apuntar a recursos locales. Por ejemplo: una aplicación no debería crear objetos File basados en una ruta recibida del navegador. Si se espera recibir rutas remotas del navegador, la aplicación debería asegurarse de que las rutas no utilicen el protocolo `file://` en lugar de un protocolo remoto.

Cierre de una aplicación

Adobe AIR 1.0 y posterior

La forma más rápida de cerrar una aplicación es llamar al método NativeApplication exit(). Esto funciona perfectamente cuando la aplicación no tiene datos que guardar ni recursos externos que limpiar. Al llamar a `exit()` se cierran todas las ventanas y después la aplicación. No obstante, para permitir que las ventanas u otros componentes de la aplicación interrumpan el proceso de cierre, quizá para guardar datos esenciales, distribuya los eventos de aviso correspondientes antes de llamar a `exit()`.

Otro aspecto que conviene tener en cuenta para cerrar correctamente una aplicación es la necesidad de proporcionar una sola ruta de ejecución, independientemente de cómo se inicia el proceso de cierre. El usuario (o el sistema operativo) puede activar el cierre de la aplicación de las siguientes maneras:

- Cerrando la última ventana de la aplicación cuando `NativeApplication.nativeApplication.autoExit` está definido en `true`.
- Seleccionando el comando de salir de la aplicación en el sistema operativo; por ejemplo, cuando el usuario selecciona en el menú predeterminado el comando de salir de la aplicación. (Esto solo sucede en Mac OS, ya que Windows y Linux no ofrecen un comando de salida de la aplicación en el fondo cromático del sistema).
- Apagando el ordenador.

Cuando el comando de salir se ejecuta a través del sistema operativo por una de estas vías, el objeto NativeApplication distribuye un evento `exiting`. Si ningún detector cancela el evento `exiting`, se cierran las ventanas que estén abiertas. Cada ventana distribuye un evento `closing` seguido de un evento `close`. Si alguna de las ventanas cancela el evento `closing`, se detiene el proceso de cierre.

Si el orden de cierre de las ventanas es un problema para la aplicación, detecte el evento `exiting` distribuido por el objeto NativeApplication y cierre usted mismo las ventanas en el orden correcto. Este podría ser el caso si, por ejemplo, hay una ventana de documento con paletas de herramientas. Puede resultar inapropiado, o peor, si el sistema cerrara las paletas, pero el usuario decidiera cancelar el comando de salir para guardar datos. En Windows, la única vez que se produce el evento `exiting` es después de cerrar la última ventana (cuando la propiedad `autoExit` del objeto NativeApplication está definida en `true`).

Para obtener un comportamiento similar en todas las plataformas -independientemente de si la secuencia de cierre se inicia a través del fondo cromático del sistema operativo, los comandos de menú o la lógica de la aplicación- conviene observar las siguientes prácticas recomendadas para salir de la aplicación:

1. Distribuya siempre un evento `exiting` a través del objeto NativeApplication antes de llamar a `exit()` en el código de la aplicación y compruebe que no vaya a cancelar el evento otro componente de la aplicación.

```
public function applicationExit():void {
    var exitingEvent:Event = new Event(Event.EXITING, false, true);
    NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        NativeApplication.nativeApplication.exit();
    }
}
```

2. Esté atento al evento `exiting` distribuido por el objeto `NativeApplication.nativeApplication` y, en el controlador, cierre las ventanas abiertas que haya (distribuyendo primero un evento `closing`). Una vez cerradas todas las ventanas, realice las tareas de limpieza necesarias, como guardar los datos de la aplicación o eliminar los archivos temporales. Utilice solamente métodos sincrónicos durante la limpieza para estar seguro de que hayan finalizado antes de que se cierre la aplicación.

Si no importa en qué orden se cierran las ventanas, se puede recorrer el conjunto `NativeApplication.nativeApplication.openedWindows` en un bucle y cerrar cada ventana a su vez. Si el orden *sí* importa, facilite un medio de cerrar las ventanas en la secuencia correcta.

```
private function onExiting(exitingEvent:Event):void {
    var winClosingEvent:Event;
    for each (var win:NativeWindow in NativeApplication.nativeApplication.openedWindows) {
        winClosingEvent = new Event(Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 Las ventanas deben siempre controlar su propia limpieza, detectando sus propios eventos `closing`.

4 Utilice un solo detector de eventos `exiting` en la aplicación, ya que los controladores que se llamaron previamente no pueden saber si los detectores posteriores cancelarán el evento `exiting` (y no conviene depender del orden de ejecución).

Capítulo 51: Trabajo con información sobre el motor de ejecución de AIR y el sistema operativo

Adobe AIR 1.0 y posterior

En esta sección se discuten las formas en que una aplicación de AIR puede gestionar la asociación con archivos del sistema operativo, detectar la actividad de los usuarios y obtener información sobre el motor de ejecución de Adobe® AIR™.

Más temas de ayuda

flash.desktop.NativeApplication

Gestión de asociaciones con archivos

Adobe AIR 1.0 y posterior

Las asociaciones entre la aplicación y un tipo de archivo deben declararse en el descriptor de la aplicación. Durante el proceso de instalación el programa de instalación de la aplicación de AIR asocia esta como aplicación de apertura predeterminada para cada uno de los tipos de archivos declarados, a menos que otra aplicación ya sea la predeterminada. El proceso de instalación de la aplicación de AIR no suprime ninguna asociación de tipo de archivo existente. Para hacerse cargo de una asociación que antes era con otra aplicación, llame al método `NativeApplication.setAsDefaultApplication()` durante el tiempo de ejecución.

Conviene siempre verificar que las asociaciones con archivos esperadas estén establecidas cuando se inicia la aplicación. El motivo es que el programa de instalación de la aplicación de AIR no suprime las asociaciones de archivos existentes, y además las asociaciones de archivos del sistema del usuario pueden cambiar en cualquier momento. Cuando hay otra aplicación asociada con el archivo actual, es además buena educación pedir permiso al usuario antes de sustituir una asociación existente por otra.

Los siguientes métodos de la clase NativeApplication permiten que una aplicación gestione las asociaciones de archivos. Cada uno de los métodos toma como parámetro la extensión del tipo de archivo:

Método	Descripción
isSetAsDefaultApplication()	Devuelve un valor de "true" si la aplicación de AIR está asociada con el tipo de archivo especificado.
setAsDefaultApplication()	Crea la asociación entre la aplicación de AIR y la acción de abrir del tipo de archivo.
removeAsDefaultApplication()	Elimina la asociación entre la aplicación de AIR y el tipo de archivo.
getDefaultApplication()	Notifica la ruta de la aplicación que está asociada con el tipo de archivo.

AIR solo puede manejar asociaciones para los tipos de archivos originalmente declarados en el descriptor de la aplicación. No se puede obtener información sobre las asociaciones de un tipo de archivo no declarado, aunque el usuario haya creado manualmente la asociación entre ese tipo de archivo y la aplicación. Si se llama a cualquiera de los métodos de gestión de asociaciones con archivos con la extensión de un tipo de archivo que no está declarado en el descriptor de la aplicación, esta emitirá una excepción del motor de ejecución.

Obtención de la versión y el nivel de revisión del motor de ejecución

Adobe AIR 1.0 y posterior

El objeto NativeApplication tiene una propiedad `runtimeVersion` que es la versión del motor de ejecución en que se ejecuta la aplicación (una cadena, por ejemplo `"1.0.5"`). El objeto NativeApplication tiene también una propiedad `runtimePatchLevel` que es el nivel de revisión del motor de ejecución (un número, por ejemplo 2960). El código que sigue utiliza estas propiedades:

```
trace(NativeApplication.nativeApplication.runtimeVersion);
trace(NativeApplication.nativeApplication.runtimePatchLevel);
```

Detección de las capacidades de AIR

Adobe AIR 1.0 y posterior

Para un archivo integrado en la aplicación de Adobe AIR, la propiedad `Security.sandboxType` tiene un valor definido por la constante `Security.APPLICATION`. Se puede cargar contenido (que puede o no contener API específicas de AIR) según esté un archivo en el entorno limitado de seguridad de Adobe AIR, como se muestra en el código siguiente:

```
if (Security.sandboxType == Security.APPLICATION)
{
    // Load SWF that contains AIR APIs
}
else
{
    // Load SWF that does not contain AIR APIs
}
```

Todos los recursos que no se instalan con la aplicación de AIR se asignan a los mismos entornos limitados de seguridad que asignaría Adobe® Flash® Player en un navegador web. Los recursos remotos se ponen en entornos limitados de acuerdo con sus dominios de origen, y los recursos locales se ponen en el entorno limitado local de red, local con sistema de archivos o local de confianza.

Se puede comprobar si la propiedad estática está definida en `Capabilities.playerType"Desktop"` para ver si el contenido se está ejecutando en el motor de ejecución (y no en Flash Player ejecutándose en un navegador).

Para obtener más información, consulte “Seguridad en AIR” en la página 1122.

Seguimiento de la presencia de usuarios

Adobe AIR 1.0 y posterior

El objeto NativeApplication distribuye dos eventos que sirven para detectar cuándo está usando activamente el ordenador el usuario. Si no se detecta ninguna actividad del ratón o el teclado en el intervalo determinado por la propiedad `NativeApplication.idleThreshold` el objeto NativeApplication distribuye un evento `userIdle`. La próxima vez que se utiliza el teclado o el ratón, el objeto NativeApplication distribuye un evento `userPresent`. El intervalo `idleThreshold` se mide en segundos y tiene un valor predeterminado de 300 (5 minutos). También se puede obtener el tiempo transcurrido en segundos desde la última entrada realizada por el usuario, en la propiedad `NativeApplication.nativeApplication.lastUserInput`.

Las siguientes líneas de código definen el límite de inactividad en 2 minutos y detectan los eventos `userIdle` y `userPresent`:

```
NativeApplication.nativeApplication.idleThreshold = 120;
NativeApplication.nativeApplication.addEventListener(Event.USER_IDLE, function(event:Event) {
    trace("Idle");
});
NativeApplication.nativeApplication.addEventListener(Event.USER_PRESENT,
function(event:Event) {
    trace("Present");
});
```

Nota: *solo se distribuye un evento* `userIdle` *entre dos eventos* `userPresent` *cualesquiera.*

Capítulo 52: Trabajo con ventanas nativas de AIR

Adobe AIR 1.0 y posterior

Puede utilizar las clases proporcionadas por la API de manejo de ventanas nativas de Adobe® AIR® para crear y gestionar ventanas de escritorio.

Aspectos básicos de las ventanas nativas en AIR

Adobe AIR 1.0 y posterior

Para ver una explicación rápida y ejemplos de código del trabajo con ventanas nativas en AIR, consulte los siguientes artículos de inicio rápido del Centro de desarrollo de Adobe:

- Creating a transparent window application (Creación de una aplicación de ventana transparente, en inglés) (Flex)
- Interacting with a window (Interacción con una ventana, en inglés) (Flex)
- Customizing the look and feel of a native window (Personalización del aspecto de una ventana nativa, en inglés) (Flex)
- Launching windows (Inicio de ventanas, en inglés) (Flex)
- Creating toast-style windows (Creación de ventanas superpuestas, en inglés) (Flex)
- Controlling the display order of windows (Control del orden de visualización de las ventanas, en inglés) (Flex)
- Creating resizable, non-rectangular windows (Creación de ventanas no rectangulares de tamaño variable, en inglés) (Flex)
- Interacting with a window (Interacción con una ventana, en inglés) (Flash)
- Customizing the look and feel of a native window (Personalización del aspecto de una ventana nativa, en inglés) (Flash)
- Creating toast-style windows (Creación de ventanas superpuestas, en inglés) (Flash)
- Controlling the display order of windows (Control del orden de visualización de las ventanas, en inglés) (Flash)
- Creating resizable, non-rectangular windows (Creación de ventanas no rectangulares de tamaño modificable, en inglés) (Flash)

AIR proporciona una API de ventana fácil de usar y válida para todas las plataformas con la que es posible crear ventanas nativas de sistema operativo utilizando Flash®, Flex™ y técnicas de programación HTML.

Con AIR, las posibilidades de desarrollar el aspecto de su aplicación son enormes. Las ventanas que se crean pueden presentar un aspecto similar a una aplicación de escritorio estándar, coincidiendo con el estilo de Apple cuando se ejecutan en Mac, ajustándose a las convenciones de Microsoft cuando se ejecutan en Windows y en armonía con el administrador de ventanas de Linux; todo ello sin incluir ninguna línea de código de plataforma específica. También puede utilizar el fondo cromático de aspecto configurable de la arquitectura de Flex para establecer su propio estilo sin importar dónde se ejecute la aplicación. Incluso puede dibujar su propio fondo cromático de ventana con ilustraciones vectoriales o de mapas de bits con compatibilidad total de transparencia y valores alfa que se fundan con el escritorio. ¿Está cansado de tanta ventana rectangular? Dibuje una redondeada.

Ventanas de AIR

Adobe AIR 1.0 y posterior

AIR admite tres API diferentes para trabajar con ventanas:

- La clase NativeWindow orientada a ActionScript proporciona la API de gestión de ventanas de nivel más bajo. Utilice NativeWindows en ActionScript y aplicaciones creadas con Flash Professional. Considere la ampliación de la clase NativeWindow para que se especialice en las ventanas utilizadas en la aplicación.
- En el entorno HTML, se puede usar la clase Window de JavaScript, tal y como se haría en una aplicación web basada en navegador. Las llamadas a los métodos Window de JavaScript se reenvían al objeto de ventana nativa subyacente.
- Las clases del marco de Flex, mx:WindowedApplication y mx:Window, proporcionan un "envolvente" de Flex para la clase NativeWindow. El componente WindowedApplication sustituye al componente Application cuando se crea una aplicación de AIR con Flex y siempre se debe usar como ventana inicial en la aplicación de Flex.

Ventanas de ActionScript

Al crear ventanas con la clase NativeWindow, puede utilizar el escenario y la lista de visualización de Flash Player directamente. Para añadir un objeto visual a una ventana NativeWindow, añada el objeto a la lista de visualización del escenario de la ventana o a otro contenedor de objetos de visualización del escenario.

Ventanas HTML

Cuando se crean ventanas HTML, se utiliza HTML, CSS y JavaScript para visualizar el contenido. Para añadir un objeto visual a una ventana HTML, añada el contenido al DOM HTML. Las ventanas HTML son una categoría especial de NativeWindow. El host de AIR define una propiedad `nativeWindow` en las ventanas HTML que proporciona acceso a la instancia subyacente de NativeWindow. Puede utilizar esta propiedad para acceder a las propiedades, métodos y eventos de NativeWindow que se describen en esta sección.

Nota: *el objeto Window de JavaScript también cuenta con métodos para crear scripts de la ventana que los contiene, como por ejemplo,* `moveTo()` *y* `close()`*. Si dispone de varios métodos al mismo tiempo, puede utilizar el que más le convenga.*

Ventanas de la arquitectura de Flex

Cuando se crean ventanas con la arquitectura de Flex, se suelen utilizar componentes MXML para llenar la ventana. Para añadir un componente de Flex a una ventana, añada el elemento del componente a la definición MXML de la ventana. También puede utilizar ActionScript para añadir contenido dinámicamente. Los componentes mx:WindowedApplication y mx:Window se diseñan como contenedores de Flex, por lo que pueden aceptar componentes de Flex directamente, mientras que los objetos NativeWindow no. Si es necesario, se puede acceder a los métodos y propiedades de NativeWindow mediante los objetos WindowedApplication y Window, y la propiedad `nativeWindow`.

Ventana inicial de la aplicación

AIR crea para el usuario la primera ventana de la aplicación automáticamente. AIR define las propiedades y el contenido de la ventana a partir de los parámetros especificados en el elemento `initialWindow` del archivo descriptor de la aplicación.

Si el contenido raíz es un archivo SWF, AIR crea una instancia de NativeWindow, carga el archivo SWF y lo añade al escenario de la ventana. Si el contenido raíz es un archivo HTML, AIR crea una ventana HTML y carga el contenido HTML.

Clases de ventanas nativas

Adobe AIR 1.0 y posterior

La API de gestión de ventanas nativas contiene las siguientes clases:

Paquete	Clases
flash.display	• NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	• NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

Flujo de eventos de ventanas nativas

Adobe AIR 1.0 y posterior

Las ventanas nativas distribuyen eventos para notificar a los componentes pertinentes sobre cambios importantes que se han producido o pueden producirse. Muchos eventos relacionados con ventanas se distribuyen por parejas. El primer evento advierte sobre un cambio que va a producirse. El segundo evento anuncia que el cambio se ha realizado. Puede cancelar un evento de advertencia, pero no uno de notificación. En la siguiente secuencia se muestra el flujo de eventos que se producen cuando un usuario hace clic en el botón Maximizar de una ventana:

1. El objeto NativeWindow distribuye un evento `displayStateChanging`.
2. Si ningún detector registrado lo cancela, la ventana se maximiza.
3. El objeto NativeWindow distribuye un evento `displayStateChange`.

   Además, el objeto NativeWindow también distribuye eventos para los cambios relacionados con el tamaño y la posición de la ventana. La ventana no distribuye eventos de advertencia para estos cambios relacionados. Los eventos relacionados son:

   a. Un evento `move` se distribuye si la esquina superior izquierda de la ventana se mueve por maximización.
   b. Un evento `resize` se distribuye si el tamaño de la ventana cambia por maximización.

   Un objeto NativeWindow distribuye una secuencia similar de eventos cuando una ventana se minimiza, se restaura, se cierra, se mueve y se cambia de tamaño.

Los eventos de advertencia solo se distribuyen si se inicia un cambio del fondo cromático de la ventana o por cualquier otro mecanismo controlado por el sistema operativo. Cuando se llama a un método de ventana para que cambie el tamaño, la posición o el estado de visualización de la ventana, esta solo distribuye un evento para anunciar el cambio. Si lo desea, puede distribuir un evento de advertencia con el método `dispatchEvent()` de la ventana y, después, ver si el evento de advertencia se ha cancelado antes de realizar el cambio.

Para obtener más información sobre las clases, métodos, propiedades y eventos de API de ventana, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Propiedades que controlan el estilo y el comportamiento de una ventana nativa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las siguientes propiedades controlan el aspecto y el comportamiento básicos de una ventana:

- `type`
- `systemChrome`
- `transparent`
- `owner`

Al crear una ventana, estas propiedades se establecen en el objeto NativeWindowInitOptions transferido al constructor de la ventana. AIR lee las propiedades de la ventana inicial de la aplicación desde el descriptor de la aplicación. (Salvo la propiedad `type`, que no se puede establecer en el descriptor de la aplicación y siempre se define en `normal`.) Las propiedades no se pueden modificar una vez creada la ventana.

Algunos ajustes de estas propiedades son incompatibles entre sí: `systemChrome` no se puede establecer como `standard` si `transparent` es `true` o si `type` es `lightweight`.

Tipos de ventanas

Adobe AIR 1.0 y posterior

Los tipos de ventanas de AIR combinan atributos de fondo cromático y visibilidad del sistema operativo nativo para crear tres tipos de ventanas funcionales. Puede utilizar las constantes definidas en la clase NativeWindowType para hacer referencia a los nombres de los tipos en el código. AIR proporciona los siguientes tipos de ventanas:

Texto	Descripción
Normal	Una ventana normal. La ventanas normales utilizan el fondo cromático de pantalla completa y se muestran en la barra de tareas de Windows o en el menú Ventana de Mac OS X.
Utilidades	Una paleta de herramientas. Las ventanas de utilidades utilizan una versión más ligera del fondo cromático del sistema y no se muestran en la barra de tareas de Windows ni en el menú Ventana de Mac OS X.
Ligeras	Las ventanas ligeras no tienen fondo cromático y no aparecen en la barra de tareas de Windows ni en el menú Ventana de Mac OS X. Además, las ventanas ligeras no disponen de menú Sistema (Alt+Barra espaciadora) en Windows. El uso de ventanas ligeras está recomendado para mensajes emergentes de notificación o controles, como cuadros emergentes áreas de visualización temporal. Si utiliza el `type` en ventanas ligeras, `systemChrome` debe establecerse como `none`.

Fondo cromático de una ventana

Adobe AIR 1.0 y posterior

El fondo cromático de una ventana es el conjunto de controles que permiten a los usuarios manipular la ventana en un entorno de escritorio. Los elementos del fondo cromático son la barra de título, los botones de la barra de título, los bordes y los selectores de cambio de tamaño.

Fondo cromático del sistema

Puede establecer la propiedad `systemChrome` como `standard` o como `none`. Seleccione el valor `standard` de fondo cromático para darle a la ventana el conjunto de controles creados e ideados por el sistema operativo del usuario. Seleccione `none` para proporcionar su propio fondo cromático a la ventana. Utilice las constantes definidas en la clase NativeWindowSystemChrome para hacer referencia a los parámetros del fondo cromático del sistema en el código.

La gestión del fondo cromático del sistema corre a cargo del propio sistema. La aplicación no tiene acceso directo a los controles, pero puede reaccionar ante la distribución de eventos cuando se utilicen los controles. Si se utiliza el fondo cromático estándar para una ventana, la propiedad `transparent` debe establecerse como `false` y la propiedad `type` debe ser `normal` o `utility`.

Fondo cromático de Flex

Si utiliza los componentes WindowedApplication o Window de Flex, la ventana puede utilizar el fondo cromático del sistema o el de la arquitectura de Flex. Para utilizar el fondo cromático de Flex, establezca la propiedad `systemChrome` utilizada para crear la ventana como `none`. Si se utilizan componentes spark de Flex 4 y no componentes mx, debe especificar la clase de aspecto para poder usar el fondo cromático de Flex. Puede utilizar los aspectos incorporados o proporcionar los suyos propios. El siguiente ejemplo muestra cómo usar la clase de aspecto spark WindowedApplication incorporada para proporcionar el fondo cromático de la ventana:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
@namespace "library://ns.adobe.com/flex/spark";
WindowedApplication
{
    skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
}
</fx:Style>
</s:WindowedApplication>
```

Para obtener más información, consulte Using Flex 4: About the AIR window containers: Controlling window chrome (en inglés)

Fondo cromático personalizado

Cuando se crea una ventana sin fondo cromático del sistema, es preciso añadir controles de fondo cromático propios para controlar la interacción entre el usuario y la ventana. También, si lo desea, puede crear ventanas no rectangulares y transparentes.

Para utilizar el fondo cromático personalizado con los componentes mx:WindowedApplication o mx:Window, el estilo `showFlexChrome` se debe establecer en `false`. De lo contrario, Flex añadirá su propio fondo cromático a las ventanas.

Transparencia de la ventana

Adobe AIR 1.0 y posterior

Para permitir la mezcla alfa en una ventana de escritorio o en cualquier otra, establezca la propiedad `transparent` de la ventana como `true`. Debe establecer la propiedad `transparent` antes de crear la ventana y no es posible modificarla.

Una ventana transparente no tiene fondo predeterminado. Cualquier área de la ventana que no contenga objetos dibujados por la aplicación es invisible. Si un objeto visualizado tiene un valor alfa menor que uno, cualquier elemento debajo del objeto será transparente, incluidos los demás objetos de visualización de la misma ventana, de otras ventanas y del escritorio.

Las ventanas transparentes resultan útiles cuando se quieren crear aplicaciones con bordes de forma irregular, aplicaciones que "desaparecen" o aplicaciones invisibles. No obstante, la representación de zonas grandes de mezcla alfa puede ser un proceso lento, por lo que el efecto debe utilizarse con precaución.

Importante: *en Linux, los eventos de ratón no pasan por píxeles completamente transparentes. Se debe evitar la creación de ventanas con áreas grandes y completamente transparentes, ya que se puede bloquear de forma invisible el acceso del usuario a otras ventanas o elementos en su escritorio. En Mac OS X y Windows, los eventos de ratón no pasan por píxeles completamente transparentes.*

La transparencia no se puede aplicar a ventanas con fondo cromático del sistema. Además, el contenido SWF y PDF del HTML no se visualiza en las ventanas transparentes. Para obtener más información, consulte "Consideraciones al cargar el contenido SWF o PDF en una página HTML" en la página 1045.

La propiedad `NativeWindow.supportsTransparency` indica si la ventana puede ser transparente. Si no se admite transparencia, la aplicación se compone con un fondo negro. En estos casos, cualquier área transparente de la aplicación se visualiza en negro opaco. Se recomienda ofrecer un recurso alternativo en caso de que esta propiedad sea `false`. Por ejemplo, puede mostrar un cuadro diálogo de advertencia al usuario o una interfaz rectangular no transparente.

Tenga en cuenta que la transparencia se admite siempre en los sistemas operativos Mac y Windows. La compatibilidad con los sistemas operativos Linux requiere un administrador de ventanas de composición, pero aunque este administrador esté activo, la transparencia puede no estar disponible debido a las opciones de visualización del usuario o a la configuración de hardware.

Transparencia en una ventana de aplicación MXML

Adobe AIR 1.0 y posterior

De forma predeterminada, el fondo de una ventana MXML es opaco, incluso si se crea la ventana con el valor *transparent*. (Observe el efecto de transparencia en las esquinas de la ventana.) Para presentar un fondo transparente en la ventana, establezca un color de fondo y un valor de alfa en la hora de estilos o en el elemento <mx:Style> del archivo MXML de la aplicación. Por ejemplo, la siguiente declaración de estilo añade una sombra verde ligeramente transparente al fondo:

```
WindowedApplication
{
    background-alpha:".8";
    background-color:"0x448234";
}
```

Transparencia en una ventana de aplicación HTML

Adobe AIR 1.0 y posterior

De forma predeterminada, el fondo del contenido HTML se visualiza en las ventanas HTML y en los objetos HTMLLoader como opaco, incluso si la ventana que lo contiene es transparente. Para desactivar el fondo predeterminado para el contenido HTML, establezca la propiedad `paintsDefaultBackground` como `false`. El siguiente ejemplo crea un objeto HTMLLoader y desactiva el fondo predeterminado:

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

Este ejemplo utiliza JavaScript para desactivar el fondo predeterminado de una ventana HTML:

```
window.htmlLoader.paintsDefaultBackground = false;
```

Si un elemento del documento HTML establece un color de fondo, el fondo de dicho elemento no es transparente. No está permitido establecer un valor de transparencia (u opacidad) parcial. Sin embargo, se puede utilizar un gráfico transparente en formato PNG como fondo de página o de elemento de página para conseguir un efecto visual parecido.

Propiedad de las ventanas

Una ventana puede *contener* una o varias ventanas. Estas ventanas de propiedad aparecen frente a la venta maestra, se minimizan y se restauran con la ventana maestra y se cierran cuando se cierra la ventana maestra. La propiedad de la ventana no se puede transferir a otra ventana ni eliminar. Una ventana solo puede formar parte de una ventana maestra, pero puede poseer cualquier cantidad de ventanas.

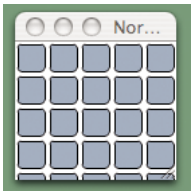
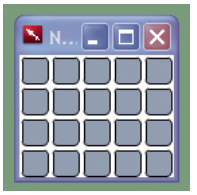
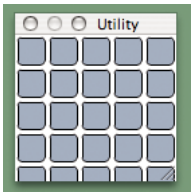
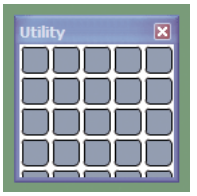
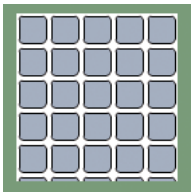
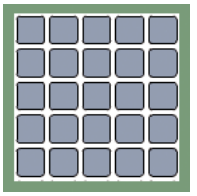
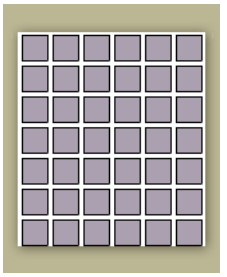
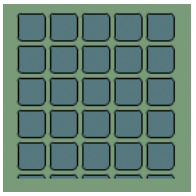
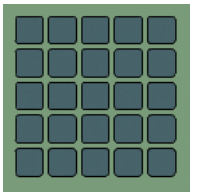
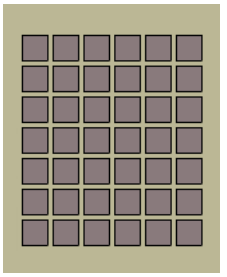
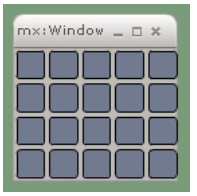
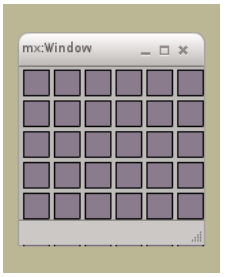
La propiedad de las ventanas se puede utilizar para facilitar la administración de ventanas utilizadas para paletas de herramientas y cuadros de diálogo. Por ejemplo, si se mostrara un cuadro de diálogo Guardar asociado a una ventana de documento, al hacer que la ventana de documento contenga el cuadro de diálogo, este se mantendrá frente a la ventana del documento automáticamente.

- NativeWindow.owner
- Christian Cantrell: Owned windows in AIR 2.6

Catálogo de ventana visual

Adobe AIR 1.0 y posterior

En la siguiente tabla se resumen los efectos visuales de las distintas combinaciones de parámetros de propiedades de ventanas en los sistemas operativos Mac OS X, Windows y Linux:

Parámetros de ventana	Mac OS X	Microsoft Windows	Linux*
Type: normal SystemChrome: estándar Transparent: false			
Type: utility SystemChrome: estándar Transparent: false			
Type: cualquiera SystemChrome: ninguno Transparent: false			
Type: cualquiera SystemChrome: ninguno Transparent: true			
mxWindowedApplication o mx:Window Type: cualquiera SystemChrome: ninguno Transparent: true			

Ubuntu con el administrador de ventanas Compiz

Nota: *los siguientes elementos del fondo cromático del sistema no se admiten en AIR: la barra de herramientas de Mac OS X, el icono proxy de Mac OS X, los iconos de barra de título de Windows y otros fondos cromáticos alternativos.*

Creación de ventanas

Adobe AIR 1.0 y posterior

AIR crea automáticamente la primera ventana de la aplicación, pero el usuario puede crear más ventanas adicionales si lo necesita. Para crear una ventana nativa, utilice el método NativeWindow.

Para crear una ventana HTML, utilice el método `createRootWindow()` de HTMLLoader o, desde un documento HTML, llame al método `window.open()` de JavaScript. La ventana creada es un objeto NativeWindow cuya lista de visualización contiene un objeto HTMLLoader. El objeto HTMLLoader interpreta y muestra el contenido HTML y JavaScript para la ventana. Se puede acceder a las propiedades del objeto NativeWindow subyacente desde JavaScript usando la propiedad `window.nativeWindow`. (A esta propiedad solo puede acceder el código que se ejecuta en el entorno limitado de la aplicación AIR.)

Cuando se inicializa una ventana (incluida la ventana inicial de la aplicación), debe considerar la creación de la ventana en estado invisible, cargar el contenido o ejecutar actualizaciones gráficas y, después, hacer la ventana visible. Esta secuencia impide que se produzcan fallos visuales en el proceso y que el usuario pueda verlos. Puede especificar que la ventana inicial de la aplicación se cree en estado invisible especificando la etiqueta `<visible>false</visible>` en el descriptor de la aplicación (o dejando la etiqueta fuera, ya que `false` es el valor predeterminado). Los nuevos objetos NativeWindow son invisibles de forma predeterminada. Cuando se crea una ventana HTML con el método HTMLLoader `createRootWindow()`, se puede establecer el argumento `visible` como `false`. Llame al método NativeWindow `activate()` o establezca la propiedad `visible` como `true` para hacer la ventana visible.

Especificación de propiedades de inicialización de una ventana

Adobe AIR 1.0 y posterior

Las propiedades de inicialización de una ventana nativa no se pueden modificar una vez creada la ventana de escritorio. Estas propiedades invariables y sus valores predeterminados son:

Propiedad	Valor predeterminado
systemChrome	standard
type	normal
transparent	false
owner	null
maximizable	true
minimizable	true
resizable	true

Establezca las propiedades de la ventana inicial creada por AIR en el archivo descriptor de la aplicación. La ventana principal de una aplicación de AIR siempre es de tipo *normal*. (Se pueden especificar propiedades adicionales de la ventana en el archivo descriptor, por ejemplo, `visible`, `width` y `height`; estas propiedades se pueden modificar en cualquier momento.)

Puede establecer las propiedades de otras ventanas nativas y HTML creadas por la aplicación mediante la clase NativeWindowInitOptions. Cuando se crea una ventana, se debe transferir un objeto NativeWindowInitOptions especificando las propiedades de la ventana en la función constructora NativeWindow o en el método `createRootWindow()` de HTMLLoader.

El siguiente código crea un objeto NativeWindowInitOptions para una ventana de utilidades:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.type = NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

No está permitido establecer *systemChrome* como *standard* si *transparent* es `true` o `type` es `lightweight`.

Nota: *no es posible definir las propiedades de inicialización de una ventana creada con la función `window.open()` de JavaScript. Sin embargo, sí se puede anular el modo en que se crean estas ventanas implementando su propia clase HTMLHost. Consulte* *"Gestión de llamadas JavaScript a window.open()" en la página 1057* *para obtener más información.*

Cuando se crea una ventana con la clase mx:Window de Flex, se especifican las propiedades de inicialización en el propio objeto window, bien en la declaración MXML de la ventana o en el código que la crea. El objeto subyacente NativeWindow no se crea hasta que se llama al método`open()`. Una vez abierta la ventana, estas propiedades de inicialización ya no se pueden modificar.

Creación de la ventana inicial de la aplicación

Adobe AIR 1.0 y posterior

AIR crea la ventana inicial de la aplicación a partir de las propiedades especificadas en el descriptor de la aplicación y carga el archivo al que se hace referencia en el elemento del contenido. El elemento de contenido debe hacer referencia a un archivo SWF o HTML.

La ventana inicial puede ser la ventana principal de la aplicación o simplemente una o varias ventanas adicionales que se abren. No tienen por qué ser visibles.

Creación de la ventana inicial con ActionScript

Adobe AIR 1.0 y posterior

Si crea una aplicación de AIR con ActionScript, la clase principal de la aplicación debe ampliar la clase Sprite (o una clase secundaria de la clase Sprite). Esta clase sirve de punto de entrada principal para la aplicación.

Cuando se inicia la aplicación, AIR crea una ventana, crea una instancia de la clase principal y añade la instancia al escenario de la ventana. Para acceder a la ventana, puede detectar el evento `addedToStage` y utilizar la propiedad `nativeWindow` del objeto Stage para obtener una referencia al objeto NativeWindow.

El siguiente ejemplo ilustra la estructura básica de la clase principal de una aplicación de AIR creada con ActionScript:

```
package {
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;

    public class MainClass extends Sprite
    {
        private var mainWindow:NativeWindow;
        public function MainClass(){
            this.addEventListener(Event.ADDED_TO_STAGE, initialize);
        }

        private function initialize(event:Event):void{
            mainWindow = this.stage.nativeWindow;
            //perform initialization...
            mainWindow.activate(); //show the window
        }
    }
}
```

Nota: *técnicamente, se PUEDE acceder a la propiedad* `nativeWindow` *en la función constructora de la clase principal. Sin embargo, se trata de un caso especial que solo se aplica a la ventana inicial de la aplicación.*

Al crear una aplicación en Flash Professional, la clase del documento principal se crea automáticamente si no se crea una clase propia en un archivo independiente de ActionScript. Es posible acceder al objeto NativeWindow de la ventana inicial utilizando la propiedad `nativeWindow` del escenario. Por ejemplo, el siguiente código activa la ventana principal en estado maximizado (desde la línea de tiempo):

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

Creación de la ventana inicial con Flex

Adobe AIR 1.0 y posterior

Si crea una aplicación de AIR con la arquitectura de Flex, utilice la clase mx:WindowedApplication como elemento raíz del archivo MXML principal. (Puede utilizar el componente mx:Application, pero tenga en cuenta que este componente no admite todas las funciones disponibles en AIR.) El componente WindowedApplication sirve de punto de entrada inicial de la aplicación.

Cuando se inicia la aplicación, AIR crea una ventana nativa, inicializa la arquitectura de Flex y añade el objeto WindowedApplication al escenario de la ventana. Cuando finaliza la secuencia de inicio, el objeto WindowedApplication distribuye un evento `applicationComplete`. Puede acceder al objeto de ventana de escritorio con la propiedad `nativeWindow` de la instancia de WindowedApplication.

El siguiente ejemplo crea un componente WindowedApplication sencillo que establece sus coordenadas x e y:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    applicationComplete="placeWindow()">
    <mx:Script>
        <![CDATA[
            private function placeWindow():void{
                this.nativeWindow.x = 300;
                this.nativeWindow.y = 300;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

Creación de una ventana nativa

Adobe AIR 1.0 y posterior

Para crear una ventana NativeWindow, transfiera un objeto NativeWindowInitOptions al constructor NativeWindow:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow:NativeWindow = new NativeWindow(options);
```

La ventana no se muestra hasta que se establece la propiedad `visible` como `true` o se llama al método `activate()`.

Una vez creada la ventana, puede inicializar sus propiedades y cargar contenido utilizando la propiedad de escenario y técnicas de lista de visualización de Flash.

En casi todos los casos, debe establecer la propiedad `scaleMode` del escenario de una nueva ventana nativa como `noScale` (utilice la constante `StageScaleMode.NO_SCALE`). Los modos de escala de Flash están diseñados para situaciones en las que el autor de la aplicación desconoce la relación de aspecto del espacio de visualización de la aplicación. Los modos de escala permiten al autor escoger la opción menos arriesgada: recortar el contenido, estrecharlo o apretarlo, o incluso llenarlo con un espacio vacío. Como el desarrollador controla el espacio de visualización en AIR (el marco de la ventana), es posible cambiar el tamaño de la ventana para que se ajuste al contenido, o cambiar el tamaño sin aplicar ningún ajuste.

El modo de escala de ventanas de Flex y HTML se establece como `noScale` automáticamente.

Nota: *para determinar los tamaños máximo y mínimo de ventana admitidos en el sistema operativo actual, utilice las siguientes propiedades estáticas de NativeWindow:*

```
var maxOSSize:Point = NativeWindow.systemMaxSize;
var minOSSize:Point = NativeWindow.systemMinSize;
```

Creación de una ventana HTML

Adobe AIR 1.0 y posterior

Para crear una ventana HTML, puede llamar al método `Window.open()` de JavaScript, o llamar al método `createRootWindow()` de la clase HTMLLoader de AIR.

El contenido HTML de cualquier entorno limitado de seguridad puede utilizar el método `Window.open()` estándar de JavaScript. Si el contenido se ejecuta fuera del entorno limitado de la aplicación, se puede llamar al método `open()` solo como respuesta a la interacción del usuario, por ejemplo, cuando hace clic con el ratón o cuando pulsa una tecla. Cuando se llama a `open()`, se crea una ventana con fondo cromático del sistema para visualizar el contenido en la dirección URL especificada. Por ejemplo:

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

Nota: *puede ampliar la clase HTMLHost en ActionScript para personalizar la ventana creada con la función `window.open()` de JavaScript. Consulte "Ampliación de la clase HTMLHost" en la página 1049.*

El contenido del entorno limitado de seguridad de la aplicación tiene acceso al método más potente para la creación de ventanas: `HTMLLoader.createRootWindow()`. Con este método, es posible especificar todas las opciones de creación de una ventana nueva. El siguiente código JavaScript, por ejemplo, crea una ventana ligera sin fondo cromático del sistema de 300x400 píxeles de tamaño:

```
 var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

Nota: *si el contenido cargado por una nueva ventana se encuentra fuera del entorno limitado de seguridad de la aplicación, el objeto window no tiene las siguientes propiedades de AIR: `runtime`, `nativeWindow` o `htmlLoader`.*

Si crea una ventana transparente, el contenido SWF incorporado en el HTML cargado en la ventana no siempre se visualizará. Debe establecer el parámetro `wmode` del objeto o etiqueta incorporada para que haga referencia al archivo SWF de `opaque` o `transparent`. El valor predeterminado de `wmode` es `window`, por lo que el contenido SWF no se visualiza en ventanas transparentes. Un PDF no puede visualizarse en ventanas transparentes, independientemente del valor de `wmode`. (Antes de AIR 1.5.2, el contenido SWF tampoco se podía visualizar en ventanas transparentes.)

Las ventanas creadas con el método `createRootWindow()` son independientes de la ventana que se abre. Las propiedades `parent` y `opener` del objeto Window de JavaScript son `null`. La ventana que se abre puede acceder al objeto Window de la nueva ventana utilizando la referencia a HTMLLoader devuelta por la función `createRootWindow()`. En el contexto del ejemplo anterior, la sentencia `newHTMLLoader.window` haría referencia al objeto Window de JavaScript de la ventana creada.

Nota: *se puede llamar a la función `createRootWindow()` desde JavaScript y ActionScript.*

Creación de una ventana mx:Window

Adobe AIR 1.0 y posterior

Para crear una ventana mx:Window, puede crear un archivo MXML utilizando mx:Window como etiqueta raíz, o bien llamar directamente al constructor de la clase Window.

El siguiente ejemplo crea y muestra una ventana mx:Window llamando al constructor Window:

```
 var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

Cómo añadir contenido a una ventana

Adobe AIR 1.0 y posterior

La forma en que se añade contenido a las ventanas de AIR depende del tipo de ventana. Por ejemplo, las ventanas MXML y HTML permiten declarar definiciones del contenido básico de la ventana. Puede incorporar recursos en los archivos SWF de la aplicación o puede cargarlos desde archivos de la aplicación independientes. El contenido de Flex, Flash y HTML se puede crear sobre la marcha y añadirlo dinámicamente a una ventana.

Cuando se carga contenido SWF o HTML que contiene JavaScript, se debe tener en cuenta el modelo de seguridad de AIR. Cualquier contenido del entorno limitado de seguridad de la aplicación, es decir, el contenido instalado y cargado con el esquema de URL app: de la aplicación, tiene privilegios completos para acceder a todas las API de AIR. Cualquier contenido cargado desde fuera de este entorno limitado no podrá acceder a las API de AIR. El contenido de JavaScript situado alojado fuera del entorno limitado de la aplicación no puede utilizar las propiedades `runtime`, `nativeWindow` o `htmlLoader` del objeto Window de JavaScript.

Para que el uso de scripts sea seguro, puede utilizar el puente de entorno limitado para facilitar una interfaz limitada entre el contenido de la aplicación y el que no lo es. En contenido HTML, también puede asignar imágenes de la aplicación al entorno limitado ajeno a a la aplicación para que el código de dicha página pueda utilizar el contenido externo de los scripts. Consulte "Seguridad en AIR" en la página 1122.

Carga de un archivo SWF o una imagen

Puede cargar archivos SWF de Flash o imágenes en la lista de visualización de una ventana nativa utilizando la clase `flash.display.Loader`:

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.display.Loader;

    public class LoadedSWF extends Sprite
    {
        public function LoadedSWF(){
            var loader:Loader = new Loader();
            loader.load(new URLRequest("visual.swf"));
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);
        }

        private function loadFlash(event:Event):void{
            addChild(event.target.loader);
        }
    }
}
```

Nota: *los archivos SWF antiguos creados con ActionScript 1 o 2 comparten estados globales, como definiciones de clases, objetos singleton y variables globales si están cargados en la misma ventana. Si este tipo de archivo SWF depende de estados globales sin modificar para que funcione correctamente, no podrá cargarse más de una vez en la misma ventana, ni cargarse otro archivo SWF en la misma ventana con las mismas definiciones de clases y variables. Este contenido se puede cargar en ventanas separadas.*

Carga de contenido HTML en una ventana NativeWindow

Para cargar contenido HTML en una ventana NativeWindow, puede añadir un objeto HTMLLoader al escenario de la ventana y cargar el contenido HTML en HTMLLoader, o bien crear una ventana que ya contenga un objeto HTMLLoader utilizando el método `HTMLLoader.createRootWindow()`. El siguiente ejemplo muestra contenido HTML en un área de visualización de 300 x 500 píxeles en el escenario de una ventana nativa:

```
 //newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

Para cargar una página HTML en una aplicación de Flex, puede utilizar el componente HTML de Flex.

El contenido SWF de un archivo HTML no se visualiza si la ventana utiliza transparencia (es decir, si la propiedad `transparent` de la ventana es `true`) a no ser que el parámetro `wmode` del objeto o la etiqueta incorporada para hacer referencia al archivo SWF se establezca en `opaque` o en `transparent`. Como el valor predeterminado de `wmode` es `window`, el contenido SWF no se visualiza en una ventana transparente. El contenido PDF no se visualiza en una ventana transparente independientemente del valor utilizado para `wmode`.

Del mismo modo, ni el contenido SWF ni PDF se visualiza cuando se escala o gira el control HTMLLoader, o si la propiedad `alpha` de HTMLLoader se establece en un valor distinto de 1.0.

Cómo añadir contenido SWF como superposición en una ventana HTML

Dado que las ventanas HTML están contenidas en una instancia de NativeWindow, es posible añadir objetos de visualización de Flash delante y detrás de la capa HTML en la lista de visualización.

Para añadir un objeto de visualización sobre la capa HTML, utilice el método `addChild()` de la propiedad `window.nativeWindow.stage`. El método `addChild()` añade contenido en capas sobre cualquier contenido existente en la ventana.

Para añadir un objeto de visualización debajo de la capa HTML, utilice el método `addChildAt()` de la propiedad `window.nativeWindow.stage`, transfiriendo un valor de cero para el parámetro `index`. Si coloca un objeto en el índice cero, se mueve el contenido existente (incluida la visualización HTML) una capa más arriba y se inserta el nuevo contenido en el capa inferior. Para que el contenido distribuido en capas debajo de la página HTML sea visible, debe establecer la propiedad `paintsDefaultBackground` del objeto `HTMLlLoader` como `false`. Además, todos los elementos de la página que establezcan un color de fondo no serán transparentes. Si, por ejemplo, establece un color de fondo para el elemento "body" de la página, ninguna parte de la página será transparente.

El siguiente ejemplo muestra cómo añadir objetos de visualización de Flash como superposiciones y como capas inferiores en una página HTML. El ejemplo crea dos objetos de formas sencillas, y añade uno debajo de contenido HTML y el otro encima. El ejemplo también actualiza la posición de la forma en función del evento `enterFrame`.

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
        this.shape.x += this.vX;
        this.shape.y += this.vY;

        //Keep the sprite within the window
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);

    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
```

```
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

Este ejemplo proporciona una introducción rudimentaria a algunas técnicas avanzadas que utilizan tanto JavaScript como ActionScript en AIR. Si no está familiarizado con el uso de objetos de visualización de ActionScript consulte "Programación de la visualización" en la página 154 en la *Guía del desarrollador de ActionScript 3.0*.

Ejemplo: Creación de una ventana nativa

Adobe AIR 1.0 y posterior

El siguiente ejemplo muestra cómo crear una ventana nativa:

```
public function createNativeWindow():void {
    //create the init options
    var options:NativeWindowInitOptions = new NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = NativeWindowSystemChrome.STANDARD;
    options.type = NativeWindowType.NORMAL;

    //create the window
    var newWindow:NativeWindow = new NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    newWindow.stage.align = StageAlign.TOP_LEFT;
    newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;

    //activate and show the new window
    newWindow.activate();
}
```

Gestión de ventanas

Adobe AIR 1.0 y posterior

Puede utilizar las propiedades y métodos de la clase NativeWindow para gestionar el aspecto, el comportamiento y el ciclo de vida de las ventanas de escritorio.

Nota: *al utilizar Flex, suele ser mejor administrar el comportamiento de la ventana utilizando las clases de la arquitectura. A la mayoría de los métodos y propiedades de NativeWindow se puede acceder mediante las clases mx:WindowedApplication y mx:Window.*

Obtención de una instancia de NativeWindow

Adobe AIR 1.0 y posterior

Para poder manipular una ventana, primero es necesario obtener la instancia de la ventana. Puede obtener una instancia de ventana de uno de los lugares siguientes:

- El constructor de ventanas nativas utilizado para crear la ventana:

  ```
  var win:NativeWindow = new NativeWindow(initOptions);
  ```

- Propiedad `nativeWindow` del escenario de la ventana:

  ```
  var win:NativeWindow = stage.nativeWindow;
  ```

- Propiedad `stage` de un objeto de visualización en la ventana:

  ```
  var win:NativeWindow = displayObject.stage.nativeWindow;
  ```

- Propiedad `target` de un evento de ventana nativa distribuido por la ventana:

  ```
  private function onNativeWindowEvent(event:NativeWindowBoundsEvent):void
  {
      var win:NativeWindow = event.target as NativeWindow;
  }
  ```

- Propiedad `nativeWindow` de una página HTML mostrara en la ventana:

  ```
  var win:NativeWindow = htmlLoader.window.nativeWindow;
  ```

- Propiedades `activeWindow` y `openedWindows` del objeto NativeApplication:

  ```
  var nativeWin:NativeWindow = NativeApplication.nativeApplication.activeWindow;
  var firstWindow:NativeWindow = NativeApplication.nativeApplication.openedWindows[0];
  ```

  `NativeApplication.nativeApplication.activeWindow` hace referencia a la ventana activa de una aplicación (pero devuelve `null` si la ventana activa no es una ventana de esta aplicación de AIR). El conjunto `NativeApplication.nativeApplication.openedWindows` contiene todas las ventanas de una aplicación de AIR que no se han cerrado.

Dado que los objetos Application, WindowedApplication y Window de Flex son objetos de visualización a los que es muy sencillo hacer referencia a la ventana de aplicación desde un archivo MXML mediante la propiedad `stage`, como puede verse a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

Nota: *hasta que se añade el componente WindowedApplication o Window al escenario de la ventana por la arquitectura de Flex, la propiedad* `stage` *del componente es* `null`*. Este comportamiento es coherente con el del componente Application de Flex, pero no significa que sea imposible acceder al escenario o a la instancia de NativeWindow en detectores de eventos anteriores al ciclo de inicialización de los componentes WindowedApplication y Window, por ejemplo,* `creationComplete`*. Es seguro acceder al escenario y a la instancia de NativeWindow cuando se distribuye el evento* `applicationComplete`*.*

Activación, visualización y ocultación de ventanas

Adobe AIR 1.0 y posterior

Para activar una ventana, llame al método `activate()` de NativeWindow. Al activar una ventana, esta se visualiza en primer plano, recibe la selección del teclado y del ratón y, si es necesario, se hace visible restaurando la ventana o estableciendo la propiedad `visible` como `true`. Cuando se activa una ventana, esta no cambia el orden del resto de ventanas de la aplicación. Si se llama al método `activate()`, la ventana distribuye un evento `activate`.

Para mostrar una ventana oculta sin activarla, establezca la propiedad `visible` como `true`. Esto llevará la ventana al primer plano, pero no recibirá la selección.

Para ocultar la visibilidad de una ventana, establezca su propiedad `visible` como `false`. Al ocultar una ventana, se suprime la visualización de la ventana, de los iconos relacionados de la barra de tareas y, en Mac OS X, la entrada del menú Ventana.

Cuando se cambia la visibilidad de una ventana, la visibilidad de cualquier ventana contenida en esa ventana también cambia. Por ejemplo, si se oculta una ventana, todas sus ventanas incluidas también se ocultan.

Nota: *en Mac OS X, no es posible ocultar por completo una ventana minimizada que tenga un icono en la parte de venta del dock. Si la propiedad* `visible` *se establece como* `false` *en una ventana minimizada, el icono del Dock de dicha ventana sigue visualizándose. Si el usuario hace clic en el icono, la ventana se restaura con su estado visible y aparece en la pantalla.*

Cambio del orden de visualización de las ventanas

Adobe AIR 1.0 y posterior

AIR proporciona varios métodos para poder cambiar directamente el orden de visualización de las ventanas. Puede colocar una ventana delante del orden de visualización o detrás, mover una ventana encima de otra o detrás. Al mismo tiempo, el usuario puede reordenar las ventanas activándolas.

Puede mantener una ventana delante del resto si establece su propiedad `alwaysInFront` como `true`. Si hay más de una ventana con este ajuste, el orden de visualización de las ventanas se ajustará entre sí, pero siempre se visualizarán delante de ventanas que tengan la propiedad `alwaysInFront` como false.

Las ventanas del grupo superior también se visualizan sobre el resto de las aplicaciones, incluso si la aplicación de AIR no está activa. Como este comportamiento no es el habitual del resto de usuarios, solo se debe establecer `alwaysInFront` como `true` cuando sea realmente necesario. Entre los ejemplos de usos justificados se incluyen:

- Ventanas emergentes temporales para controles como sugerencias, listas, menús personalizados o cuadros combinados. Como estas ventanas se cierran cuando dejan de recibir la selección, está justificado el comportamiento para evitar que el usuario no pueda ver las ventanas.
- Mensajes de error y alertas realmente importantes. Si se produce un cambio irrevocable y el usuario no responde a tiempo, está justificado el comportamiento para poner la alerta en primer plano. Sin embargo, la mayoría de los errores y alertas se pueden gestionar con el orden de visualización normal de las ventanas.
- Ventanas temporales superpuestas.

Nota: *AIR no obliga a utilizar la propiedad `alwaysInFront` correctamente. Si embargo, si su aplicación no se ajusta al flujo normal del resto de usuarios, es muy probable que termine en la papelera de reciclaje.*

Si una ventana contiene otras ventanas, estas siempre se ordenan frente a la misma. Si se llama a `orderToFront()` o se establece `alwaysInFront` en `true` en una ventana que incluya otra ventana, las ventanas incluidas se reordenan junto con la ventana propietaria frente a las demás ventanas, pero las ventanas incluidas aún se muestran frente a la propietaria.

La llamada a los métodos de ordenación de ventanas en las ventanas incluidas funciona normalmente entre las ventanas contenidas en la misma ventana, pero también se puede cambiar el orden de todo el grupo de ventanas incluidas en comparación con las ventanas fuera de ese grupo. Por ejemplo, si se llama a `orderToFront()` en una ventana incluida, tanto esa ventana como su propietaria y cualquier otra ventana incluida por la misma propietaria, se mueven a la parte frontal del orden de visualización de ventanas.

La clase NativeWindow proporciona las siguientes propiedades y métodos para establecer el orden de visualización de una ventana con respecto a otras:

Miembro	**Descripción**
Propiedad alwaysInFront	Especifica si la ventana se muestra en el grupo superior de ventanas. En casi todos los casos, el mejor ajuste es `false`. Si cambia el valor de `false` a `true`, la ventana se coloca delante del resto (aunque no se activa). Si cambia el valor de `true` a `false`, la ventana se coloca detrás del resto de ventanas del grupo superior, aunque delante del resto de ventanas. Si no cambia la propiedad de la ventana, tampoco cambia su orden de visualización. La configuración `alwaysInFront` no se aplica a las ventanas incluidas por otra ventana.
orderToFront()	Trae la ventana al frente.
orderInFrontOf()	Coloca la ventana directamente delante de una ventana concreta.
orderToBack()	Envía la ventana detrás del resto de ventanas.
orderBehind()	Envía la ventana directamente detrás de una ventana concreta.
activate()	Coloca la ventana en primer plano, la hace visible y le asigna la selección.

Nota: *si una ventana está oculta (`visible` es `false`) o minimizada, llamar a los métodos de orden de visualización no produce ningún efecto.*

En el sistema operativo Linux, los diferentes administradores de ventana aplican reglas distintas relacionadas con el orden de visualización de la ventana:

- En algunos administradores, las ventanas de utilidades siempre se muestran delante de las ventanas normales.
- En algunos administradores de ventanas, una ventana de pantalla completa con `alwaysInFront` establecido en `true`, siempre se muestra delante de otras ventanas que también tienen el valor `alwaysInFront` definido como `true`.

Cómo cerrar una ventana

Adobe AIR 1.0 y posterior

Para cerrar una ventana, utilice el método `NativeWindow.close()`.

Al cerrar la ventana, se descarga su contenido, pero si otros objetos tienen referencias a él, este no se elimina. El método `NativeWindow.close()` se ejecuta de forma asíncrona y la aplicación contenida en la ventana se sigue ejecutando durante el proceso de cierre. El método close distribuye un evento close una vez finalizada la operación de cierre. El objeto NativeWindow sigue siendo técnicamente válido, aunque al intentar acceder a la mayoría de propiedades y métodos de una ventana cerrada, se genera un error IllegalOperationError. No es posible volver a abrir una ventana cerrada. Verifique la propiedad `closed` de una ventana para ver si se ha cerrado o no. Simplemente para ocultar la visibilidad de una ventana, establezca la propiedad `NativeWindow.visible` como `false`.

Si la propiedad `Nativeapplication.autoExit` es `true` (valor predeterminado), la aplicación se cerrará al cerrarse la última ventana.

Cualquier ventana que tenga una propietaria, se cierra cuando la propietaria se cierra. Las ventanas contenidas no distribuyen un evento de cierre y, por lo tanto, no pueden evitarlo. Se distribuye un evento close.

Permiso para cancelar operaciones con ventanas

Adobe AIR 1.0 y posterior

Si una ventana utiliza fondo cromático del sistema, es posible cancelar la interacción con el usuario que esté detectando cancelar el comportamiento predeterminado de los eventos adecuados. Por ejemplo, cuando un usuario hace clic en el botón para cerrar el fondo cromático del sistema, se distribuye el evento `closing`. Si algún detector registrado llama al método `preventDefault()` del evento, la ventana no se cierra.

Si una ventana no utiliza fondo cromático, los eventos de notificación de cambios intencionados no se distribuyen automáticamente antes del cambio. Por ello, si llama a métodos para cerrar una ventana, para cambiar su estado o para definir las propiedades de su contorno, el cambio no se puede cancelar. Para avisar a los componentes de la aplicación antes de que se realice un cambio de ventana, la lógica de la aplicación puede distribuir el evento de notificación correspondiente mediante el método `dispatchEvent()` de la ventana.

Por ejemplo, la siguiente lógica implementa un controlador de eventos cancelable para un botón de cierre de una ventana:

```
public function onCloseCommand(event:MouseEvent):void{
    var closingEvent:Event = new Event(Event.CLOSING,true,true);
    dispatchEvent(closing);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

El método `dispatchEvent()` devuelve `false` si un detector llama al método `preventDefault()` del evento. No obstante, también puede devolver `false` por otros motivos. Por ello, es mejor utilizar explícitamente el método `isDefaultPrevented()` para probar si se debe cancelar o no el cambio.

Maximización, minimización y restauración de una ventana

Adobe AIR 1.0 y posterior

Para maximizar la ventana, utilice el método `maximize()` de NativeWindow.

```
myWindow.maximize();
```

Para minimizar la ventana, utilice el método `minimize()` de NativeWindow.

```
myWindow.minimize();
```

Para restaurar la ventana (es decir, hacer que recupere el tamaño que tenía antes de maximizarla o minimizarla), utilice el método `restore()` de NativeWindow.

```
myWindow.restore();
```

Una venta que tenga una propietaria se minimiza y se restaura cuando la ventana propietaria se minimice o se restaure. La ventana incluida no distribuye eventos cuando se minimiza, ya que su propietaria también se minimiza.

Nota: *el comportamiento resultante de maximizar una ventana de AIR es distinto al comportamiento estándar de Mac OS X. En vez de alternar entre el tamaño "estándar" definido por la aplicación y el último tamaño establecido por el usuario, las ventanas de AIR alternan entre el último tamaño establecido por la aplicación y el tamaño completo utilizable de la pantalla.*

En el sistema operativo Linux, los diferentes administradores de ventana aplican reglas distintas relacionadas con la configuración del estado de visualización de la ventana:

- En algunos administradores, las ventanas de utilidades no se puede maximizar.
- Si se define un tamaño máximo para la ventana, algunas ventanas no permiten que se maximice una ventana. Otros administradores de ventanas establecen el estado de visualización como maximizado, pero no cambian el tamaño de la ventana. En cualquier caso, no se distribuye ningún evento de cambio de estado de visualización.
- Algunos administradores de ventanas no aceptan la configuración `maximizable` o `minimizable` de la ventana.

Nota: *en Linux, las propiedades de la ventana se modifican de forma asincrónica. Si se cambia el estado de visualización en una línea del programa y se lee el valor en la siguiente, el valor leído aún reflejará la configuración anterior. En todas las plataformas, el objeto NativeWindow distribuye el evento `displayStateChange` cuando cambia el estado de visualización. Si se necesita realizar alguna operación en función del nuevo estado de la ventana, siempre se debe llevar a cabo en un controlador de eventos `displayStateChange`. Consulte "Detección de eventos de ventanas" en la página 951.*

Ejemplo: Minimizar, maximizar, restaurar y cerrar una ventana

Adobe AIR 1.0 y posterior

La siguiente aplicación corta de MXML muestra los métodos `maximize()`, `minimize()`, `restore()` y `close()` de Window:

```
<?xml version="1.0" encoding="utf-8"?>

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">


    <mx:Script>
    <![CDATA[
    public function minimizeWindow():void
    {
        this.stage.nativeWindow.minimize();
    }

    public function maximizeWindow():void
    {
        this.stage.nativeWindow.maximize();
    }

    public function restoreWindow():void
    {
        this.stage.nativeWindow.restore();
    }

    public function closeWindow():void
    {
        this.stage.nativeWindow.close();
    }
    ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button label="Minimize" click="minimizeWindow()"/>
        <mx:Button label="Restore" click="restoreWindow()"/>
        <mx:Button label="Maximize" click="maximizeWindow()"/>
        <mx:Button label="Close" click="closeWindow()"/>
    </mx:VBox>

</mx:WindowedApplication>
```

El siguiente ejemplo de ActionScript para Flash crea cuatro campos de texto en los que se puede hacer clic y que activan los métodos `minimize()`, `maximize()`, `restore()` y `close()` de NativeWindow:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    public class MinimizeExample extends Sprite
    {
        public function MinimizeExample():void
        {
            var minTextBtn:TextField = new TextField();
            minTextBtn.x = 10;
            minTextBtn.y = 10;
            minTextBtn.text = "Minimize";
            minTextBtn.background = true;
            minTextBtn.border = true;
            minTextBtn.selectable = false;
            addChild(minTextBtn);
            minTextBtn.addEventListener(MouseEvent.CLICK, onMinimize);

            var maxTextBtn:TextField = new TextField();
            maxTextBtn.x = 120;
            maxTextBtn.y = 10;
            maxTextBtn.text = "Maximize";
            maxTextBtn.background = true;
            maxTextBtn.border = true;
            maxTextBtn.selectable = false;
            addChild(maxTextBtn);
            maxTextBtn.addEventListener(MouseEvent.CLICK, onMaximize);

            var restoreTextBtn:TextField = new TextField();
            restoreTextBtn.x = 230;
            restoreTextBtn.y = 10;
            restoreTextBtn.text = "Restore";
            restoreTextBtn.background = true;
            restoreTextBtn.border = true;
            restoreTextBtn.selectable = false;
            addChild(restoreTextBtn);
            restoreTextBtn.addEventListener(MouseEvent.CLICK, onRestore);

            var closeTextBtn:TextField = new TextField();
            closeTextBtn.x = 340;
            closeTextBtn.y = 10;
            closeTextBtn.text = "Close Window";
            closeTextBtn.background = true;
            closeTextBtn.border = true;
            closeTextBtn.selectable = false;
            addChild(closeTextBtn);
```

```
            closeTextBtn.addEventListener(MouseEvent.CLICK, onCloseWindow);
        }
        function onMinimize(event:MouseEvent):void
        {
            this.stage.nativeWindow.minimize();
        }
        function onMaximize(event:MouseEvent):void
        {
            this.stage.nativeWindow.maximize();
        }
        function onRestore(event:MouseEvent):void
        {
            this.stage.nativeWindow.restore();
        }
        function onCloseWindow(event:MouseEvent):void
        {
            this.stage.nativeWindow.close();
        }
    }
}
```

Cambio de tamaño y desplazamiento de una ventana

Adobe AIR 1.0 y posterior

Si una ventana utiliza fondo cromático del sistema, este proporciona controles de arrastre para cambiar el tamaño de la ventana y moverla por el escritorio. Si una ventana no utiliza fondo cromático del sistema, deberá añadir sus propios controles para que el usuario pueda cambiarla de tamaño o desplazarla.

Nota: *para cambiar el tamaño de una ventana, primero debe obtener una referencia a la instancia de NativeWindow. Para obtener más información sobre cómo obtener una referencia a la ventana, consulte* *"Obtención de una instancia de NativeWindow" en la página 941.*

Cambio de tamaño de una ventana

Para que un usuario pueda cambiar el tamaño de una ventana de forma interactiva, utilice el método `startResize()` de NativeWindow. Si se llama a este método desde un evento `mouseDown`, la operación de cambio de tamaño corre a cargo del ratón y finaliza cuando el sistema operativo recibe un evento `mouseUp`. Cuando se llama a `startResize()`, se transfiere un argumento que especifica el borde y la esquina desde los que se puede cambiar el tamaño de la ventana.

Para establecer el tamaño de la ventana mediante programación, defina las propiedades de la ventana `width`, `height` o `bounds` con las dimensiones deseadas. Si se establecen los límites, la posición y el tamaño de la ventana se pueden modificar al mismo tiempo. Sin embargo, no se puede garantizar el orden en que suceden los cambios. Algunos administradores de ventanas de Linux no permiten que las ventanas se amplíen fuera de los límites de la pantalla del escritorio. En estos casos, el tamaño final de la ventana puede estar limitado debido al orden en que se establecen las propiedades, aunque con el resultado goblal de los cambios se podría haber obtenido una ventana válida. Por ejemplo, si se modifica la altura y la posición y de una ventana junto a la parte inferior de la pantalla, puede que no se produzca el cambio completo de altura cuando se aplique el cambio de altura antes del cambio de posición y.

Nota: *en Linux, las propiedades de la ventana se modifican de forma asincrónica. Si cambia el tamaño de una ventana en una línea del programa y lee las dimensiones en la siguiente, aún se reflejará la configuración anterior. En todas las plataformas, el objeto NativeWindow distribuye el evento* `resize` *cuando se cambia el tamaño de la ventana. Si se necesita realizar alguna operación, como distribuir controles en la ventana, en función del nuevo tamaño o estado de la ventana, siempre se debe llevar a cabo en un controlador de eventos* `resize`*. Consulte "Detección de eventos de ventanas" en la página 951.*

El modo de escala del escenario determina cómo se comporta el escenario de la ventana y su contenido cuando se cambia la ventana de tamaño. No olvide que los modos de escala del escenario están diseñados para situaciones (como un navegador web) en las que la aplicación no controla el tamaño ni la relación de aspecto de la visualización. En general, los mejores resultados se obtienen estableciendo la propiedad `scaleMode` como `StageScaleMode.NO_SCALE`. Si quiere escalar también el contenido de la ventana, debe establecer los parámetros `scaleX` y `scaleY` del contenido como respuesta a los cambios de los límites de la ventana.

Desplazamiento de una ventana

Para desplazar una ventana sin cambiar su tamaño, utilice el método `startMove()` de NativeWindow. Al igual que el método `startResize()`, cuando se llama a `startMove()` desde un evento `mouseDown`, el proceso de desplazamiento corre a cargo del ratón y finaliza cuando el sistema operativo recibe un evento `mouseUp`.

Para obtener más información sobre los métodos `startResize()` y `startMove()`, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Para mover una ventana mediante programación, establezca las propiedades `x`, `y` o `bounds` de la ventana en la posición deseada. Si se establecen los límites, la posición y el tamaño de la ventana se pueden modificar al mismo tiempo.

Nota: *en Linux, las propiedades de la ventana se modifican de forma asincrónica. Si se mueve una ventana en una línea del programa y se lee la posición en la siguiente, el valor leído aún reflejará la configuración anterior. En todas las plataformas, el objeto NativeWindow distribuye el evento* `move` *cuando cambia la posición. Si se necesita realizar alguna operación en función de la nueva posición de la ventana, siempre se debe llevar a cabo en un controlador de eventos* `move`*. Consulte "Detección de eventos de ventanas" en la página 951.*

Ejemplo: Cambio de tamaño y desplazamiento de ventanas

Adobe AIR 1.0 y posterior

El siguiente ejemplo muestra como iniciar el campo de tamaño y las operaciones de desplazamiento de una ventana:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.display.NativeWindowResize;

    public class NativeWindowResizeExample extends Sprite
    {
        public function NativeWindowResizeExample():void
        {
            // Fills a background area.
            this.graphics.beginFill(0xFFFFFF);
            this.graphics.drawRect(0, 0, 400, 300);
            this.graphics.endFill();

            // Creates a square area where a mouse down will start the resize.
            var resizeHandle:Sprite =
                createSprite(0xCCCCCC, 20, this.width - 20, this.height - 20);
            resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

            // Creates a square area where a mouse down will start the move.
            var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);
        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite
        {
            var s:Sprite = new Sprite();
            s.graphics.beginFill(color);
            s.graphics.drawRect(0, 0, size, size);
            s.graphics.endFill();
            s.x = x;
            s.y = y;
            this.addChild(s);
            return s;
        }

        public function onStartResize(event:MouseEvent):void
        {
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);
        }

        public function onStartMove(event:MouseEvent):void
        {
            this.stage.nativeWindow.startMove();
        }
    }
}
```

Detección de eventos de ventanas

Adobe AIR 1.0 y posterior

Para detectar los eventos distribuidos por una ventana, debe registrar un detector en la instancia de la ventana. Por ejemplo, para detectar un evento closing, debe registrar un detector con la ventana del modo siguiente.

```
myWindow.addEventListener(Event.CLOSING, onClosingEvent);
```

Cuando se distribuye un evento, la propiedad `target` hace referencia a la ventana que envía el evento.

La mayoría de eventos de ventanas tienen dos mensajes relacionados. El primer mensaje indica que el cambio en la ventana es inminente (se puede cancelar), mientras que el segundo mensaje indica que el cambio ya se ha producido. Por ejemplo, si el usuario hace clic en el botón de cierre de una ventana, de distribuye el mensaje del evento closing. Si ningún detector cancela el evento, la ventana se cierre y el evento close se distribuye a todos los detectores.

Normalmente, los eventos de advertencia como `closing` solo se distribuyen si se utiliza el fondo cromático del sistema para activar un evento. Si se llama al método `close()` de la ventana, por ejemplo, no se distribuye automáticamente el evento `closing` (solo el evento `close`). Sin embargo, es posible construir un objeto de evento de cierre y distribuirlo mediante el método `dispatchEvent()` de ventana.

Los eventos de ventana que distribuye un objeto Event son:

Evento	Descripción
activate	Se distribuye cuando la ventana recibe la selección.
deactivate	Se distribuye cuando la ventana deja de recibir la selección.
closing	Se distribuye cuando la ventana va a cerrarse. Esto solo ocurre automáticamente al pulsar el botón de cierre del fondo cromático o, en Mac OS X, al invocar el comando Salir.
close	Se distribuye cuando la ventana se ha cerrado.

Los eventos de ventana que distribuye un objeto NativeWindowBoundsEvent son:

Evento	Descripción
moving	Si se distribuye inmediatamente antes de que la esquina superior izquierda de la ventana cambie de posición, bien como resultado del desplazamiento, cambio de tamaño o modificación del estado de visualización de la ventana.
move	Se distribuye una vez que la esquina superior izquierda de la ventana ha cambiado de posición.
resizing	Se distribuye inmediatamente antes de que la anchura o la altura de la ventana cambie, bien como resultado del cambio de tamaño o por la modificación del estado de visualización.
resize	Se distribuye después de que la ventana haya cambiado de tamaño.

En eventos NativeWindowBoundsEvent, puede utilizar las propiedades `beforeBounds` y `afterBounds` para determinar los límites de la ventana antes y después el cambio.

Los eventos de ventana que distribuye un objeto NativeWindowDisplayStateEvent son:

Evento	Descripción
displayStateChanging	Se distribuye inmediatamente antes de que cambie el estado de visualización de la ventana.
displayStateChange	Se distribuye una vez ha cambiado el estado de visualización de la ventana.

En eventos NativeWindowDisplayStateEvent, puede utilizar las propiedades `beforeDisplayState` y `afterDisplayState` para determinar el estado de visualización de la ventana antes y después del cambio.

En algunos administradores de ventanas de Linux, no se distribuye ningún evento de cambio de estado de visualización cuando se maximiza una ventana con una configuración de tamaño máximo. (La ventana se establece en el estado de visualización maximizado, pero no se cambia el tamaño.)

Visualización de ventanas a pantalla completa

Adobe AIR 1.0 y posterior

Al establecer la propiedad `displayState` de la clase Stage como `StageDisplayState.FULL_SCREEN_INTERACTIVE` la ventana pasa a modo de pantalla completa. La acción del teclado *sí* esta permitida en este modo. (En contenido SWF ejecutado en un navegador, no está permitida la acción del teclado). Para salir del modo de pantalla completa, el usuario debe pulsar la tecla Esc.

Nota: *algunos administradores de ventanas de Linux no cambiarán las dimensiones de la ventana para que se rellene la pantalla si se establece un tamaño máximo para la ventana (aunque se elimina el fondo cromático del sistema de la ventana).*

Por ejemplo, el siguiente código de Flex define una aplicación sencilla de AIR que configura un terminal simple a pantalla completa:

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to
exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

El siguiente ejemplo de ActionScript para Flash simula un terminal simple de texto a pantalla completa:

```
    import flash.display.Sprite;
    import flash.display.StageDisplayState;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class FullScreenTerminalExample extends Sprite
    {
        public function FullScreenTerminalExample():void
        {
            var terminal:TextField = new TextField();
            terminal.multiline = true;
            terminal.wordWrap = true;
            terminal.selectable = true;
            terminal.background = true;
            terminal.backgroundColor = 0x00333333;

            this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

            addChild(terminal);
            terminal.width = 550;
            terminal.height = 400;

            terminal.text = "Welcome to the dumb terminal application. Press the ESC key to
exit.\n_";

            var tf:TextFormat = new TextFormat();
            tf.font = "Courier New";
            tf.color = 0x00CCFF00;
            tf.size = 12;
            terminal.setTextFormat(tf);

            terminal.setSelection(terminal.text.length - 1, terminal.text.length);
        }
    }
```

Capítulo 53: Pantallas en AIR

Adobe AIR 1.0 y posterior

Para obtener información sobre las pantallas conectadas a un ordenador o dispositivo, utilice la clase Screen de Adobe® AIR®.

Más temas de ayuda

flash.display.Screen

Aspectos básicos de las pantallas en AIR

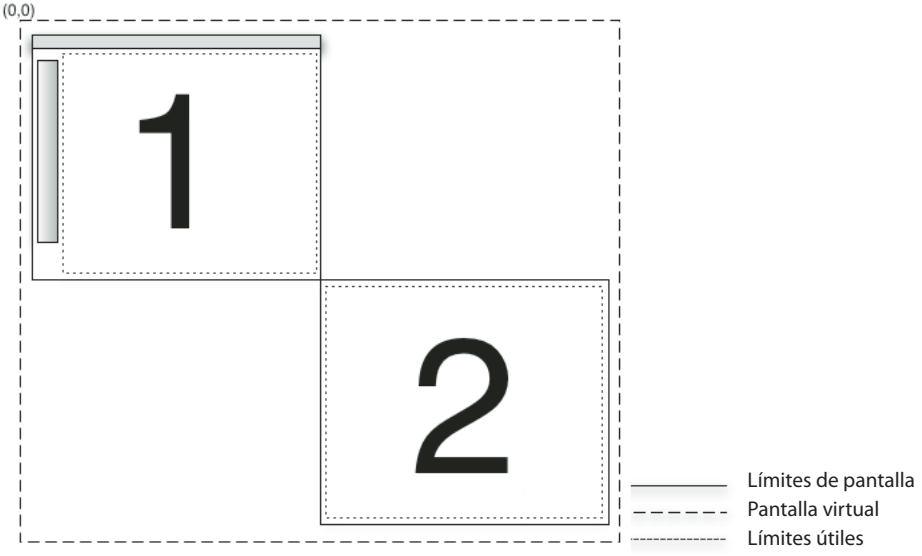
Adobe AIR 1.0 y posterior

- Measuring the virtual desktop (Medición del escritorio virtual, en inglés) (Flex)
- Measuring the virtual desktop (Medición del escritorio virtual, en inglés) (Flash)

La API de la pantalla contiene una sola clase, Screen, que proporciona miembros estáticos para obtener información sobre las pantallas del sistema y miembros de instancia para describir una pantalla en particular.

Un sistema informático puede tener conectados varios monitores que se corresponden con diversas pantallas de escritorio dispuestas en un espacio virtual. La clase Screen de AIR facilita información sobre las pantallas, la disposición y el espacio aprovechable de las mismas. Si hay más de un monitor que se corresponde con la misma pantalla, es que existe una sola pantalla. Si el tamaño de la pantalla es superior a la superficie de visualización del monitor, no hay manera de determinar qué parte de la pantalla está visible en este momento.

Una pantalla representa una superficie independiente de visualización del escritorio. Las pantallas se describen como rectángulos dentro del escritorio virtual. El punto cero del sistema de coordenadas del escritorio virtual es el ángulo superior izquierdo de la pantalla designada como pantalla principal. Todos los valores que se utilizan para describir una pantalla se expresan en píxeles.



En esta disposición de pantallas, hay dos pantallas en el escritorio virtual. Las coordenadas del ángulo superior izquierdo de la pantalla principal (nº 1) son siempre (0,0). Si se modifica la disposición de pantallas para designar la pantalla nº 2 como pantalla principal, las coordenadas de la pantalla nº 1 pasan a ser cifras negativas. Las barras de menús, las barras de tareas y los docks se excluyen al notificar los límites utilizables para una pantalla.

Para obtener más información sobre la clase de API de pantalla, sus métodos, propiedades y eventos, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Enumeración de las pantallas

Adobe AIR 1.0 y posterior

Las pantallas del escritorio virtual se pueden enumerar utilizando los siguientes métodos y propiedades de pantalla:

Método o propiedad	Descripción
Screen.screens	Proporciona un conjunto de objetos Screen que describen las pantallas disponibles. El orden del conjunto no es relevante.
Screen.mainScreen	Proporciona un objeto Screen para la pantalla principal. En Mac OS X la pantalla principal es la que contiene la barra de menús. En Windows la pantalla principal es la designada como tal por el sistema.
Screen.getScreensForRectangle()	Proporciona un conjunto de objetos Screen que describen las pantallas a las que cruza el rectángulo determinado. El rectángulo que se pasa a este método tiene las coordenadas en píxeles en el escritorio virtual. Si el rectángulo no forma intersección con ninguna pantalla, el conjunto estará vacío. Este método puede emplearse para averiguar en qué pantallas se muestra una ventana.

Los valores que producen los métodos y propiedades de la clase Screen no deben guardarse. El usuario o el sistema operativo puede modificar en cualquier momento las pantallas disponibles y la disposición de las mismas.

En el ejemplo siguiente se utiliza la API de la pantalla para desplazar una ventana entre varias pantallas en función de la pulsación de las teclas de flecha. Para desplazar la ventana a la siguiente pantalla, el ejemplo obtiene el conjunto `screens` y lo ordena en sentido vertical u horizontal (dependiendo de la tecla de flecha que se pulse). El código pasa por el conjunto ordenado y compara cada pantalla con las coordenadas de la pantalla actual. Para identificar la pantalla actual de la ventana, el ejemplo llama a `Screen.getScreensForRectangle()` y se pasan los límites de la ventana.

```
package {
    import flash.display.Sprite;
    import flash.display.Screen;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;

    public class ScreenExample extends Sprite
    {
        public function ScreenExample()
        {
                stage.align = StageAlign.TOP_LEFT;
                stage.scaleMode = StageScaleMode.NO_SCALE;

                stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);
        }

        private function onKey(event:KeyboardEvent):void{
            if(Screen.screens.length > 1){
                switch(event.keyCode){
                    case Keyboard.LEFT :
                        moveLeft();
                        break;
                    case Keyboard.RIGHT :
                        moveRight();
                        break;
                    case Keyboard.UP :
                        moveUp();
                        break;
                    case Keyboard.DOWN :
                        moveDown();
                        break;
                }
            }
        }

        private function moveLeft():void{
            var currentScreen = getCurrentScreen();
            var left:Array = Screen.screens;
            left.sort(sortHorizontal);
            for(var i:int = 0; i < left.length - 1; i++){
                if(left[i].bounds.left < stage.nativeWindow.bounds.left){
                    stage.nativeWindow.x +=
                        left[i].bounds.left - currentScreen.bounds.left;
                    stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
                }
            }
        }

        private function moveRight():void{
            var currentScreen:Screen = getCurrentScreen();
            var left:Array = Screen.screens;
            left.sort(sortHorizontal);
            for(var i:int = left.length - 1; i > 0; i--){
                if(left[i].bounds.left > stage.nativeWindow.bounds.left){
                    stage.nativeWindow.x +=
```

```
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveUp():void{
        var currentScreen:Screen = getCurrentScreen();
        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = 0; i < top.length - 1; i++){
            if(top[i].bounds.top < stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function moveDown():void{
        var currentScreen:Screen = getCurrentScreen();

        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = top.length - 1; i > 0; i--){
            if(top[i].bounds.top > stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function sortHorizontal(a:Screen,b:Screen):int{
        if (a.bounds.left > b.bounds.left){
            return 1;
        } else if (a.bounds.left < b.bounds.left){
            return -1;
        } else {return 0;}
    }

    private function sortVertical(a:Screen,b:Screen):int{
        if (a.bounds.top > b.bounds.top){
            return 1;
        } else if (a.bounds.top < b.bounds.top){
            return -1;
        } else {return 0;}
    }

    private function getCurrentScreen():Screen{
        var current:Screen;
        var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
        (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
        return current;
    }
  }
}
```

Capítulo 54: Impresión

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los motores de ejecución de Flash (como Adobe® Flash® Player y Adobe® AIR™) pueden comunicarse con la interfaz de impresión de un sistema operativo para poder pasar páginas a la cola de impresión. Cada página enviada a la cola puede incluir contenido visible, dinámico o fuera de la pantalla para el usuario, incluidos valores de base de datos y texto dinámico. Asimismo, se establecen las propiedades de la clase flash.printing.PrintJob en función de la configuración de impresora del usuario, por lo que puede aplicar un formato adecuado a las páginas.

En este capítulo se detallan las estrategias para utilizar los métodos y las propiedades de la clase flash.printing.PrintJob para crear un trabajo de impresión, leer la configuración de impresión de un usuario y realizar ajustes en un trabajo de impresión basándose en la respuesta del motor de ejecución de Flash y del sistema operativo del usuario.

Fundamentos de impresión

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En ActionScript 3.0 se utiliza la clase PrintJob para crear instantáneas de contenido de la pantalla con el fin de convertirlas en las representaciones en tinta y papel de una copia impresa. En cierto modo, configurar contenido para imprimir es lo mismo que configurarlo para la visualización en pantalla: se disponen los elementos y se ajusta su tamaño para crear el diseño deseado. Sin embargo, la impresión tiene peculiaridades que la distinguen de la visualización en pantalla. Por ejemplo, la resolución de las impresoras es distinta de la de los monitores de ordenador, el contenido de una pantalla de ordenador es dinámico y puede cambiar mientras que el contenido impreso es estático, y al planificar la impresión hay que tener en cuenta las restricciones de un tamaño fijo de página y la posibilidad de la impresión de varias páginas.

Aunque estas diferencias pueden parecer obvias, es importante tenerlas en cuenta al configurar la impresión con ActionScript. La precisión de la impresión depende de una combinación de los valores especificados por el programador y las características de la impresora del usuario. La clase PrintJob incluye propiedades que permiten determinar las características importantes de la impresora del usuario.

Conceptos y términos importantes

La siguiente lista de referencia contiene términos importantes relacionados con la impresión:

Cola de impresión Parte del sistema operativo o software controlador de impresora que hace un seguimiento de las páginas en espera de ser impresas y las envía a la impresora cuando esta disponible.

Orientación de la página Rotación del contenido impreso con relación al papel (horizontal o vertical).

Trabajo de impresión Página o conjunto de páginas que constituyen una impresión individual.

Impresión de una página

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para controlar la impresión se puede utilizar una instancia de la clase PrintJob. Para imprimir una página básica a través de Flash Player o AIR, se debe utilizar la siguiente secuencia de cuatro sentencias:

- `new PrintJob()`: crea una nueva instancia de trabajo de impresión con el nombre especificado por el usuario.
- `PrintJob.start()`: inicia el proceso de impresión en el sistema operativo, llama al cuadro de diálogo de impresión para el usuario y llena los valores de las propiedades de solo lectura del trabajo de impresión.
- `PrintJob.addPage()`: contiene los detalles relativos al contenido del trabajo de impresión, como el objeto Sprite (y sus elementos secundarios), el tamaño del área de impresión y si la impresora debe imprimir la imagen como un vector o un mapa de bits. Se pueden utilizar llamadas sucesivas a `addPage()` para imprimir varios objetos Sprite en varias páginas.
- `PrintJob.send()`: envía las páginas a la impresora del sistema operativo.

Por ejemplo, un script de un trabajo de impresión sencillo es el siguiente (incluyendo las sentencias `package`, `import` y`class` para la compilación):

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

Nota: *este ejemplo pretende mostrar los elementos básicos de un script de trabajo de impresión y no contiene gestión de errores. Para generar un script que responda adecuadamente a la cancelación de un trabajo de impresión por parte del usuario, consulte* *"Trabajo con excepciones y valores devueltos" en la página 960.*

Si es necesario borrar las propiedades de un objeto PrintJob por algún motivo, debe establecerse la variable PrintJob en `null` (por ejemplo, `myPrintJob = null`).

Interfaz de impresión del sistema y tareas del motor de ejecución de Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Debido a que los motores de ejecución de Flash distribuyen páginas a la interfaz del impresión del sistema operativo, deben conocerse las tareas administradas por ambas aplicaciones y las tareas administradas por la interfaz de impresión del sistema operativo. Los motores de ejecución de Flash pueden iniciar un trabajo de impresión, leer parte de la configuración de página de una impresora, pasar el contenido de un trabajo de impresión al sistema operativo y verificar si el usuario o el sistema han cancelado un trabajo de impresión. Otros procesos, como mostrar los cuadros de diálogo específicos de la impresora, cancelar un trabajo de impresión en cola o notificar el estado de la impresora son tareas que gestiona el sistema operativo. Los motores de ejecución de Flash pueden responder si hay un problema para iniciar o dar formato a un trabajo de impresión, pero solo pueden ofrecer información sobre determinadas propiedades o condiciones de la interfaz de usuario del sistema operativo. Como desarrollador, su código debe poder responder a estas propiedades o condiciones.

Trabajo con excepciones y valores devueltos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si el usuario ha cancelado el trabajo de impresión, hay que comprobar si el método `PrintJob.start()` devuelve `true` antes de ejecutar llamadas a `addPage()` y `send()`. Una forma sencilla de comprobar si se han cancelado estos métodos antes de continuar es incluirlos en una sentencia `if`, tal y como se muestra en el siguiente ejemplo:

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Si `PrintJob.start()` es `true`, el usuario seleccionó Print (o un motor de ejecución de Flash, como as Flash Player o AIR, ha iniciado un comando Print). Por lo tanto, se puede llamar a los métodos `addPage()` y `send()`.

Asimismo, para ayudar a administrar el proceso de impresión, los motores de ejecución de Flash emiten excepciones para el método `PrintJob.addPage()`, de forma que es posible capturar errores y proporcionar información y opciones al usuario. Si un método `PrintJob.addPage()` no se ejecuta correctamente, se puede llamar a otra función o detener el trabajo de impresión actual. Estas excepciones se pueden capturar incorporando llamadas `addPage()` en una sentencia `try..catch`, tal y como se muestra en el siguiente ejemplo. En el ejemplo, `[params]` es un marcador de posición para los parámetros, que especifica el contenido real que se desea imprimir:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

Cuando se inicia el trabajo de impresión, se puede añadir contenido mediante `PrintJob.addPage()` y ver si se genera una excepción (por ejemplo, si el usuario ha cancelado el trabajo de impresión). Si es así, se puede añadir lógica a la sentencia `catch` para proporcionar al usuario (o el motor de ejecución de Flash) información y opciones, o bien, se puede detener el trabajo de impresión actual. Si se añade la página correctamente, se puede continuar enviando las páginas a la impresora mediante `PrintJob.send()`.

Si los motores de ejecución de Flash encuentran un problema al enviar el trabajo de impresión a la impresora (por ejemplo, si la impresora está sin conexión), se puede capturar también dicha excepción y proporcionar al usuario información o más opciones (por ejemplo, mostrar el texto de un mensaje o proporcionar una alerta en una animación). Por ejemplo, puede asignar texto nuevo a un campo de texto en una sentencia `if..else`, tal y como se indica en el siguiente código:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Para ver ejemplos de uso, consulte “Ejemplo de impresión: Ajuste de escala, recorte y respuesta” en la página 970.

Trabajo con las propiedades de página

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando el usuario hace clic en Aceptar en el cuadro de diálogo Imprimir y `PrintJob.start()` devuelve `true`, se puede acceder a las propiedades definidas por la configuración de la impresora. Estas propiedades se refieren a la anchura del papel, la altura del papel (`pageWidth` y `pageHeight`) y a la orientación del contenido en el papel. Debido a que se trata de la configuración de impresora, no controlada por el motor de ejecución de Flash, no es posible modificarla; sin embargo, se puede utilizar para alinear el contenido enviado a la impresora para que coincida con la configuración actual. Para obtener más información, consulte “Configuración del tamaño, la escala y la orientación” en la página 963.

Configuración de la representación vectorial o de mapa de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Es posible configurar manualmente el trabajo de impresión para que cada una de las páginas se coloque en cola como un gráfico vectorial o una imagen de mapa de bits. En algunos casos, la impresión vectorial producirá un archivo de cola más pequeño y una imagen mejor que la impresión de mapa de bits. Sin embargo, si el contenido incluye una imagen de mapa de bits y se desea conservar la transparencia alfa o los efectos de color, se debe imprimir la página como una imagen de mapa de bits. Además, las impresoras que no son PostScript convierten automáticamente los gráficos vectoriales en imágenes de mapa de bits.

Para especificar la impresión de mapa de bits, pase un objeto PrintJobOptions como tercer parámetro de `PrintJob.addPage()`.

Para Flash Player y AIR anterior a AIR 2, defina el parámetro `printAsBitmap` del objeto PrintJobOptions como `true` del modo siguiente:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Si no se especifica un valor para el tercer parámetro, el trabajo de impresión utilizará el valor predeterminado, que es la impresión vectorial.

Para AIR 2 y posterior, utilice la propiedad `printMethod` del objeto PrintJobOptions para especificar el método de impresión. Esta propiedad acepta tres valores que se definen como constantes en la clase PrintMethod:

- `PrintMethod.AUTO`: selecciona automáticamente el mejor método de impresión basándose en el contenido que se va a imprimir. Por ejemplo, si la página contiene texto, se elige el método de impresión vectorial. Sin embargo, si hay una imagen con marca de agua y transparencia alfa sobre el texto, se opta por la impresión de mapa de bits para conservar la transparencia.
- `PrintMethod.BITMAP`: fuerza la impresión de mapa de bits sin importar el contenido
- `PrintMethod.VECTOR`: fuerza la impresión vectorial sin importar el contenido

Sincronización de las sentencias del trabajo de impresión

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

ActionScript 3.0 no restringe un objeto PrintJob a un solo fotograma (como ocurría en versiones anteriores de ActionScript). Sin embargo, como el sistema operativo muestra al usuario información del estado de la impresión después de que el usuario haya hecho clic en el botón Aceptar del cuadro de diálogo Imprimir, debería llamarse a `PrintJob.addPage()` y `PrintJob.send()`, en cuanto sea posible, para enviar páginas a la cola. Una demora al llegar al fotograma que contiene la llamada a `PrintJob.send()` retrasará el proceso de impresión.

En ActionScript 3.0 hay un límite de tiempo de espera de script de 15 segundos. Por lo tanto, el tiempo entre cada sentencia principal de una secuencia de un trabajo de impresión no puede superar los 15 segundos. Dicho de otro modo, el límite de tiempo de espera del script de 15 segundos se aplica a los siguientes intervalos:

- Entre `PrintJob.start()` y el primer `PrintJob.addPage()`
- Entre `PrintJob.addPage()` y el siguiente `PrintJob.addPage()`
- Entre el último `PrintJob.addPage()` y `PrintJob.send()`

Si alguno de los intervalos anteriores dura más de 15 segundos, la siguiente llamada a `PrintJob.start()` en la instancia de PrintJob devolverá `false` y el siguiente `PrintJob.addPage()` de la instancia de PrintJob hará que Flash Player o AIR emitan una excepción de tiempo de ejecución.

Configuración del tamaño, la escala y la orientación

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En la sección "Impresión de una página" en la página 959 se detallan los pasos de un trabajo de impresión básico, en el que la salida refleja directamente el equivalente impreso del tamaño de pantalla y la posición del objeto Sprite especificado. Sin embargo, las impresoras utilizan resoluciones distintas para imprimir y pueden tener configuraciones que afecten negativamente al aspecto del objeto Sprite impreso.

Los motores de ejecución de Flash pueden leer la configuración de impresión de un sistema operativo, pero se debe tener en cuenta que estas propiedades son de solo lectura: aunque se puede responder a sus valores, estos no pueden definirse. Así pues, por ejemplo, se puede buscar la configuración de tamaño de página de la impresora y ajustar el contenido a dicho tamaño. También se puede determinar la configuración de márgenes y la orientación de página de una impresora. Para responder a la configuración de la impresora, es posible que sea necesario especificar un área de impresión, ajustar la diferencia entre la resolución de una pantalla y las medidas de puntos de una impresora, o transformar el contenido para ajustarlo a la configuración de tamaño u orientación de la impresora del usuario.

Uso de rectángulos en el área de impresión

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `PrintJob.addPage()` permite especificar la región de un objeto Sprite que se desea imprimir. El segundo parámetro, `printArea` tiene la forma de un objeto Rectangle. Hay tres formas posibles de proporcionar un valor para este parámetro:

- Crear un objeto Rectangle con propiedades específicas y luego utilizar el rectángulo en la llamada a `addPage()`, como en el siguiente ejemplo:

  ```
  private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
  myPrintJob.addPage(sheet, rect1);
  ```

- Si no se ha especificado todavía un objeto Rectangle, puede hacerse en la propia llamada, como en el siguiente ejemplo:

  ```
  myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
  ```

- Si se ha previsto proporcionar valores para el tercer parámetro en la llamada a `addPage()`, pero no se desea especificar un rectángulo, se puede utilizar `null` para el segundo parámetro, como en el siguiente ejemplo:

  ```
  myPrintJob.addPage(sheet, null, options);
  ```

Comparación de puntos y píxeles

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La anchura y la altura de un rectángulo son valores expresados en píxeles. Una impresora utiliza los puntos como unidad de medida de impresión. Los puntos tienen un tamaño físico fijo (1/72 pulgadas), pero el tamaño de un píxel en pantalla depende de la resolución de cada pantalla. Así, la relación de conversión entre píxeles y puntos depende de la configuración de la impresora y del hecho de que el objeto Sprite tenga ajustada la escala. Un objeto Sprite sin escalar de 72 píxeles de ancho se imprimirá con una anchura de una pulgada, cada punto equivaldrá a un píxel, independientemente de la resolución de la pantalla.

Puede utilizar las equivalencias siguientes para convertir los valores en pulgadas o centímetros en twips o puntos (un twip es 1/20 de un punto):

- 1 punto = 1/72 pulgadas =20 twips
- 1 pulgada = 72 puntos = 1440 twips
- 1 centímetro = 567 twips

Si se omite o pasa de forma incorrecta el parámetro `printArea`, se imprime el área completa del objeto Sprite.

Escala

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si desea realizar un ajuste de escala en un objeto Sprite antes de imprimirlo, establezca las propiedades de escala (consulte "Manipulación del tamaño y ajuste de escala de los objetos" en la página 184) antes de llamar al método `PrintJob.addPage()` y restablezca los valores originales después de imprimir. La escala de un objeto Sprite no tiene relación con la propiedad `printArea`. En otras palabras, si se especifica un área de impresión de 50 por 50 píxeles, se imprimen 2500 píxeles. Si se cambia la escala del objeto Sprite, se seguirán imprimiendo 2500 píxeles, pero el objeto Sprite se imprimirá con el tamaño con la escala ajustada.

Para ver un ejemplo, consulte "Ejemplo de impresión: Ajuste de escala, recorte y respuesta" en la página 970.

Impresión de la orientación horizontal o vertical

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player y AIR pueden detectar la configuración de la orientación, por lo que se puede generar lógica en el código ActionScript para ajustar el tamaño del contenido o la rotación como respuesta a la configuración de la impresora, como se muestra en el siguiente ejemplo:

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

Nota: *si va a leer la configuración del sistema para la orientación de contenido en papel, recuerde importar la clase PrintJobOrientation. La clase PrintJobOrientation proporciona valores constantes que definen la orientación del contenido en la página. La case se importa utilizando la siguiente instrucción:*

```
import flash.printing.PrintJobOrientation;
```

Respuesta a la altura y anchura de la página

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

A través de una estrategia similar a la gestión de la configuración de orientación de la impresora, se puede leer la configuración de altura y anchura de la página, y responder a sus valores mediante la incorporación de lógica en una sentencia `if`. El siguiente código muestra un ejemplo:

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

Además, la configuración de márgenes de una página puede determinarse comparando las dimensiones de la página y del papel, como se muestra en el siguiente ejemplo:

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Técnicas de impresión avanzada

Adobe AIR 2 y posterior

Desde Adobe AIR 2, la clase PrintJob cuenta con propiedades y métodos adicionales, y se admiten tres clases más: PrintUIOptions, PaperSize y PrintMethod. Estos cambios permiten flujos adicionales de trabajo de impresoras y dan a los creadores más control sobre el proceso de impresión. Los cambios incluyen:

- Diálogos de configuración de página: se pueden ver diálogos estándar y personalizados para la configuración de página. El usuario puede definir rangos de páginas, tamaño del papel, orientación y escala antes de imprimir.
- Vista de impresión: es posible crear un modo de visualización para mostrar con precisión el tamaño del papel, los márgenes y la posición del contenido en la página.
- Impresión restringida: los autores pueden restringir las opciones de impresión, como el rango de páginas imprimibles.
- Opciones de calidad: los autores pueden ajustar la calidad de impresión de un documento y permitir que los usuarios puedan seleccionar la resolución y las opciones de color.
- Multisesiones de impresión: ahora es posible utilizar una sola instancia de PrintJob para multisesiones de impresión. Las aplicaciones pueden proporcionar ajustes coherentes cada vez que se muestran los diálogos de configuración de página e impresión.

Cambios del flujo de trabajo de impresión

El nuevo flujo de trabajo de impresión está formado por estos pasos:

- `new PrintJob()`: crea una instancia de PrintJob (o reutiliza una existente). Muchas propiedades y métodos PrintJob nuevos, como `selectPaperSize()`, están disponibles antes de que se inicie el trabajo de impresión o durante el proceso.
- `PrintJob.showPageSetupDialog()`: (opcional) muestra el diálogo Configurar página sin iniciar un trabajo de impresión.
- `PrintJob.start()` o `PrintJob.start2()`: además del método `start()`, el método `start2()` se utiliza para iniciar el proceso de cola de impresión. El método `start2()` permite elegir si se muestra el diálogo Imprimir y personalizarlo en caso de que se visualice.
- `PrintJob.addPage()`: añade contenido al trabajo de impresión. No cambia en el proceso existente.
- `PrintJob.send()` o `PrintJob.terminate()`: envía las páginas a la impresora seleccionada o finaliza el trabajo sin enviar nada. Los trabajos de impresión finalizan como respuesta a un error. Si se cancela una instancia de PrintJob, puede reutilizarse. Independientemente de si el trabajo de impresión se ha enviado a la impresora o se ha cancelado, se conserva la configuración actual de la impresora para cuando reutilice la instancia de PrintJob.

Cuadro de diálogo de configuración de página

El método `showPageSetupDialog()` muestra el diálogo Configurar página del sistema operativo si el entorno actual así lo admite. Compruebe siempre la propiedad `supportsPageSetupDialog` antes de llamar a este método. Aquí tiene un sencillo ejemplo:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
//check for static property supportsPageSetupDialog of PrintJob class
if (PrintJob.supportsPageSetupDialog) {
    myPrintJob.showPageSetupDialog();
}
```

Opcionalmente, se puede llamar a este método con una propiedad de la clase PrintUIOptions para controlar qué opciones aparecen en el diálogo Configuración de página. Es posible establecer los números de página máx. y mín. El siguiente ejemplo limita la impresión a las tres primeras páginas:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
if (PrintJob.supportsPageSetupDialog) {
    var uiOpt:PrintUIOptions = new PrintUIOptions();
    uiOpt.minPage = 1;
    uiOpt.maxPage = 3;
    myPrintJob.showPageSetupDialog(uiOpt);
}
```

Modificación de la configuración de impresión

Los parámetros de una instancia de PrintJob se pueden cambiar el cualquier momento tras su construcción. Esto incluye el cambio de ajustes entre llamadas `addPage()` y una vez enviado o finalizado el trabajo de impresión. Algunos ajustes, como la propiedad `printer`, se aplican a todo el trabajo de impresión, no a páginas por separado. La configuración deben establecerse antes de llamar a `start()` o a `start2()`.

Se puede llamar al método `selectPaperSize()` para establecer el tamaño de papel predeterminado en los diálogo Configurar página e Imprimir. También se puede llamar durante un trabajo de impresión para establecer el tamaño del papel en un rango de páginas. Se le llama utilizando constantes definidas en la clase `PaperSize`, como en este ejemplo, que selecciona un tamaño de sobre 10:

```
import flash.printing.PrintJob;
import flash.printing.PaperSize;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.selectPaperSize(PaperSize.ENV_10);
```

Utilice la propiedad `printer` para obtener o definir el nombre de la impresora del trabajo de impresión actual. De forma predeterminada, se establece como el nombre de la impresora predeterminada. La propiedad `printer` es `null` si no hay ninguna impresora disponible o si el sistema no admite impresión. Para cambiar la impresora, primero obtenga la lista de impresoras disponibles mediante la propiedad `printers`. Dicha propiedad es un vector cuyos elementos de cadena son nombres de impresoras disponibles. Defina la propiedad `printer` como uno de los valores de cadena que hacen que la impresora deseada sea la activa. La propiedad `printer` de un trabajo de impresión activo no se puede cambiar. Intente cambiarlo tras una llamada correcta a `start()` o `start2()`, y antes de que se envíe el trabajo o de que finalice. Aquí tiene un ejemplo para establecer esta propiedad:

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.printer = "HP_LaserJet_1";
myPrintJob.start();
```

La propiedad `copies` obtiene el valor del número de copias establecido en el diálogo Imprimir del sistema operativo. Las propiedades `firstPage` y `lastPage` obtienen el rango de páginas. La propiedad `orientation` obtiene el parámetro de orientación del papel. Estas propiedades se pueden establecer para que anulen los valores del diálogo Imprimir. El siguiente ejemplo establece estas propiedades:

```
import flash.printing.PrintJob;
import flash.printing.PrintJobOrientation;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.copies = 3;
myPrintJob.firstPage = 1;
myPrintJob.lastPage = 3;
myPrintJob.orientation = PrintJobOrientation.LANDSCAPE;
```

Los siguientes ajustes de solo lectura asociados a `PrintJob` proporcionan información útil sobre la configuración de la impresora actual:

- `paperArea`: límites rectangulares del medio de impresión, en puntos.
- `printableArea`: límites rectangulares del área de impresión, en puntos.
- `maxPixelsPerInch`: resolución física de la impresora actual, en píxeles por pulgada.
- `isColor`: capacidad de la impresora actual para imprimir en color (devuelve `true` si la impresora actual puede imprimir en color).

Consulte “Ejemplo de impresión: Configuración de página y opciones de impresión” en la página 971.

Ejemplo de impresión: Impresión de varias páginas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se imprime más de una página de contenido, se puede asociar cada una de las páginas a un objeto Sprite distinto (en este caso, `sheet1` y `sheet2`) y luego utilizar `PrintJob.addPage()` en cada uno de los objetos Sprite. El siguiente código muestra esta técnica:

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
            createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
            sheet2 = new Sprite();
            createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe
end.", null);
        }

        private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
        {
            sheet.graphics.beginFill(0xEEEEEE);
            sheet.graphics.lineStyle(1, 0x000000);
            sheet.graphics.drawRect(0, 0, 100, 200);
            sheet.graphics.endFill();

            var txt:TextField = new TextField();
            txt.height = 200;
            txt.width = 100;
            txt.wordWrap = true;
            txt.text = str;

            if (imgValue != null)
            {
                var img:Sprite = new Sprite();
                img.graphics.beginFill(0xFFFFFF);
                img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
                img.graphics.endFill();
                sheet.addChild(img);
            }
            sheet.addChild(txt);
        }

        private function printPages():void
        {
            var pj:PrintJob = new PrintJob();
```

```
            var pagesToPrint:uint = 0;
            if (pj.start())
            {
                if (pj.orientation == PrintJobOrientation.LANDSCAPE)
                {
                    throw new Error("Page is not set to an orientation of portrait.");
                }

                sheet1.height = pj.pageHeight;
                sheet1.width = pj.pageWidth;
                sheet2.height = pj.pageHeight;
                sheet2.width = pj.pageWidth;

                try
                {
                    pj.addPage(sheet1);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                // Respond to error.
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

Ejemplo de impresión: Ajuste de escala, recorte y respuesta

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En algunos casos, es posible que se desee ajustar el tamaño (u otras propiedades) de un objeto de visualización cuando se imprime, con el fin de ajustar las diferencias entre su aspecto en la pantalla y su aspecto impreso en papel. Cuando se ajustan las propiedades de un objeto de visualización antes de imprimir (por ejemplo, usando las propiedades `scaleX` y `scaleY`), hay que tener en cuenta que si se ajusta la escala del objeto más allá del rectángulo definido en el área de impresión, el objeto quedará recortado. También es probable que se deseen restablecer las propiedades después de imprimir las páginas.

El siguiente código ajusta la escala de las dimensiones del objeto de visualización `txt` (pero no el fondo del cuadro verde) y el campo de texto queda recortado por las dimensiones del rectángulo especificado. Después de imprimir, el campo de texto recupera su tamaño original de visualización en la pantalla. Si el usuario cancela el trabajo de impresión desde el cuadro Imprimir del sistema operativo, el contenido del motor de ejecución de Flash cambia para avisar al usuario de que se ha cancelado el trabajo.

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
```

```
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

Ejemplo de impresión: Configuración de página y opciones de impresión

Adobe AIR 2 y posterior

El siguiente ejemplo inicializa los ajustes de PrintJob para número de copias, tamaño de papel (legal) y orientación de página (horizontal). Obliga a que el cuadro de diálogo Configurar página se muestre primero y que después se inicie el trabajo de impresión desde el diálogo Imprimir.

```
package
{
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.printing.PaperSize;
    import flash.printing.PrintUIOptions;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintAdvancedExample extends Sprite
    {
        private var bg:Sprite = new Sprite();
        private var txt:TextField = new TextField();
        private var pj:PrintJob = new PrintJob();
        private var uiOpt:PrintUIOptions = new PrintUIOptions();

        public function PrintAdvancedExample():void
        {
            initPrintJob();
            initContent();
            draw();
            printPage();
        }

        private function printPage():void
        {
            //test for dialog support as a static property of PrintJob class
            if (PrintJob.supportsPageSetupDialog)
            {
                pj.showPageSetupDialog();
            }
            if (pj.start2(uiOpt, true))
            {
                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job terminated";
                pj.terminate();
            }
        }

        private function initContent():void
        {
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();
```

```
            txt.border = true;
            txt.text = "Hello World";
        }

        private function initPrintJob():void
        {
            pj.selectPaperSize(PaperSize.LEGAL);
            pj.orientation = PrintJobOrientation.LANDSCAPE;
            pj.copies = 2;
            pj.jobName = "Flash test print";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

Capítulo 55: Localización geográfica

Si un dispositivo admite geolocalización, puede utilizar la API de geolocalización para obtener la ubicación geográfica actual del dispositivo. Si el dispositivo admite esta función, puede obtener información de geolocalización. Esta información incluye altitud, precisión, dirección, velocidad y marca de hora de los últimos cambios en su posición.

La clase Geolocation distribuye eventos `update` como respuesta al sensor de posición del dispositivo. El evento `update` es un objeto GeolocationEvent.

Más temas de ayuda

flash.sensors.Geolocation

flash.events.GeolocationEvent

Detección de cambios de geolocalización

Para utilizar el sensor de geolocalización, cree una instancia del objeto Geolocation y registre los eventos `update` que distribuya. El evento `update` es un objeto de evento Geolocation. El evento tiene ocho propiedades:

- `altitude`: la altitud, en metros.
- `heading`: la dirección de movimiento (con respecto al norte verdadero), en grados.
- `horizontalAccuracy`: la precisión horizontal, en metros.
- `latitude`: la latitud, en grados.
- `longitude`: la longitud, en grados.
- `speed`: la velocidad, en metros por segundo.
- `timestamp`: el número de milisegundos en el momento del evento desde que se inicializó el motor de ejecución.
- `verticalAccuracy`: la precisión vertical, en metros.

La propiedad `timestamp` es un objeto int. El resto son objetos Number.

A continuación incluimos un ejemplo básico que muestra datos de geolocalización en un campo de texto:

```
var geo:Geolocation;
if (Geolocation.isSupported)
{
    geo = new Geolocation();
    geo.addEventListener(GeolocationEvent.UPDATE, updateHandler);
}
else
{
    geoTextField.text = "Geolocation feature not supported";
}
function updateHandler(event:GeolocationEvent):void
{
    geoTextField.text = "latitude: " + event.latitude.toString() + "\n"
            + "longitude: " + event.longitude.toString() + "\n"
            + "altitude: " + event.altitude.toString()
            + "speed: " + event.speed.toString()
            + "heading: " + event.heading.toString()
            + "horizontal accuracy: " + event.horizontalAccuracy.toString()
            + "vertical accuracy: " + event.verticalAccuracy.toString()
}
```

Para usar este ejemplo, debe crear un campo de texto `geoTextField` y añadirlo a la lista de visualización antes de utilizar este código.

Puede ajustar el intervalo de tiempo que desee para los eventos de geolocalización llamando al método `setRequestedUpdateInterval()` del objeto Geolocation. Este método toma un parámetro, `interval`, que es el intervalo de actualización requerido, en milisegundos:

```
var geo:Geolocation = new Geolocation();
geo.setRequestedUpdateInterval(10000);
```

El tiempo real entre las actualizaciones de geolocalización puede ser mayor o menor que este valor. Cualquier cambio en el intervalo de actualización afecta a todos los detectores registrados. Si no llama al método `setRequestedUpdateInterval()`, la aplicación recibe actualizaciones basadas en el intervalo predeterminado del dispositivo.

El usuario puede impedir que la aplicación acceda a los datos de geolocalización. Por ejemplo, el iPhone muestra un mensaje al usuario cuando una aplicación intenta obtener datos de geolocalización. Como respuesta al mensaje, el usuario puede denegar el acceso de la aplicación a los datos de geolocalización. El objeto Geolocation distribuye un evento `status` cuando el usuario impide el acceso a los datos de geolocalización. Asimismo, el objeto Geolocation cuenta con una propiedad `muted` que se establece en `true` cuando el sensor de geolocalización no está disponible. El objeto Geolocation distribuye un evento `status` cuando la propiedad `muted` cambia. El siguiente código muestra cómo detectar cuándo los datos de geolocalización no están disponibles:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.GeolocationEvent;
    import flash.events.MouseEvent;
    import flash.events.StatusEvent;
    import flash.sensors.Geolocation;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class GeolocationTest extends Sprite
    {

        private var geo:Geolocation;
        private var log:TextField;

        public function GeolocationTest()
        {
            super();
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            setUpTextField();

            if (Geolocation.isSupported)
            {
                geo = new Geolocation();
                if (!geo.muted)
                {
                    geo.addEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
                }
                geo.addEventListener(StatusEvent.STATUS, geoStatusHandler);
            }
            else
            {
                log.text = "Geolocation not supported";
            }
        }

        public function geoUpdateHandler(event:GeolocationEvent):void
        {
            log.text = "latitude : " + event.latitude.toString() + "\n";
            log.appendText("longitude : " + event.longitude.toString() + "\n");
        }

        public function geoStatusHandler(event:StatusEvent):void
        {
            if (geo.muted)
                geo.removeEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
            else
                geo.addEventListener(GeolocationEvent.UPDATE, geoStatusHandler);
        }

        private function setUpTextField():void
```

```
        {
            log = new TextField();
            var format:TextFormat = new TextFormat("_sans", 24);
            log.defaultTextFormat = format;
            log.border = true;
            log.wordWrap = true;
            log.multiline = true;
            log.x = 10;
            log.y = 10;
            log.height = stage.stageHeight - 20;
            log.width = stage.stageWidth - 20;
            log.addEventListener(MouseEvent.CLICK, clearLog);
            addChild(log);
        }
        private function clearLog(event:MouseEvent):void
        {
            log.text = "";
        }
    }
}
```

Nota: *los iPhone de primera generación, que no incluyen unidad GPS, distribuyen eventos* `update` *de forma ocasional. En estos dispositivos, un objeto Geolocation inicialmente distribuye uno o dos eventos* `update`*. Seguidamente, distribuye eventos* `update` *cuando cambia notablemente la información.*

Comprobación de compatibilidad con geolocalización

Utilice la propiedad `Geolocation.isSupported` para probar si el entorno del motor de ejecución tiene capacidad para usar esta función:

```
if (Geolocation.isSupported)
{
    // Set up geolocation event listeners and code.
}
```

Actualmente, la geolocalización solo se admite en aplicaciones basadas en ActionScript para iPhone y en Flash Lite 4. Si `Geolocation.isSupported` es `true` en tiempo de ejecución, se admite la geolocalización.

Algunos modelos de iPhone no disponen de unidad GPS. Estos modelos utilizan otros medios (como triangulización de teléfonos móviles) para obtener los datos de geolocalización. Para estos modelos, o para iPhone con GPS desactivado, un objeto Geolocation solo puede distribuir uno o dos eventos `update` iniciales.

Capítulo 56: Internacionalización de aplicaciones

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

El paquete flash.globalization facilita la creación de software internacional que pueda adaptarse a las distintas convenciones de idiomas y regiones diferentes.

Más temas de ayuda

Paquete flash.globalization

"Localización de aplicaciones" en la página 996

Charles Bihis: Want to Localize your Flex/AIR Apps? (Desea localizar sus aplicaciones de Flex/AIR; en inglés)

Conceptos básicos de internacionalización de aplicaciones

Los términos globalización e internacionalización a veces se usan indistintamente. No obstante, la mayoría de las definiciones de estos términos dicen que la globalización se refiere a una combinación de procesos de negocio e ingeniería, mientras que la internacionalización solo se refiere a la ingeniería.

A continuación encontrará definiciones de términos importantes:

Globalización Una amplia gama de procesos de ingeniería y de negocio necesitan preparar y lanzar productos y actividades de la empresa globalmente. La globalización consiste en actividades de ingeniería, como la internacionalización, la localización y la culturización, así como actividades de la empresa como gestión de productos, planificación económica, marketing y aspectos legales. La globalización a veces se abrevia como *G11n* (que representa la letra G, 11 letras más y por último la letra n). *"La globalización es lo que hacen los negocios."*

Internacionalización Un proceso de ingeniería para generalizar un producto de modo que pueda aceptar varios idiomas, scripts y convenciones culturales (incluidas divisas, reglas de ordenación, formatos de número y fecha, etc.) sin necesidad de rediseñar ni recompilar. Este proceso se puede dividir en dos conjuntos de actividades: habilitación y localización. La internacionalización a veces se denomina *listo para el mundo* y suele abreviarse como *I18n*. *"La internacionalización es lo que hacen los ingenieros."*

Localización Un proceso de adaptación de productos o servicios a un idioma, cultura o aspecto final determinado. La localización a veces se abrevia como *L10n*. *"La localización es lo que hacen los traductores."*

Culturización Un proceso de ingeniería que desarrolla o adapta funciones concretas a las necesidades específicas de una cultura. Algunos ejemplos son las funciones de edición en japonés de Adobe InDesign o la función de compatibilidad con Hanko de Adobe Acrobat.

Otros términos de internacionalización importantes se pueden definir del modo siguiente:

Conjunto de caracteres Caracteres que se utilizan para un idioma o grupo de idiomas. Un conjunto de caracteres incluye caracteres nacionales, especiales (signos de puntuación y símbolos matemáticos), dígitos numéricos y caracteres de control.

Ordenación Ordenación del texto según un orden correcto dado por una configuración regional.

Configuración local Valor que refleja el idioma y las convenciones culturales aplicadas a una región cultural, política o geográfica (que, en muchos casos, suele indicar un país). Un identificador de configuración regional exclusivo (ID de configuración regional) representa este valor. El ID de configuración regional se utiliza para buscar un conjunto de datos de configuración regional que sean compatibles. Esta compatibilidad se aplica a unidades de medida, análisis y formato de números y fechas, etc.

Paquete de recursos Conjunto guardado de elementos específicos de una configuración regional en la que se utilizará la aplicación. Un paquete de recursos suele contener todos los elementos de texto de la interfaz de usuario de la aplicación. Dentro del paquete, estos elementos se convierten al lenguaje adecuado para la configuración regional correspondiente. También puede contener otros parámetros que alteren el diseño o el comportamiento de la interfaz de usuario en una configuración regional concreta. Un paquete de recursos puede contener otros tipos de medios o referencias a otros tipos de medios con configuración regional específica.

Información general sobre el paquete flash.globalization

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

El paquete flash.globalization reúne las funciones de compatibilidad cultural del sistema operativo subyacente. Facilita la escritura de aplicaciones que siguen las convenciones culturales de usuarios individuales.

Las clases principales del paquete incluyen:

- La clase Collator que rige la ordenación y la coincidencia de las cadenas
- La clase CurrencyFormatter que da formato a números de cadenas de cantidad de divisa y analiza las cantidades y los símbolos en las cadenas de entrada
- La clase DateTimeFormatter que da formato a valores de fecha
- La clase LocaleID para recuperar información sobre una configuración regional específica
- La clase NumberFormatter que da formato y analiza valores numéricos
- La clase StringTools que gestiona la conversión tipográfica de cadenas para distinguir entre mayúsculas y minúsculas

El paquete flash.globalization y la localización de recursos

El paquete flash.globalization no se encarga de la localización de recursos. No obstante, puede utilizar la ID de configuración regional flash.globalization como valor de clave para recuperar recursos localizados mediante otras técnicas. Por ejemplo, puede localizar recursos de aplicaciones creadas con Flex con las clases ResourceManager y ResourceBundle. Para obtener más información, consulte Localizing Flex Applications (solo disponible en inglés).

Adobe AIR 1.1 también contiene algunas funciones que ayudan a localizar aplicaciones de AIR, tal como se aborda en "Localización de aplicaciones de AIR" en la página 997.

Enfoque general de la internacionalización de una aplicación

Los siguientes pasos describen un enfoque habitual de alto nivel para internacionalizar una aplicación con el paquete flash.globalization:

1. Determine o establezca la configuración regional.
2. Cree una instancia de una clase de servicio (Collator, CurrencyFormatter, DateTimeFormatter, NumberFormatter o StringTools).
3. Busque errores y repuestos con las propiedades lastOperationStatus.
4. Dé formato y visualice información con ajustes específicos de una configuración regional.

El siguiente paso es cargar y visualizar cadenas y recursos de la interfaz de usuario específicos de la configuración regional. Este paso puede incluir tareas como:

- Usar las funciones autolayout para cambiar la UI de tamaño y poder admitir la longitud de la cadenas
- Elegir las fuentes adecuadas y las de repuesto
- Usar el motor de texto FTE para admitir otros sistemas de escritura
- Garantizar que los editores de método de entrada se gestionan correctamente

Comprobación de errores y repuestos

Las clases del servicio flash.globalization siguen todas un mismo patrón para identificar errores. También comparten un patrón para ofrecer una configuración regional sustituta compatible con el sistema operativo de cada usuario.

El siguiente ejemplo muestra cómo comprobar errores y repuestos cuando se crean instancias de clases de servicio. Cada clase de servicio tiene una propiedad lastOperationStatus que indica si la llamada ma´s reciente al método activó errores o advertencias.

```
var nf:NumberFormatter = new NumberFormatter("de-DE");
if(nf.lastOperationStatus != LastOperationStatus.NO_ERROR)
{
    if(nf.lastOperationStatus == LastOperationStatus.USING_FALLBACK_WARNING)
    {
        // perform fallback logic here, if needed
        trace("Warning - Fallback locale ID: " + nf.actualLocaleIDName);
    }
    else
    {
        // perform error handling logic here, if needed
        trace("Error: " + nf.lastOperationStatus);
    }
}
```

Este ejemplo simplemente traza un mensaje si se utiliza un ID de configuración regional de repuesto o si se produce un error. Su aplicación puede aplicar lógica de gestión de errores adicional, si es necesario. Por ejemplo, puede mostrar un mensaje al usuario o hacer que la aplicación utilice una configuración regional específica admitida.

Determinación de la configuración regional

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

Una configuración regional identifica una combinación específica de convenciones idiomáticas y culturales que se aplican a un país o región.

Los identificadores de configuración regional se pueden administrar como una cadena. Pero puede utilizar la clase LocaleID para obtener información adicional relacionada con la configuración regional.

Los objetos LocaleID se crean del modo siguiente:

```
var locale:LocaleID = new LocaleID("es-MX");
```

Una vez creado el objeto LocaleID, puede recuperar los datos de su ID de configuración regional. Utilice los métodos `getKeysAndValues()`, `getLanguage()`, `getRegion()`, `getScript()`, `getVariant()` y `isRightToLeft()`, y la propiedad `name`.

Los valores recuperados de estos métodos y propiedades pueden contener información adicional que no podría extraerse directamente del identificador de configuración regional.

Cuando una aplicación crea un servicio con detección de configuración regional, como un formateador de datos, debe especificar la configuración regional que se va a utilizar. La lista de configuraciones regionales admitidas varía según el sistema operativo; por ello, es posible que la configuración regional solicitada no siempre esté disponible.

Flash Player primero intenta buscar coincidencias del código de idioma de la configuración regional solicitada. Seguidamente, intenta afinar la búsqueda de configuración regional y busca un sistema de escritura (script) y una región. Por ejemplo:

```
var loc:LocaleID = new LocaleID("es");
trace(loc.getLanguage()); // es
trace(loc.getScript()); // Latn
trace(loc.getRegion()); // ES
```

En este ejemplo, el constructor `LocaleID()` recupera datos sobre la configuración regional que coincide con el código de idioma “es” de dicho usuario.

Definición de la ID de configuración regional

Hay varias formas de definir la configuración regional actual de una aplicación, por ejemplo:

- Incluir un ID de configuración regional único en el código de la aplicación. Es una táctica habitual, pero no permite la internacionalización de la aplicación.
- Utilizar las preferencias del ID de configuración regional del sistema operativo del usuario, o del navegador, u otro conjunto de preferencias del usuario. Normalmente esta técnica produce los mejores resultados de coincidencia de configuración regional para el usuario, aunque no es totalmente preciso. Existe el riesgo de que la configuración del sistema operativo no refleje las preferencias reales del usuario. Por ejemplo, el usuario puede estar utilizando un equipo compartido y no poder cambiar las configuraciones regionales preferidas del sistema operativo.
- Tras definir el ID de configuración regional en las preferencias del usuario, deje que el usuario seleccione la configuración regional que desee en una lista. Esta estrategia suele ser la mejor opción si la aplicación admite más de una configuración regional.

Puede implementar la siguiente tercera opción también si lo desea:

1 Recupere una lista con las configuraciones regionales o los idiomas preferidos del usuario desde su perfil, la configuración del navegador, la configuración del sistema operativo o desde una cookie. (Su aplicación debería poder implementar esta lógica pos sí misma. La biblioteca flash.globalization no admite leer directamente preferencias de este tipo.)

2 Determine qué configuración regional de las mostradas se admite en la aplicación y seleccione la mejor como predeterminada. Utilice el método LocaleID.determinePreferredLocales() para encontrar las mejores configuraciones regionales para un usuario en función de sus configuraciones regionales preferidas y de las admitidas por el sistema operativo.

3 Deje que el usuario pueda cambiar la configuración predeterminada de la configuración regional en caso de que la predeterminada no sea la elegida.

Limitaciones de otras clases de configuración regional e idioma

La clase `fl.lang.Locale` permite reemplazar cadenas de texto a partir de una configuración regional, mediante paquetes de recursos con valores de cadena. Sin embargo, esta clase no admite otras características de internacionalización, como los números, la divisa o el formato de fechas, la ordenación, etc. Además, esta clase solo está disponible en Flash Professional.

También puede recuperar la configuración del código de idioma actual para el sistema operativo con la propiedad `flash.system.Capabilities.language`. Sin embargo, esta propiedad recupera solo el código del lenguaje ISO 639-1 de dos caracteres (no el ID de configuración regional completo) y solo admite un grupo concreto de configuraciones regionales.

Con AIR 1.5, puede utilizar la propiedad `flash.system.Capabilities.languages`. Esta propiedad proporciona un conjunto de idiomas de interfaz de usuario preferidos por los usuarios. De esto modo, no tiene las limitaciones de `Capabilities.language`.

Formato de números

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

El formato de visualización de valores numéricos varía sustancialmente entre las regiones. Por ejemplo, el número 123456.78 se escribiría del modo siguiente en determinadas configuraciones regionales:

Configuración local	Formato de número
en-US (inglés, EE.UU.)	-123,456.78
de-DE (alemán, Alemania)	-123.456,78
fr-FR (francés, Francia)	-123 456,78
de-CH (alemán, Suiza)	-123'456.78
en-IN (inglés, India)	-1,23,456.78
Varias configuraciones regionales árabes	123,456.78-

Existen muchos factores que influyen en los formatos de números, por ejemplo:

- Separadores. El separador decimal se coloca entre el entero y la fracción de un número. Puede ser un punto, una coma u otro carácter. El separador de agrupación o separador de miles puede ser un punto, una coma, un espacio duro u otro carácter.
- Patrones de agrupación. El número de dígitos entre cada separador de agrupación a la izquierda de la coma decimal puede ser dos, tres u otro valor.
- Indicadores de números negativos. Los números negativos se pueden escribir con un signo menos a la izquierda o a la derecha del número, o entre paréntesis en aplicaciones financieras. Por ejemplo, un 19 negativo se puede escribir como -19, 19- o (19).
- Ceros a la derecha y a la izquierda. Algunas convenciones culturales añaden ceros a la derecha o a la izquierda para escribir números. Por ejemplo, el valor 0.17 puede escribirse como .17, 0.17 o 0.170, entre otras posibilidades.
- Conjuntos de caracteres de dígitos. Muchos idiomas, como el hindi, el árabe y el japonés, utilizan distintos conjuntos de caracteres de dígitos. El paquete flash.globalization admite todos los conjuntos de caracteres de dígitos que representan los dígitos 0-9.

La clase NumberFormatter considera todos los factores al dar formato a valores numéricos.

Uso de la clase NumberFormatter

La clase NumberFormatter aplica formato a valores numéricos (de tipo int, uint o Number) según las convenciones de una configuración regional específica.

El siguiente ejemplo muestra el modo más sencillo de dar formato a un número con las propiedades predeterminadas de formato proporcionadas por el sistema operativo del usuario:

```
var nf:NumberFormatter = new NumberFormatter(LocaleID.DEFAULT);
trace(nf.formatNumber(-123456.789))
```

El resultado variará en función de la configuración regional del usuario y de sus preferencias. Por ejemplo, si la configuración regional del usuario es fr-FR, el valor con formato sería:

-123.456,789

Si solo quiere dar formato a un número para una configuración regional específica, independientemente de la configuración del usuario, defina el nombre de la configuración regional exacta que desee. Por ejemplo:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
trace(nf.formatNumber(-123456.789));
```

Los resultados en este caso son:

-123'456.789

El método formatNumber() toma un objeto Number como parámetro. La clase NumberFormatter también tiene un método formatInt() que toma un objeto int como entrada, y un método formatUint() que toma un objeto uint.

Puede controlar de forma explícita la lógica de formato estableciendo propiedades de la clase NumberFormatter, tal como puede verse en este ejemplo:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
nf.negativeNumberFormat = 0;
nf.fractionalDigits = 5;
nf.trailingZeros = true;
nf.decimalSeparator = ",";
nf.useGrouping = false;
trace(nf.formatNumber(-123456.789)); //(123456.78900)
```

Este ejemplo primero crea un objeto NumberFormatter y después:

- establece el formato de número negativo con paréntesis (como en aplicaciones financieras);
- establece el número de dígitos detrás del separador decimal en 5;
- especifica los ceros a la derecha que deben usarse para garantizar cinco posiciones decimales;
- define el separador decimal en una coma;
- pide al aplicador de formato no utilizar separadores de agrupación.

Nota: *al cambiar algunas de estas propiedades, el formato de número resultante ya no corresponderá al formato preferido en la configuración regional especificada. Utilice algunas de estas propiedades solo si no es importante la configuración regional; si necesita control preciso sobre algún aspecto de formato, como el número de ceros a la derecha; o si el usuario solicita el cambio directamente, por ejemplo, desde el Panel de control de Windows.*

Análisis de cadenas con valores numéricos

La clase NumberFormatter también puede extraer valores numéricos de cadenas que se ajusten a los requisitos de formato específicos de la configuración regional. El método NumberFormatter.parseNumber() extra un solo valor numérico de una cadena. Por ejemplo:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "-1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // -1234567.89
trace("Status:" + nf.lastOperationStatus); // noError
```

El método parseNumber() controla cadenas que solo contienen dígitos y caracteres de formato de números, como signos negativos y separadores. Si la cadena contiene otros caracteres, se define un código de error:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "The value is 1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // NaN
trace("Status:" + nf.lastOperationStatus); // parseError
```

Para extraer números de cadenas con caracteres alfabéticos adicionales, utilice el método NumberFormatter.parse():

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String = "The value is 123,456,7.890";
var parseResult:NumberParseResult = nf.parse(inputNumberString);
trace("Value:" + parseResult.value); // 1234567.89
trace("startIndex: " + parseResult.startIndex); // 14
trace("Status:" + nf.lastOperationStatus); // noError
```

El método parse() devuelve un objeto NumberParseResult que contiene el valor numérico analizado en su propiedad value. La propiedad startIndex indica el índice del primer carácter numérico encontrado. Puede utilizar las propiedades startIndex y endIndex para extraer las partes de la cadena que van antes y después de los dígitos.

Formato de valores de divisa

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

Los formatos de visualización de valores de divisa varían tanto como los de los números. Por ejemplo, un valor en dólares estadounidenses de $123456.78 tiene distinto formato en determinadas regiones:

Configuración local	Formato de número
en-US (inglés, EE.UU.)	$123,456.78
de-DE (alemán, Alemania)	123.456,78 $
en-IN (inglés, India)	$ 1,23,456.78

El formato de divisa implica los mismos factores que el de números, además de algunos adicionales:

- Código ISO de divisa. El código de divisa ISO 4217 de tres letras para la configuración regional real que se utiliza, por ejemplo, USD o EUR.
- Símbolo de divisa. El símbolo o cadena de divisa para la configuración regional real que se utiliza, por ejemplo, $ o €.
- Formato de divisa negativa. Define la ubicación del símbolo de divisa y del símbolo negativo o el paréntesis en relación con la parte numérica de la cantidad de divisa con formato.
- Formato de divisa positiva. Define la ubicación del símbolo de divisa relativo a la parte numérica de la cantidad de divisa.

Uso de la clase CurrencyFormatter

La clase CurrencyFormatter aplica a valores numéricos formato de cadenas que contienen cadenas de divisa y números con formato, según las convenciones de una configuración regional específica.

Cuando se crea una nueva instancia del objeto CurrencyFormatter, establece su divisa en la divisa predeterminada de la configuración regional seleccionada.

El siguiente ejemplo muestra cómo un objeto CurrencyFormatter creado con una configuración regional de alemán asume que las cantidades de divisa están en euros:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "de-DE" );
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

En la mayoría de los casos no se debe depender de la divisa predeterminada en una configuración regional. Si la configuración regional del usuario no se admite, la clase CurrencyFormatter asigna una configuración regional de repuesto. La configuración regional de repuesto puede tener una divisa predeterminada distinta. Además, normalmente querrá que los formatos de divisa tengan un aspecto correcto para el usuario, incluso si las cantidades no están en la divisa local del usuario. Por ejemplo, un usuario canadiense quiere ver los precios de una empresa alemana en euros, pero con formato canadiense.

El método CurrencyFormatter.setCurrency() especifica la cadena de divisa exacta y el símbolo que debe utilizarse.

El siguiente ejemplo muestra cantidades de divisa en euros a usuarios en la parte francófona de Canadá:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "fr-CA" );
cf.setCurrency("EUR", "€");
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

El método setCurrency() también se puede utilizar para reducir la confusión y establecer símbolos de divisa sin ambigüedades. Por ejemplo:

```
cf.setCurrency("USD","US$");
```

De forma predeterminada, el método format() muestra un código de divisa ISO 4217 de tres caracteres en vez del símbolo de la divisa. Los códigos ISO 4217 son inequívocos y no requieren localización. Sin embargo, muchos usuarios prefieren utilizar los símbolos de divisa en vez de los códigos ISO.

La clase CurrencyFormatter puede ayudarle a decidir si una cadena de divisa con formato debe utilizar un símbolo de divisa, como el símbolo del dólar o del euro, o si debe utilizar una cadena de divisa ISO de tres caracteres, como USD o EUR. Por ejemplo, una cantidad en dólares canadienses puede visualizarse como $200 para un usuario en Canadá. Para un usuario en España, sin embargo, se visualizaría como CAD 200. Utilice el método formattingWithCurrencySymbolIsSafe() para determinar si el símbolo de divisa de la cantidad será ambiguo o incorrecto teniendo en cuenta la configuración regional del usuario.

El siguiente ejemplo aplica formato de configuración regional en-US a un valor en euros. En función de la configuración regional del usuario, la cadena de salida usará el código de divisa ISO o el símbolo de la divisa.

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-CA");

if (cf.formattingWithCurrencySymbolIsSafe("USD"))
{
    trace(cf.format(1234567.89, true)); // $1,234,567.89
}
else
{
    cf.setCurrency("USD", "$");
    trace(cf.format(1234567.89)); // USD1,234,567.89
}
```

Análisis de cadenas con valores de divisa

La clase CurrencyFormatter también puede extraer una cantidad de divisa y una cadena de divisa de una cadena de entrada que se ajuste a los requisitos de formato específicos de la configuración regional. EL método CurrencyFormatter.parse() guarda la cantidad analizada y la cadena de divisa en un objeto CurrencyParseResult, como puede verse aquí:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "(GBP 123,56,7.890)";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("parsed amount: " + parseResult.value); // -1234567.89
trace("currencyString: " + parseResult.currencyString ); // GBP
```

La parte de cadena de divisa de la cadena de entrada puede contener un símbolo de divisa, un código ISO de divisa y caracteres de texto adicionales. Las posiciones de la cadena de divisa, el indicador de número negativo y el valor numérico deben coincidir con los formatos especificados en las propiedades negativeCurrencyFormat y positiveCurrencyFormat. Por ejemplo:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "Total $-123,56,7.890";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // parseError
trace("parsed amount: " + parseResult.value); // NaN
trace("currencyString: " + parseResult.currencyString ); //
cf.negativeCurrencyFormat = 2;
parseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // noError
trace("parsed amount: " + parseResult.value); // -123567.89
trace("currencyString: " + parseResult.currencyString ); // Total $
```

En este ejemplo, la cadena de entrada tiene una cadena de divisa seguida de un signo menos y de un número. Sin embargo, el valor predeterminado de negativeCurrencyFormat para la configuración regional en-US especifica que el indicador negativo debe ir primero. Como resultado, el método parse() genera un error y el valor analizado es NaN.

Tras definir negativeCurrencyFormat como 2, que especifica que la cadena de divisa va primero, el método parse() se ejecuta correctamente.

Formato de fechas y horas

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

El formato de visualización de los valores de fecha y hora también varía sustancialmente de región a región. Por ejemplo, el segundo día de enero de 1962 a las 1:01 PM se visualizaría del modo siguiente en formato corto y en determinadas configuraciones regionales:

Configuración local	Formato de fecha y hora
en-US (inglés, EE.UU.)	1/2/62 1:01pm
fr-FR (francés, Francia)	2/1/62 13:01
ja-JP (japonés, Japón)	1962/2/1 13:01

Uso de la clase DateTimeFormatter

La clase DateTimeFormatter aplica formato a valores Date en cadenas de fecha y hora según las convenciones de una configuración regional específica.

El formato está regido por una cadena de patrones que contiene secuencias de letras que se reemplazan con valores de fecha y hora. Por ejemplo, en el patrón "yyyy/MM", los caracteres "yyyy" se reemplazan por un año de cuatro dígitos seguido de un carácter "/" y un mes de dos dígitos.

La cadena de patrones se puede definir de forma explícita con el método setDateTimePattern(). Sin embargo, es mejor dejar que el patrón se defina automáticamente en función de la configuración regional del usuario y de las preferencias del sistema operativo. Esta práctica ayuda a garantizar que el resultado es correcto desde un punto de vista cultural.

DateTimeFormatter puede representar fechas y horas en tres estilos estándar (LONG, MEDIUM y SHORT), y también puede utilizar un patrón CUSTOM. Se puede utilizar un estilo para la fecha y un segundo estilo para la hora. Los patrones reales utilizados por cada estilo pueden variar en función del sistema operativo, aunque no demasiado.

Puede especificar los estilos al crear un objeto DateTimeFormatter. Si no se especifican parámetros de estilo, se establecen de forma predeterminada en DateTimeStyle.LONG. Puede cambiar los estilos más adelante con el método setDateTimeStyles(), como puede verse en el siguiente ejemplo:

```
var date:Date = new Date(2009, 2, 27, 13, 1);
var dtf:DateTimeFormatter = new DateTimeFormatter("en-US",
    DateTimeStyle.LONG, DateTimeStyle.LONG);

var longDate:String = dtf.format(date);
trace(longDate); // March 27, 2009 1:01:00 PM

dtf.setDateTimeStyles(DateTimeStyle.SHORT, DateTimeStyle.SHORT);
var shortDate:String = dtf.format(date);
trace(shortDate); // 3/27/09 1:01 PM
```

Localización de nombres de meses y de días

Muchas aplicaciones utilizan listas de nombres de meses y de días de la semana y vistas de calendario o en listas desplegables.

Puede recuperar una lista localizada de los nombres de los meses con el método DateTimeFormatter.getMonthNames(). En función del sistema operativo, dispondrá de versiones completas y abreviadas. Pase el valor DateTimeNameStyle.FULL para obtener nombres de meses largos. Pase los valores DateTimeNameStyle.LONG_ABBREVIATION o DateTimeNameStyle.SHORT_ABBREVIATION para obtener una versión más corta.

En algunos idiomas, el nombre del mes cambia (en su forma genitiva) cuando no se sitúa junto al valor del día en un formato de fecha. Si planea utilizar nombres de meses aislados, transfiera el valor DateTimeNameContext.STANDALONE al método getMonthNames(). Para utilizar nombres de mes en fechas con formato, sin embargo, transfiera el valor DateTimeNameContext.FORMAT.

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var months:Vector.<String> = dtf.getMonthNames(DateTimeNameStyle.FULL,
        DateTimeNameContext.STANDALONE);
trace(months[0]); // janvier
months = dtf.getMonthNames(DateTimeNameStyle.SHORT_ABBREVIATION,
         DateTimeNameContext.STANDALONE);
trace(months[0]); // janv.
```

El método DateTimeFormatter.getWeekdayNames() proporciona una lista localizada de los nombres de los días de la semana. El método getWeekdayNames() acepta los mismos parámetros nameStyle y context que el método getMonthNames().

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var weekdays:Vector.<String> = dtf.getWeekdayNames(DateTimeNameStyle.FULL,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dimanche
weekdays = dtf.getWeekdayNames(DateTimeNameStyle.LONG_ABBREVIATION,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dim.
```

Además, el método getFirstWeekday() devuelve el valor de índice del día que suele marcar el inicio de la semana en la configuración regional seleccionada.

Ordenación y comparación de cadenas

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

Ordenación es el proceso de organizar cosas para que estén en un orden adecuado. Las reglas de ordenación varían sustancialmente según la configuración regional. Las reglas también son distintas si ordena una lista o compara elementos similares, como en un algoritmo de búsqueda de texto.

Durante la ordenación, las pequeñas diferencias como letras en mayúsculas y en minúsculas o los signos diacríticos como los acentos, suelen tener mucha importancia. Por ejemplo, la letra ö (o con diéresis) se considera equivalente a la letra o en francés o en inglés. La misma letra, sin embargo, corresponde a la letra z en sueco. Asimismo, en francés y en otros idiomas, un acento en una palabra determina su orden en una lista.

Al hacer búsquedas, normalmente se omiten las diferencias entre mayúscula y minúscula o los signos diacríticos para aumentar la posibilidad de encontrar coincidencias relevantes. Por ejemplo, una búsqueda de los caracteres “cote” en un documento en francés probablemente devolverá coincidencias para “cote”, “côte” y “coté”.

Uso de la clase Collator

Los métodos principales de la clase Collator son el método compare(), empleado principalmente para ordenar, y el método equals(), utilizado para coincidencia de valores.

El siguiente ejemplo muestra el distinto comportamiento de los métodos compare() y equals().

```
var words:Array = new  Array("coté", "côte");

var sorter:Collator = new Collator("fr-FR", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // côte,coté

var matcher:Collator = new Collator("fr-FR", CollatorMode.MATCHING);
if (matcher.equals(words[0], words[1]))
{
    trace(words[0] + " = " + words[1]); // côte = coté
}
```

El ejemplo primer crea un objeto Collator en modo SORTING para la configuración regional de francés de Francia. Seguidamente, ordena dos palabras que se distinguen únicamente en signos diacríticos. Esto muestra que la comparación SORTING distingue entre caracteres con y sin acentos.

La ordenación se realiza pasando una referencia al método sort() del objeto Collator como un parámetro al método Array.sort(). Esta técnica es una de las formas más eficaces de utilizar un objeto Collator para controlar la ordenación.

Seguidamente, el ejemplo crea un objeto en modo MATCHING. Cuando dicho objeto Collator compara las palabras, las trata como iguales. Esto muestra que los valores de comparación MATCHING son los mismos en caracteres con y sin acento.

Personalización del comportamiento de la clase Collator

De forma predeterminada, la clase Collator utiliza reglas de comparación de cadenas del sistema operativo a partir de la configuración regional y las preferencias del sistema del usuario. Es posible personalizar el comportamiento de los métodos compare() y equals() estableciendo de forma explícita diversas propiedades. La siguiente tabla contiene las propiedades y el efecto que tienen en las comparaciones:

Propiedad Collator	**Efecto**
numericComparison	Controla si los caracteres de dígitos se tratan como números o como texto.
ignoreCase	Controla si se tienen en cuenta las diferencias entre mayúsculas y minúsculas.
ignoreCharacterWidth	Controla si se consideran iguales formas de ancho completo y de ancho medio en determinados caracteres chinos y japoneses.
ignoreDiacritics	Controla si las cadenas que utilizan los mismos caracteres básicos pero distintos acentos o signos diacríticos se consideran iguales.
ignoreKanaType	Controla si las cadenas que se diferencian solo por el tipo de carácter kana utilizado se consideran iguales.
ignoreSymbols	Controla si los caracteres de símbolos, como los espacios, símbolos de divisa, símbolos matemáticos, etc., se omiten.

El siguiente código muestra que establecer la propiedad ignoreDiacritics en true, cambia el orden de una lista de palabras en francés:

```
var words:Array = new  Array("COTE", "coté", "côte", "Coté","cote");
var sorter:Collator = new Collator("fr-CA", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // cote,COTE,côte,coté,Coté

sorter.ignoreDiacritics = true;
words.sort(sorter.compare);
trace(words); // côte,coté,cote,Coté,COTE
```

Conversión tipográfica

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

Los idiomas también tienen reglas distintas para convertir letras de caja alta (mayúsculas) y de caja baja (minúsculas).

Por ejemplo, la mayoría de los idiomas que utilizan el alfabeto latino, la caja baja de la “I” mayúscula es “i”. Sin embargo, algunos idiomas (como el turco o el azeri) existe otra “i” sin punto: la letra “ı”. Como resultado, en esos idiomas una “ı” minúscula sin punto se transforma en una “I” mayúscula. Una “i” minúscula se transforma en una “İ” mayúscula con un punto.

La clase StringTools proporciona métodos que utilizan reglas específicas de un idioma para llevar a cabo dichas transformaciones.

Uso de la clase StringTools

La clase StringTools proporciona dos métodos para llevar a cabo transformaciones de caja: toLowerCase() y toUpperCase(). Puede crear un objeto StringTools llamando al constructor con un ID de configuración regional. La clase StringTools recuperará las reglas de conversión de caja para dicha configuración regional (o para la de repuesto) en el sistema operativo. No es posible personalizar más el algoritmo de conversión de caja.

El siguiente ejemplo utiliza los métodos toUpperCase() y toLowerCase() para transformar una frase en alemán que contiene la letra “ß”.

```
var phrase:String = "Schloß Neuschwanstein";
var converter:StringTools = new StringTools("de-DE");

var upperPhrase:String = converter.toUpperCase(phrase);
trace(upperPhrase); // SCHLOSS NEUSCHWANSTEIN

var lowerPhrase:String = converter.toLowerCase(upperPhrase);
trace(lowerPhrase);// schloss neuschwanstein
```

El método toUpperCase() transforma la letra “ß” minúscula en las letras “SS” mayúsculas. Esta transformación no es bidireccional. Cuando las letras “SS” se convierten otra vez a minúscula, el resultado es “ss”, no “ß”.

Ejemplo: Internacionalización de una aplicación de bolsa

Flash Player 10.1 y posterior, Adobe AIR 2.0 y posterior

La aplicación Global Stock Ticker recupera y muestra datos ficticios sobre las acciones en tres mercados distintos: Estados Unidos, Japón y Europa. Aplica formato a los datos según las convenciones de distintas configuraciones regionales.

Este ejemplo ilustra las siguientes funciones del paquete flash.globalization:

- Formato de números según configuración regional
- Formato de divisas según configuración regional
- Definición del código ISO de divisa y de símbolos de divisa
- Formato de fechas según configuración regional
- Recuperación y visualización de abreviaturas correctas de nombres de meses

Para obtener los archivos de la aplicación de este ejemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_es. Los archivos de la aplicación Global Stock Ticker se encuentran en la carpeta Samples/GlobalStockTicker. La aplicación consta de los siguientes archivos:

Archivo	Descripción
GlobalStockTicker.mxml o GlobalStockTicker.fla	La interfaz de usuario de la aplicación para Flex (MXML) o Flash (FLA).
styles.css	Estilos de la interfaz de usuario de la aplicación (solo Flex).
com/example/programmingas3/stockticker/flex/FinGraph.mxml	Un componente MXML que muestra un gráfico con la simulación de los datos de las acciones (solo Flex).
com/example/programmingas3/stockticker/flash/GlobalStockTicker.as	Clase de documento que contiene la lógica de la interfaz de usuario para la aplicación (solo Flash).
comp/example/programmingas3/stockticker/flash/RightAlignedColumn.as	Procesador de celdas personalizado para el componente DataGrid de Flash (solo Flash).
com/example/programmingas3/stockticker/FinancialGraph.as	Clase de ActionScript que dibuja un gráfico con la simulación de los datos de las acciones.
com/example/programmingas3/stockticker/Localizer.as	Clase de ActionScript que administra la configuración regional y la divisa, y gestiona el formato localizado de números, cantidades de divisa y fechas.
com/example/programmingas3/stockticker/StockDataModel.as	Clase de ActionScript que guarda los datos de muestra del ejemplo de la aplicación Global Stock Ticker.

Aspectos básicos de la interfaz de usuario y los datos de muestra

Los elementos principales de la interfaz de usuario de la aplicación son:

- un cuadro combinado para seleccionar una configuración regional
- un cuadro combinado para seleccionar un mercado
- un objeto DataGrid que muestra datos de seis empresas en cada mercado
- un gráfico que muestre la simulación de datos históricos de las acciones de la empresa seleccionada

La aplicación guarda todos estos datos de muestra sobre las configuraciones regionales, los mercados y las acciones de las empresas en la clase StockDataModel. Una aplicación real como esta recuperaría datos de un servidor y los guardaría en una clase como StockDataModel. En este ejemplo, todos los datos se incluyen en la clase StockDataModel en el código.

Nota: *los datos visualizados en el gráfico financiero no coinciden necesariamente con los del control DataGrid. El gráfico se dibuja de nuevo de forma aleatoria cada vez que se selecciona una empresa. Se ha diseñado así únicamente con fines ilustrativos.*

Definición de la de configuración regional

Tras cierto trabajo inicial de configuración, la aplicación llama al método Localizer.setLocale() para crear objetos de formato para la configuración regional predeterminada. También se llama al método setLocale() cada vez que el usuario selecciona un nuevo valor en el cuadro combinado de configuración regional.

```
public function setLocale(newLocale:String):void
{
    locale  = new LocaleID(newLocale);

    nf = new NumberFormatter(locale.name);
    traceError(nf.lastOperationStatus, "NumberFormatter", nf.actualLocaleIDName);

    cf = new CurrencyFormatter(locale.name);
    traceError(cf.lastOperationStatus, "CurrencyFormatter", cf.actualLocaleIDName);
    symbolIsSafe = cf.formattingWithCurrencySymbolIsSafe(currentCurrency);
    cf.setCurrency(currentCurrency, currentSymbol);
    cf.fractionalDigits = currentFraction;

    df = new DateTimeFormatter(locale.name, DateTimeStyle.LONG, DateTimeStyle.SHORT);
    traceError(df.lastOperationStatus, "DateTimeFormatter", df.actualLocaleIDName);
    monthNames = df.getMonthNames(DateTimeNameStyle.LONG_ABBREVIATION);
}

public function traceError(status:String, serviceName:String, localeID:String) :void
{
    if(status != LastOperationStatus.NO_ERROR)
    {
        if(status == LastOperationStatus.USING_FALLBACK_WARNING)
```

```
        {
            trace("Warning - Fallback locale ID used by "
                    + serviceName + ": " + localeID);
        }
        else if (status == LastOperationStatus.UNSUPPORTED_ERROR)
        {
            trace("Error in " + serviceName + ": " + status);
            //abort application
            throw(new Error("Fatal error", 0));
        }
        else
        {
            trace("Error in " + serviceName + ": " + status);
        }
    }
    else
    {
        trace(serviceName + " created for locale ID: " + localeID);
    }
}
```

Primero, el método setLocale() crea un objeto LocaleID. Este objeto facilita la obtención de detalles sobre la configuración regional real más adelante, si fuera necesario.

A continuación, crea nuevos objetos NumberFormatter, CurrencyFormatter y DateTimeFormatter para la configuración regional. Tras crear cada objeto formatter, llama al método traceError(). Este método muestra mensajes de error y de advertencia en la consola si se detecta un problema con la configuración regional solicitada. (Una aplicación real iniciaría alguna acción a partir de estos errores, no se limitaría a detectarlos).

Una vez creado el objeto CurrencyFormatter, el método setLocale() define el código ISO de divisa de formato, el símbolo de divisa y las propiedades fractionalDigits en valores determinados previamente. (Estos valores se establecen cada vez que el usuario selecciona un mercado en el cuadro combinado de mercados).

Tras crear el objeto DateTimeFormatter, el método setLocale() también recupera un conjunto de abreviaturas de nombres de meses localizados.

Aplicación de formato a los datos

Los datos de las acciones con formato se presentan en un control DataGrid. Cada columna de DataGrid llama a una función label que aplica formato al valor de la columna con el objeto de formato apropiado.

En la versión para Flash, por ejemplo, el siguiente código establece las columnas DataGrid:

```
var col1:DataGridColumn = new DataGridColumn("ticker");
col1.headerText = "Company";
col1.sortOptions = Array.NUMERIC;
col1.width = 200;

var col2:DataGridColumn = new DataGridColumn("volume");
col2.headerText = "Volume";
col2.width = 120;
col2.cellRenderer = RightAlignedCell;
col2.labelFunction = displayVolume;

var col3:DataGridColumn = new DataGridColumn("price");
col3.headerText = "Price";
col3.width = 70;
col3.cellRenderer = RightAlignedCell;
col3.labelFunction = displayPrice;

var col4:DataGridColumn = new DataGridColumn("change");
col4.headerText = "Change";
col4.width = 120;
col4.cellRenderer = RightAlignedCell;
col4.labelFunction = displayPercent;
```

La versión para Flex del ejemplo declara su DataGrid en MXML. También define funciones label similares para cada columna.

Las propiedades labelFunction hacen referencia a las siguientes funciones, que llaman a métodos de formato de la clase Localizer:

```
private function displayVolume(item:Object):String
{
    return localizer.formatNumber(item.volume, 0);
}

private function displayPercent(item:Object):String
{
    return localizer.formatPercent(item.change )  ;
}

  private function displayPrice(item:Object):String
{
    return localizer.formatCurrency(item.price);
}
```

Seguidamente, los métodos Localizer establecen y llaman a los objetos de formato apropiados:

```
public function formatNumber(value:Number, fractionalDigits:int = 2):String
{
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value);
}

public function formatPercent(value:Number, fractionalDigits:int = 2):String
{
    // HACK WARNING: The position of the percent sign, and whether a space belongs
    // between it and the number, are locale-sensitive decisions. For example,
    // in Turkish the positive format is %12 and the negative format is -%12.
    // Like most operating systems, flash.globalization classes do not currently
    // provide an API for percentage formatting.
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value) + "%";
}

public function formatCurrency(value:Number):String
{
    return cf.format(value, symbolIsSafe);
}

public function formatDate(dateValue:Date):String
{
    return df.format(dateValue);
}
|
```

Capítulo 57: Localización de aplicaciones

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La localización es el proceso de incluir componentes que admitan el uso de varias configuraciones regionales. Una configuración regional es la combinación de un idioma y un código de país. Por ejemplo: es_ES se refiere al idioma español tal y como se habla en España, y fr_FR se refiere al idioma francés que se habla en Francia. Para localizar una aplicación para estas configuraciones regionales se necesitarían dos series de componentes: una para la configuración regional es_ES y otra para la configuración regional fr_FR.

Un mismo idioma puede utilizarse en distintas configuraciones regionales. Por ejemplo, es_ES y es_UY (Uruguay) son distintas configuraciones regionales. En este caso, ambas configuraciones utilizan el idioma español, pero el código de país indica que son distintos lugares, por lo que quizá utilicen componentes diferentes. Por ejemplo: una aplicación en la configuración regional es_ES podría usar la ortografía "vídeo", mientras que en la configuración regional es_UY la palabra sería "video" (sin acento). Además, las unidades monetarias serían euros o pesos, dependiendo de la configuración regional, y el formato para fechas y horas también podría diferir.

También se puede proporcionar una serie de componentes para un idioma sin especificar un código de país. Por ejemplo: se pueden proporcionar componentes “es” para el idioma español y añadir recursos adicionales para la configuración regional es_ES que son específicos del español de España.

La localización va más allá de solamente traducir las cadenas de caracteres que figuran en la aplicación. También puede incluir cualquier tipo de componente como archivos de audio, imágenes y vídeos.

Elección de una configuración regional

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para determinar qué configuración regional utiliza la aplicación se puede utilizar uno de los métodos siguientes:

- Paquete flash.globalization: utilice las clases según la configuración regional en el paquete flash.globalization para recuperar la configuración regional predeterminada para el usuario a partir del sistema operativo y sus preferencias. Este es el enfoque preferido para aplicaciones que se ejecutarán en Flash Player 10.1 o posterior o en AIR 2.0 o motores de ejecución posteriores. Consulte “Determinación de la configuración regional” en la página 981 para obtener más información.
- Solicitud al usuario: se puede iniciar la aplicación con una configuración regional predeterminada y después pedir al usuario que seleccione la que prefiera.
- `(solo AIR) Capabilities.languages`: la propiedad `Capabilities.languages` presenta un conjunto de idiomas disponibles en los idiomas preferidos del usuario, según se configuran en el sistema operativo. Las cadenas contienen etiquetas de idioma (así como información sobre la región y los scripts, si procede), tal como se define en RFC4646 (http://www.ietf.org/rfc/rfc4646.txt). En las cadenas se utiliza como delimitador un guion (por ejemplo: `"es-ES"` o `"ja-JP"`). La primera entrada del conjunto devuelto tiene el mismo ID de idioma principal que la propiedad de idioma ("language"). Por ejemplo: si `languages[0]` está definido en `"es-ES"`, la propiedad `language` se define en`"es"`. Sin embargo, si se define la propiedad de idioma en `"xu"` (especificando un idioma desconocido), el primer elemento del conjunto `languages` es distinto.

- `Capabilities.language`: la propiedad`Capabilities.language` proporciona el código del idioma de la interfaz de usuario tal y como figura en el sistema operativo. No obstante, esta propiedad está limitada a 20 idiomas conocidos. En los sistemas anglosajones, esta propiedad solo devuelve el código de idioma, no el código del país. Por estos motivos resulta más conveniente utilizar el primer elemento del conjunto `Capabilities.languages`.

Localización de contenido de Flex

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Adobe Flex incluye una arquitectura para localizar el contenido de Flex. Dicha arquitectura incluye las clases Locale, ResourceBundle y ResourceManagerImpl, además de las interfaces IResourceBundle e IResourceManagerImpl.

Dispone de una biblioteca de localización de Flex que contiene clases de utilidades para ordenar configuraciones regionales de aplicaciones en Google Code (http://code.google.com/p/as3localelib/).

Más temas de ayuda

http://code.google.com/p/as3localelib/

Localización de contenido de Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Adobe Flash Professional incluye una clase Locale en los componentes ActionScript 3.0. La clase Locale permite controlar cómo muestra un archivo SWF un texto multilingüe. El panel Cadenas de Flash permite utilizar ID de cadenas en lugar de literales de cadenas en campos de texto dinámicos. Esto permite crear un archivo SWF que muestre el texto cargado desde un archivo XML de un idioma específico. Para obtener información sobre el uso de la clase Locale, consulte Referencia de ActionScript 3.0 para la plataforma de Adobe Flash.

Localización de aplicaciones de AIR

Adobe AIR 1.0 y posterior

El SDK de AIR proporciona una arquitectura de localización de HTML (que se encuentra en el archivo AIRLocalizer.js file). Esta arquitectura incluye API que ayudan a trabajar con diversas configuraciones regionales en una aplicación basada en HTML. Puede encontrar una biblioteca de ActionScript para ordenar configuraciones regionales en http://code.google.com/p/as3localelib/.

Más temas de ayuda

http://code.google.com/p/as3localelib/

Localización de fechas, horas y monedas

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La forma en que se expresan la fecha, la hora y la moneda en las aplicaciones varía mucho para cada configuración regional. Por ejemplo: la norma estadounidense de representar las fechas es mes/día/año, mientras que la europea es día/mes/año.

Se puede escribir código para dar formato a fechas, horas y monedas. En el siguiente ejemplo, el código convierte un objeto Date (fecha) en formato mes/día/año o día/mes/año. Si la variable `locale` (que representa la configuración regional) se define en `"es_ES"`, la función devuelve el formato mes/día/año. En el ejemplo, el código convierte un objeto Date en formato día/mes/año para todas las demás configuraciones regionales:

```
function convertDate(date)
{
    if (locale == "en_US")
    {
        return (date.getMonth() + 1) + "/" + date.getDate() + "/" + date.getFullYear();
    }
    else
    {
        return date.getDate() + "/" + (date.getMonth() + 1) + "/" + date.getFullYear();
    }
}
```

ADOBE FLEX

La arquitectura de Flex incluye controles para dar formato a fechas, horas y monedas, entre ellos los controles DateFormatter y CurrencyFormatter.

- mx:DateFormatter
- mx:CurrencyFormatter

Capítulo 58: Entorno HTML

Adobe AIR 1.0 y posterior

Adobe®AIR™ utiliza WebKit *(www.webkit.org)*, también utilizado por el navegador web Safari, para analizar, maquetar y representar contenido en formato HTML y JavaScript. El uso de las API de AIR en el contenido HTML es optativo. Se puede realizar toda la programación del contenido de un objeto HTMLLoader o una ventana HTML empleando solamente HTML y JavaScript. La mayoría de las páginas y aplicaciones de HTML existentes deberían funcionar con poca modificación (suponiendo que utilizan funciones de HTML, CSS, DOM y JavaScript que sean compatibles con WebKit).

Importante: las nuevas versiones del motor de ejecución Adobe AIR puede incluir versiones actualizadas de WebKit. Una actualización de WebKit en una nueva versión de AIR *puede* implicar cambios inesperados en una aplicación implementada de AIR. Estos cambios pueden afectar al comportamiento o la apariencia de contenido HTML en una aplicación. Por ejemplo, las mejoras o correcciones en el procesamiento de WebKit pueden modificar el diseño de elementos en una interfaz de usuario de la aplicación. Por este motivo, se recomienda que proporcione un mecanismo de actualización en la aplicación. Si se necesita actualizar la aplicación debido a un cambio en la versión de WebKit incluido en AIR, el mecanismo de actualización de AIR puede solicitar al usuario que instale la nueva versión de la aplicación.

La siguiente tabla muestra la versión del navegador web Safari que utiliza la versión equivalente del WebKit de AIR:

Versión de AIR	Versión de Safari
1.0	2.04
1.1	3.04
1.5	4.0 Beta
2.0	4.03
2.5	4.03
2.6	4.03
2.7	4.03
3	5.0.3

Siempre es posible determinar qué versión de WebKit hay instalada: basta con examinar la cadena de agente del usuario predeterminado devuelta por un objeto HTMLLoader:

```
var htmlLoader:HTMLLoader = new HTMLLoader();
trace( htmlLoader.userAgent );
```

Recuerde que la versión de WebKit empleada en AIR no es la misma que la versión de código abierto. Algunas funciones no se admiten en AIR y la versión de AIR puede incluir soluciones a errores y problemas de seguridad aún no disponibles en la versión correspondiente de WebKit. Consulte "Funciones de WebKit no admitidas en AIR" en la página 1015.

Al ejecutarse las aplicaciones de AIR directamente en el escritorio con pleno acceso al sistema de archivos, el modelo de seguridad para el contenido HTML es más estricto que el modelo de seguridad de los navegadores web habituales. En AIR solo se pone en el *entorno limitado de la aplicación* contenido cargado desde el directorio de instalación de la aplicación. El entorno limitado de la aplicación tiene el máximo nivel de privilegio y permite tener acceso a las API de AIR. AIR coloca otros tipos de contenido en entornos limitados aislados en función de la procedencia del contenido. Los archivos cargados desde el sistema de archivos pasan a un entorno limitado local. El entorno limitado al que pasan los archivos cargados desde la red mediante uso de los protocolos http: o https: depende del dominio del servidor remoto. El contenido de estos entornos limitados ajenos a la aplicación no tiene acceso a ninguna API de AIR y en esencia funciona tal y como lo haría en un navegador web habitual.

El contenido HTML en AIR no muestra contenido SWF o PDF si se aplican opciones de alfa, escala o transparencia. Para obtener más información, consulte "Consideraciones al cargar el contenido SWF o PDF en una página HTML" en la página 1045 y "Transparencia de la ventana" en la página 929.

Más temas de ayuda

Referencia del DOM de Webkit

Referencia de HTML de Safari

Referencia de CSS de Safari

www.webkit.org

Información general sobre el entorno HTML

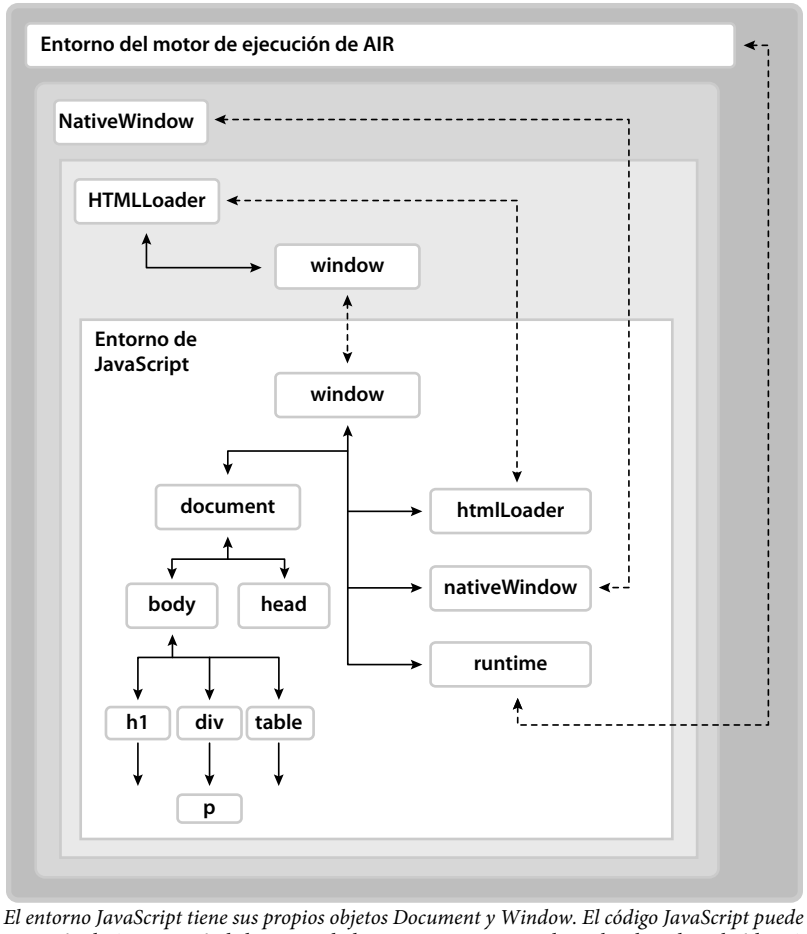
Adobe AIR 1.0 y posterior

Adobe AIR proporciona un entorno JavaScript completo similar a un navegador con procesador de HTML, modelo de objetos de documento e intérprete de JavaScript. La clase HTMLLoader de AIR representa el entorno JavaScript. En las ventanas HTML, un objeto HTMLLoader contiene todo el contenido HTML y está, a su vez, contenido en un objeto NativeWindow. En el contenido SWF se puede añadir la clase HTMLLoader, que amplía la clase Sprite, a la lista de visualización de un escenario como cualquier otro objeto de visualización. Las propiedades de Adobe® ActionScript® 3.0 de la clase se describen en "Uso de scripts en el contenedor HTML de AIR" en la página 1043 y también en *Referencia de ActionScript 3.0 para la plataforma de Adobe Flash*. En la arquitectura de Flex la clase HTMLLoader de AIR está agrupada en un componente mx:HTML. El componente mx:HTML amplía la clase UIComponent de modo que se pueda utilizar directamente con otros contenedores de Flex. Por lo demás, el entorno JavaScript dentro del componente mx:HTML es idéntico.

Entorno JavaScript y su relación con el host de AIR

Adobe AIR 1.0 y posterior

El siguiente esquema ilustra la relación entre el entorno JavaScript y el entorno del motor de ejecución de AIR. Si bien se muestra una sola ventana nativa, una aplicación de AIR puede contener varias ventanas. (Y una sola ventana puede contener varios objetos HTMLLoader).



El entorno JavaScript tiene sus propios objetos Document y Window. El código JavaScript puede interactuar con el entorno del motor de ejecución de AIR a través de las propiedades runtime, nativeWindow y htmlLoader. El código ActionScript puede interactuar con el entorno JavaScript a través de la propiedad de ventana ("window") de un objeto HTMLLoader, que es una referencia al objeto Window de JavaScript. Además, tanto los objetos ActionScript como los objetos JavaScript pueden detectar eventos distribuidos por objetos de AIR y JavaScript.

La propiedad `runtime` facilita el acceso a las clases API de AIR, lo cual permite crear nuevos objetos de AIR además de acceder a miembros estáticos. Para tener acceso a una API de AIR se añade el nombre de la clase, con el paquete, a la propiedad `runtime`. Por ejemplo, para crear un objeto File se utilizaría la sentencia:

```
var file = new window.runtime.filesystem.File();
```

Nota: *el SDK de AIR proporciona un archivo JavaScript, `AIRAliases.js`, que define los alias más convenientes para las clases de AIR de uso más frecuente. Al importar este archivo se puede utilizar la forma abreviada air.Class en lugar de window.runtime.package.Class. Por ejemplo, se podría crear el objeto File con `new air.File()`.*

El objeto NativeWindow proporciona propiedades para controlar la ventana del escritorio. Desde una página HTML se puede tener acceso al objeto NativeWindow contenedor con la propiedad `window.nativeWindow`.

El objeto HTMLLoader proporciona propiedades, métodos y eventos para controlar cómo se carga y representa el contenido. Desde una página HTML se puede tener acceso al objeto HTMLLoader principal con la propiedad `window.htmlLoader`.

Importante: *solo las páginas instaladas como parte de una aplicación tienen las propiedades* `htmlLoader`*,* `nativeWindow` *o* `runtime` *y solo cuando se cargan como documento del nivel superior. Estas propiedades no se añaden cuando se carga un documento en un fotograma o en un iframe. (Un documento secundario tiene acceso a estas propiedades en el documento principal siempre que se encuentre en el mismo entorno limitado de seguridad. Por ejemplo, un documento cargado en un fotograma tiene acceso a la propiedad* `runtime` *del documento principal con* `parent.runtime`*).*

Información sobre seguridad

Adobe AIR 1.0 y posterior

AIR ejecuta todos los códigos en un entorno limitado de seguridad basado en el dominio de origen. El contenido de la aplicación, que se limita al contenido cargado desde el directorio de instalación de la aplicación, se pone en el entorno limitado *de la aplicación*. El acceso al entorno del motor de ejecución y las API de AIR solo está disponible para HTML y JavaScript cuando se ejecutan en este entorno limitado. Al mismo tiempo, la mayor parte de la ejecución y evaluación dinámica de JavaScript queda bloqueada en el entorno limitado de la aplicación tras haberse devuelto todos los controladores del evento `load` de la página.

Se puede asignar una página de la aplicación a un entorno limitado ajeno a la aplicación cargando la página en un fotograma o en un iframe y configurando los atributos del fotograma `sandboxRoot` y `documentRoot` que son específicos de AIR. Si se define el valor `sandboxRoot` en un dominio remoto real, se puede habilitar el contenido del entorno limitado para usar scripts entre contenidos de ese dominio. La asignación de páginas de esta forma resulta útil al cargar contenido remoto y usar scripts con el mismo, como en una aplicación *mashup*.

Otra forma de posibilitar el uso de scripts entre contenidos de la aplicación y ajenos -y la única manera de brindar acceso a las API de AIR al contenido ajeno- es crear un *puente de entorno limitado*. Un puente *de principal a secundario* permite al contenido de un fotograma, iframe o ventana de nivel secundario tener acceso a métodos y propiedades designados que están definidos en el entorno limitado de la aplicación. El caso contrario, de un puente *de secundario a principal*, permite al contenido de la aplicación tener acceso a métodos y propiedades designados que están definidos en el entorno limitado de nivel secundario. Los puentes de entorno limitado se establecen definiendo las propiedades `parentSandboxBridge` y `childSandboxBridge` del objeto window. Para obtener más información, consulte "Seguridad HTML en Adobe AIR" en la página 1127 y "Elementos frame e iframe de HTML" en la página 1011.

Plug-ins y objetos incorporados

Adobe AIR 1.0 y posterior

AIR es compatible con el plug-in Adobe® Acrobat®. El usuario debe contar con Acrobat o Adobe® Reader® 8.1 (o superior) para poder visualizar contenido PDF. El objeto HTMLLoader proporciona una propiedad para comprobar si el sistema del usuario puede visualizar PDF. También se puede visualizar el contenido de archivos SWF en el entorno HTML, pero esta capacidad está incorporada en AIR y no utiliza un plug-in externo.

Ningún otro plug-in de Webkit es compatible con AIR.

Más temas de ayuda

"Seguridad HTML en Adobe AIR" en la página 1127

"Entornos limitados en HTML" en la página 1003

"Elementos frame e iframe de HTML" en la página 1011

"Objeto Window en JavaScript" en la página 1009

"Objeto XMLHttpRequest" en la página 1005

"Cómo añadir contenido PDF en AIR" en la página 575

AIR y WebKit

Adobe AIR 1.0 y posterior

Adobe AIR utiliza el motor WebKit de código abierto que también se utiliza en el navegador web Safari. AIR añade varias extensiones que facilitan el acceso a las clases y los objetos del motor de ejecución, además de aumentar la seguridad. Por otra parte, el propio Webkit añade funciones que no están incluidas en las normas W3C para HTML, CSS y JavaScript.

Aquí solo se tratan los complementos de AIR y las extensiones de WebKit más notables. Para ver documentación suplementaria sobre HTML, CSS y JavaScript no estándar, visite www.webkit.org y developer.apple.com (en inglés). Para obtener información sobre las normas, visite el sitio web de W3C en. Mozilla también proporciona una valiosa referencia general sobre temas relacionados con HTML, CSS y DOM (naturalmente, los motores de Webkit y Mozilla no son idénticos).

JavaScript en AIR

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

AIR modifica de varias maneras el comportamiento típico de objetos comunes de JavaScript. Muchas de estas modificaciones tienen la finalidad de facilitar la escritura de aplicaciones seguras en AIR. Al mismo tiempo, debido a estas diferencias de comportamiento es posible que algunos patrones de codificación de JavaScript, así como las aplicaciones web existentes que utilicen esos patrones, no siempre se ejecutan de la forma prevista en AIR. Para obtener información sobre la corrección de este tipo de problema, consulte "Cómo evitar errores de JavaScript relacionados con la seguridad" en la página 1021.

Entornos limitados en HTML

Adobe AIR 1.0 y posterior

AIR pone contenido en entornos limitados aislados en función del origen del contenido. Las reglas de entornos limitados son coherentes con la política de "mismo origen" que implementa la mayoría de los navegadores web, además de las reglas para entornos limitados implementados por Adobe Flash Player. Por otra parte, AIR proporciona un nuevo tipo de entorno limitado de la *aplicación* que contiene y protege el contenido de la aplicación. Para obtener más información sobre los tipos de entornos limitados que puede encontrar al desarrollar aplicaciones de AIR, consulte "Entornos limitados de seguridad" en la página 1087.

El acceso al entorno del motor de ejecución y las API de AIR solo está disponible para HTML y JavaScript cuando se ejecutan en el entorno limitado de la aplicación Al mismo tiempo, la ejecución y evaluación dinámica de JavaScript, en sus diversas modalidades, está en gran medida restringida en el entorno limitado de la aplicación por razones de seguridad. Estas restricciones rigen independientemente de si la aplicación carga información directamente desde un servidor o no. (Incluso el contenido de los archivos, las cadenas pegadas y la información introducida por el usuario pueden no ser fidedignos).

El origen del contenido de una página determina el entorno limitado al que se consigna. En el entorno limitado de la aplicación solo se pone el contenido cargado desde el directorio de la aplicación (el directorio de instalación al que remite el esquema `app:` de URL). El contenido cargado desde el sistema de archivos se pone en el entorno limitado *local con sistema de archivos* o *local de confianza*, que permite el acceso y la interacción con el contenido del sistema de archivos local, pero no con el contenido remoto. El contenido cargado desde la red se pone en un entorno limitado remoto que corresponde con su dominio de origen.

Para que una página de una aplicación pueda interactuar libremente con el contenido de un entorno limitado remoto, la página se puede asignar al mismo dominio que el contenido remoto. Por ejemplo, si es escribe una aplicación que muestra datos de mapas de un servicio de Internet, la página de la aplicación que descarga el contenido del servicio y lo presenta puede asignarse al dominio del servicio. Los atributos para asignar páginas en un entorno limitado remoto y un dominio son atributos nuevos que se añaden a los elementos de frame e iframe en HTML.

Para que el contenido de un entorno limitado ajeno a la aplicación pueda utilizar las funciones de AIR de forma segura, se puede establecer un puente de entorno limitado principal. Para que el contenido de la aplicación pueda llamar a métodos y tener acceso a propiedades del contenido de otros entornos limitados, se puede establecer un puente de entorno limitado secundario. La seguridad en este contexto significa que el contenido remoto no puede obtener accidentalmente referencias a objetos, propiedades o métodos que no están expuestos de forma explícita. Solo pueden pasar por el puente los tipos de datos, funciones y objetos anónimos sencillos. No obstante, aun así debe evitarse exponer explícitamente las funciones que sean potencialmente peligrosas. Por ejemplo: si expusiera una interfaz que permite que un contenido remoto lea y escriba archivos en cualquier parte del sistema del usuario, podría estar facilitando al contenido remoto los medios para causar graves daños al usuario.

Función eval() de JavaScript

Adobe AIR 1.0 y posterior

Una vez que ha terminado de cargarse una página, el uso de la función `eval()` queda restringido al entorno limitado de la aplicación. Se admiten algunos usos para que datos en formato JSON puedan analizarse con seguridad, pero toda evaluación que arroje sentencias ejecutables producirá un error. En "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130 se describen los usos admitidos de la función `eval()`.

Constructores Function

Adobe AIR 1.0 y posterior

En el entorno limitado de la aplicación pueden utilizarse constructores de función antes de que se termine de cargar una página. Una vez que han terminado todos los controladores de eventos `load` de la página, no se pueden crear funciones nuevas.

Carga de scripts externos

Adobe AIR 1.0 y posterior

Las páginas en HTML del entorno limitado de la aplicación no pueden utilizar la etiqueta `script` para cargar archivos JavaScript desde fuera del directorio de la aplicación. Para que una página de la aplicación cargue un script desde fuera del directorio de la aplicación, la página debe asignarse a un entorno limitado externo.

Objeto XMLHttpRequest

Adobe AIR 1.0 y posterior

AIR proporciona un objeto XMLHttpRequest (XHR) que pueden utilizar las aplicaciones para realizar peticiones de datos. El ejemplo siguiente ilustra una petición de datos sencilla:

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

A diferencia de los navegadores, AIR permite que el contenido que se ejecuta en el entorno limitado de la aplicación solicite datos de cualquier dominio. El resultado de una petición XHR que contiene una cadena JSON puede producir objetos de datos, a menos que el resultado también contenga código ejecutable. Si hay sentencias ejecutables en el resultado de la XHR, se emite un error y el intento de evaluación falla.

Para evitar la inyección accidental de código desde fuentes remotas, las XHR sincrónicas devuelven un resultado vacío si se realizan antes de haberse terminado de cargar la página. Las XHR asíncronas siempre devuelven un resultado después de haberse cargado una página.

De forma predeterminada, AIR bloquea las peticiones XMLHttpRequest entre dominios en los entornos limitados ajenos a la aplicación. Una ventana principal en el entorno limitado de la aplicación puede optar por permitir las peticiones entre dominios en un fotograma secundario que contenga contenido en un entorno limitado ajeno a la aplicación definiendo `allowCrossDomainXHR` -un atributo que añade AIR- en `true` en el elemento frame o iframe contenedor:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

Nota: *si conviene, la clase URLStream de AIR también puede utilizarse para descargar datos.*

Si se distribuye una petición XMLHttpRequest a un servidor remoto desde un fotograma o iframe que contenga contenido de la aplicación que se ha asignado a un entorno limitado remoto, asegúrese de que la URL de asignación no oculte la dirección del servidor que se utiliza en la XHR. Tomemos por ejemplo la siguiente definición de iframe, que asigna contenido de la aplicación a un entorno limitado remoto para el dominio example.com:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

Debido a que el atributo `sandboxRoot` reasigna la URL raíz de la dirección www.example.com, todas las peticiones se cargan desde el directorio de la aplicación y no desde el servidor remoto. Las peticiones se reasignan independientemente de si derivan de haber visitado distintas páginas o de una petición XMLHttpRequest.

Para evitar bloquear accidentalmente las peticiones de datos al servidor remoto, asigne `sandboxRoot` a un subdirectorio de la URL remota y no a la raíz. No es necesario que exista el directorio. Por ejemplo, para permitir que las peticiones a www.example.com se carguen desde el servidor remoto en lugar del directorio de la aplicación, cambie el iframe anterior a:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

En este caso, solo se carga localmente el contenido del subdirectorio `air`.

Para obtener más información, consulte "Elementos frame e iframe de HTML" en la página 1011 y "Seguridad HTML en Adobe AIR" en la página 1127.

Cookies

Adobe AIR 1.0 y posterior

En las aplicaciones de AIR solo el contenido de los entornos limitados remotos (contenido cargado desde fuentes http: y https:) puede utilizar cookies (la propiedad `document.cookie`). En el entorno limitado de la aplicación, existen otros medios para almacenar datos persistentes, incluyendo las clases EncryptedLocalStore, SharedObject y FileStream.

Objeto Clipboard

Adobe AIR 1.0 y posterior

La API Clipboard del WebKit funciona con los eventos `copy`, `cut` y `paste`. El objeto de evento que se pasa en estos eventos proporciona acceso al portapapeles a través de la propiedad `clipboardData`. Para leer o escribir datos en el portapapeles, utilice los siguientes métodos del objeto `clipboardData`:

Método	Descripción
clearData(mimeType)	Borra los datos del portapapeles. Defina el parámetro `mimeType` en el tipo MIME de los datos a borrar.
getData(mimeType)	Obtiene los datos del portapapeles. Este método solo se puede llamar en un controlador del evento `paste`. Defina el parámetro `mimeType` en el tipo MIME de los datos a devolver.
setData(mimeType, data)	Copia datos en el portapapeles. Defina el parámetro `mimeType` en el tipo MIME de los datos.

El código JavaScript que esté fuera del entorno limitado de la aplicación solo tiene acceso al portapapeles a través de estos eventos. No obstante, el contenido del entorno limitado de la aplicación tiene acceso directo al portapapeles del sistema a través de la clase Clipboard de AIR. Por ejemplo, se puede utilizar la siguiente sentencia para obtener datos en formato de texto del portapapeles:

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

Los tipos de datos MIME válidos son:

Tipo MIME	Valor
Texto	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
Mapa de bits	"image/x-vnd.adobe.air.bitmap"
Lista de archivos	"application/x-vnd.adobe.air.file-list"

Importante: *únicamente el contenido del entorno limitado de la aplicación tiene acceso a los datos de archivo que hay en el portapapeles. Si un contenido ajeno a la aplicación intenta acceder a un objeto de archivo desde el portapapeles, se emite un error de seguridad.*

Para obtener más información sobre el uso del portapapeles, consulte "Copiar y pegar" en la página 621 y Using the Pasteboard from JavaScript (Uso del Pasteboard de JavaScript) (Centro de desarrollo de Apple).

Arrastrar y colocar

Adobe AIR 1.0 y posterior

Los gestos de arrastrar y colocar hacia dentro y hacia fuera de HTML producen los siguientes eventos del DOM: `dragstart`, `drag`, `dragend`, `dragenter`, `dragover`, `dragleave` y `drop`. El objeto de evento que se pasa en estos eventos proporciona acceso a los datos arrastrados a través de la propiedad `dataTransfer`. La propiedad `dataTransfer` hace referencia a un objeto que proporciona los mismos métodos que el objeto `clipboardData` asociado con un evento clipboard. Por ejemplo, se puede utilizar la siguiente función para obtener datos en formato de texto de un evento `drop`:

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

El objeto `dataTransfer` tiene los siguientes miembros importantes:

Miembro	Descripción
clearData(mimeType)	Borra los datos. Defina el parámetro `mimeType` en el tipo MIME de la representación de datos a borrar.
getData(mimeType)	Obtiene los datos arrastrados. Este método solo se puede llamar en un controlador del evento `drop`. Defina el parámetro `mimeType` en el tipo MIME de los datos a obtener.
setData(mimeType, data)	Defina los datos a arrastrar. Defina el parámetro `mimeType` en el tipo MIME de los datos.
types	Un conjunto de cadenas que contiene los tipos MIME de todas las representaciones de datos que están disponibles actualmente en el objeto `dataTransfer`.
effectsAllowed	Especifica si los datos arrastrados se pueden copiar, mover o vincular, o una combinación de estas operaciones. Defina la propiedad `effectsAllowed` en el controlador del evento `dragstart`.
dropEffect	Especifica qué efectos de arrastrar admitidos se pueden utilizar con un destino de arrastre. Defina la propiedad `dropEffect` en el controlador del evento `dragEnter`. Durante la operación de arrastrar el cursor cambia para indicar qué efecto se produciría si el usuario soltara el ratón. Si no se especifica nada para `dropEffect`, se elige un efecto para la propiedad `effectsAllowed`. El efecto de copia tiene prioridad sobre el efecto de mover, que a su vez tiene prioridad sobre el efecto de vinculación. El usuario puede modificar la prioridad determinada con el teclado.

Para obtener más información sobre cómo ampliar la compatibilidad con arrastrar y colocar en una aplicación de AIR, consulte "Operación de arrastrar y colocar en AIR" en la página 633 y Using the Drag-and-Drop from JavaScript (Uso de arrastrar y colocar de JavaScript) (Centro de desarrollo de Apple).

Propiedades innerHTML y outerHTML

Adobe AIR 1.0 y posterior

AIR impone restricciones de seguridad en el uso de las propiedades `innerHTML` y `outerHTML` para contenido que se ejecuta en el entorno limitado de la aplicación. Previo al evento de cargar la página, así como durante la ejecución de un controlador de eventos de carga, el uso de las propiedades `innerHTML` y `outerHTML` es irrestricto. Sin embargo, una vez cargada la página solo se puede utilizar una propiedad `innerHTML` o `outerHTML` para añadir contenido estático al documento. Se pasa por alto toda sentencia en la cadena asignada a `innerHTML` o `outerHTML` que produzca código ejecutable. Por ejemplo, si se incluye un atributo callback de evento en la definición de un elemento, no se añade el detector de eventos. Asimismo, no se evalúan las etiquetas `<script>`. Para obtener más información, consulte "Seguridad HTML en Adobe AIR" en la página 1127.

Métodos Document.write() y Document.writeln()

Adobe AIR 1.0 y posterior

El uso de los métodos `write()` y `writeln()` no está restringido en el entorno limitado de la aplicación previo al evento `load` de la página. Sin embargo, una vez cargada la página, al llamar a cualquiera de estos dos métodos la página no se borra ni se crea una nueva. En un entorno limitado ajeno a la aplicación, al igual que con la mayoría de los navegadores web, cuando se llama a `document.write()` o `writeln()` tras haberse terminado de cargar una página, la página actual se borra y se abre una nueva página en blanco.

Propiedad Document.designMode

Adobe AIR 1.0 y posterior

Defina la propiedad `document.designMode` en `on` para que todos los elementos del documento sean editables. Entre las funciones integradas del editor se encuentran las de editar texto, copiar, pegar, y arrastrar y pegar. Definir `designMode` en `on` equivale a definir la propiedad `contentEditable` del elemento `body` en `true`. La propiedad `contentEditable` puede utilizarse en la mayoría de los elementos HTML para definir qué secciones de un documento son editables. Para obtener más información, consulte "Atributo HTML contentEditable" en la página 1014.

Eventos unload (para objetos body y frameset)

Adobe AIR 1.0 y posterior

En la etiqueta de nivel superior `frameset` o `body` de una ventana (incluida la ventana principal de la aplicación) no se debe utilizar un evento `unload` para responder a la acción de cerrar la ventana (o aplicación). En su lugar, utilice el evento `exiting` del objeto NativeApplication (para detectar cuándo se cierra una aplicación). O bien, utilice el evento `closing` del objeto NativeWindow (para detectar cuándo se cierra una ventana). Por ejemplo, el siguiente código JavaScript presenta un mensaje (`"Goodbye."`) cuando el usuario cierra la aplicación:

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

No obstante, los scripts *sí pueden* responder satisfactoriamente al evento `unload` provocado por la navegación por el contenido de un fotograma, iframe o ventana de nivel superior.

Nota: *es posible que estas limitaciones se eliminen en una versión futura de Adobe AIR.*

Objeto Window en JavaScript

Adobe AIR 1.0 y posterior

El objeto Window (ventana) sigue siendo el objeto global en el contexto de la ejecución de JavaScript. En el entorno limitado de la aplicación, AIR añade nuevas propiedades al objeto Window de JavaScript para facilitar el acceso a las clases integradas de AIR, además de importantes objetos host. Por otra parte, algunos métodos y propiedades se comportan de distinta manera, según se encuentren o no en el entorno limitado de la aplicación.

Propiedad Window.runtime La propiedad `runtime` permite instanciar y utilizar las clases runtime integradas desde el entorno limitado de la aplicación. Estas clases incluyen las API de AIR y Flash Player (pero no, por ejemplo, la arquitectura de Flex). Por ejemplo, la sentencia siguiente crea un objeto de archivo de AIR:

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

El archivo `AIRAliases.js`, incluido con el SDK de AIR, contiene definiciones de alias que permiten abreviar estas referencias. Por ejemplo, cuando se importa `AIRAliases.js` en una página, se puede crear un objeto File con la sentencia siguiente:

```
var preferencesFile = new air.File();
```

La propiedad `window.runtime` solo se define para contenido que se encuentra en el entorno limitado de la aplicación, y solo para el documento principal de una página con fotogramas o iframes.

Consulte el apartado "Uso del archivo AIRAliases.js" en la página 1027.

Propiedad Window.nativeWindow La propiedad `nativeWindow` proporciona una referencia al objeto de ventana nativa subyacente. Con esta propiedad se pueden programar funciones y propiedades de ventanas como posición en la pantalla, tamaño y visibilidad, así como controlar eventos de ventanas como cerrar, redimensionar y trasladar. Por ejemplo, la sentencia siguiente cierra la ventana:

```
window.nativeWindow.close();
```

Nota: *las funciones de control de ventanas que ofrece el objeto NativeWindow coinciden en parte con las funciones que brinda el objeto Window de JavaScript. En estos casos, se puede utilizar el método que más convenga.*

La propiedad `window.nativeWindow` solo se define para contenido que se encuentra en el entorno limitado de la aplicación, y solo para el documento principal de una página con fotogramas o iframes.

Propiedad Window.htmlLoader La propiedad `htmlLoader` proporciona una referencia al objeto HTMLLoader de AIR que contiene el contenido HTML. Con esta propiedad se puede programar el aspecto y comportamiento del entorno HTML. Por ejemplo, se puede utilizar la propiedad `htmlLoader.paintsDefaultBackground` para determinar si el control pinta un fondo blanco predeterminado:

```
window.htmlLoader.paintsDefaultBackground = false;
```

Nota: *el objeto HTMLLoader en sí tiene una propiedad `window` que hace referencia al objeto Window de JavaScript del contenido HTML que contiene. Esta propiedad puede utilizarse para acceder al entorno JavaScript a través de una referencia al HTMLLoader contenedor.*

La propiedad `window.htmlLoader` solo se define para contenido que se encuentra en el entorno limitado de la aplicación, y solo para el documento principal de una página con fotogramas o iframes.

Propiedades Window.parentSandboxBridge y Window.childSandboxBridge Las propiedades `parentSandboxBridge` y `childSandboxBridge` permiten definir una interfaz entre un fotograma principal y uno secundario. Para obtener más información, consulte "Uso de scripts entre contenidos en diferentes entornos limitados de seguridad" en la página 1038.

Funciones Window.setTimeout() y Window.setInterval() AIR impone restricciones de seguridad en el uso de las funciones `setTimeout()` y `setInterval()` en el entorno limitado de la aplicación. Cuando se llama a `setTimeout()` o `setInterval()`, el código a ejecutarse no se puede definir en forma de cadena, sino que se debe utilizar una referencia a una función. Para obtener más información, consulte el apartado "setTimeout() y setInterval()" en la página 1024.

Función Window.open() Cuando lo llama un código que se ejecuta en un entorno limitado ajeno a la aplicación, el método `open()` solo abre una ventana si la llamada es el resultado de una interacción del usuario (por ejemplo, tras haber pulsado una tecla o el botón del ratón). Además, el título de la ventana tiene como prefijo el título de la aplicación (para evitar que las ventanas abiertas por contenido remoto se hagan pasar por ventanas abiertas por la aplicación). Para obtener más información, consulte "Restricciones para llamar al método window.open() JavaScript" en la página 1133.

Objeto air.NativeApplication

Adobe AIR 1.0 y posterior

El objeto NativeApplication facilita información sobre el estado de la aplicación, distribuye varios eventos importantes a nivel de aplicación y ofrece funciones de utilidad para controlar el comportamiento de la aplicación. Se crea automáticamente una única instancia del objeto NativeApplication, a la que se tiene acceso a través de la propiedad `NativeApplication.nativeApplication` definida por la clase.

Para acceder al objeto desde el código JavaScript se puede usar:

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

O bien, si se ha importado el script `AIRAliases.js`, se puede utilizar la forma abreviada:

```
var app = air.NativeApplication.nativeApplication;
```

Solo se tiene acceso al objeto NativeApplication desde el entorno limitado de la aplicación. Para obtener más información sobre el objeto NativeApplication, consulte "Trabajo con información sobre el motor de ejecución de AIR y el sistema operativo" en la página 921.

Esquema URL de JavaScript

Adobe AIR 1.0 y posterior

La ejecución del código definido en un esquema URL de JavaScript (por ejemplo, en `href="javascript:alert('Test')"`) está bloqueada en el entorno limitado de la aplicación. No se emite ningún error.

HTML en AIR

Adobe AIR 1.0 y posterior

AIR y WebKit definen algunos elementos y atributos de HTML que no son estándar, entre ellos:

"Elementos frame e iframe de HTML" en la página 1011

"Controladores de eventos de elementos HTML" en la página 1013

Elementos frame e iframe de HTML

Adobe AIR 1.0 y posterior

AIR añade nuevos atributos a los elementos frame e iframe del contenido en el entorno limitado de la aplicación:

Atributo sandboxRoot El atributo `sandboxRoot` especifica otro dominio de origen, ajeno a la aplicación, para el archivo especificado por el atributo `src` del fotograma. El archivo se carga en el entorno limitado externo que corresponde al dominio especificado. El contenido del archivo y el contenido cargado desde el dominio especificado pueden usar scripts entre sí.

Importante: *si se define el valor de `sandboxRoot` en la URL de base del dominio, toda petición de contenido desde ese dominio se carga desde el directorio de la aplicación en lugar del servidor remoto (sea esa petición el resultado de visitas a distintas páginas, de una petición XMLHttpRequest o de cualquier otro medio de cargar contenido).*

Atributo documentRoot El atributo `documentRoot` especifica el directorio local del cual cargar las URL que se resuelven en archivos en el lugar especificado en `sandboxRoot`.

Al resolver las URL, sea en el atributo `src` del fotograma o en contenido cargado en el fotograma, la parte de la URL que coincide con el valor especificado en `sandboxRoot` se sustituye por el valor especificado en `documentRoot`. Por lo tanto, en la siguiente etiqueta de fotograma:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

`child.html` se carga desde el subdirectorio `sandbox` de la carpeta de instalación de la aplicación. Las URL relativas en `child.html` se resuelven partiendo de la base del directorio `sandbox`. Obsérvese que no se tiene acceso desde el fotograma a los archivos que se encuentren en el servidor remoto en `www.example.com/air`, puesto que AIR intentaría cargarlos desde el directorio app:/sandbox/.

Atributo allowCrossDomainXHR Para permitir que el contenido del fotograma realice peticiones del tipo XMLHttpRequests a cualquier dominio remoto, incluya el atributo `allowCrossDomainXHR="allowCrossDomainXHR"` en la etiqueta "frame" inicial. De forma predeterminada, el contenido ajeno a la aplicación solo puede realizar estas peticiones a su propio dominio de origen. Admitir peticiones XHR entre dominios conlleva graves implicaciones para la seguridad. El código de la página puede intercambiar datos con cualquier dominio. Si llegara a infiltrarse contenido malintencionado en la página, podrían verse comprometidos los datos que son accesibles al código en el entorno limitado actual. Habilite las peticiones XHR entre dominios solamente para las páginas que cree y controle usted mismo y solo cuando realmente resulte necesario cargar datos entre distintos dominios. También conviene validar cuidadosamente todos los datos externos que cargue la página para impedir la infiltración de código dañino u otra clase de ataque.

Importante: *si el elemento frame o iframe incluye el atributo `allowCrossDomainXHR`, significa que están habilitadas las peticiones XHR entre dominios (a menos que el valor asignado sea "0" o empiece con la letra "f" o "n"). Por ejemplo, si se define `allowCrossDomainXHR` en `"deny"`, aún estarían habilitadas las XHR entre dominios. Si no desea habilitar las peticiones entre dominios, omita el atributo de la declaración del elemento.*

Atributo ondominitialize Especifica un controlador de eventos para el evento `dominitialize` de un fotograma. Este evento es específico de AIR y se activa cuando se han creado los objetos de ventana y de documento del fotograma, pero antes de que se hayan analizado los scripts o creado los elementos de documento.

El fotograma distribuye el evento `dominitialize` lo bastante temprano en la secuencia de carga para que todo script en la página secundaria pueda hacer referencia a los objetos, variables y funciones que añada al documento secundario el controlador del evento `dominitialize`. Para poder añadir o tener acceso a los objetos de un documento secundario, la página principal debe encontrarse en el mismo entorno limitado que la página secundaria. No obstante, una página

principal que esté en el entorno limitado de la aplicación puede establecer un puente de entorno limitado para tener comunicación con el contenido de un entorno limitado externo.

Los ejemplos siguientes muestran el uso de la etiqueta iframe en AIR:

Para colocar el documento `child.html` en un entorno limitado remoto sin asignarlo a un dominio real en un servidor remoto:

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

Para colocar el documento `child.html` en un entorno limitado remoto y habilitar las peticiones del tipo XMLHttpRequests solamente a `www.example.com`:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

Para colocar el documento `child.html` en un entorno limitado remoto y habilitar las peticiones del tipo XMLHttpRequests a cualquier dominio remoto:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

Para colocar el documento `child.html` en un entorno limitado local con sistema de archivos:

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

Para colocar el documento `child.html` en un entorno limitado remoto, utilizando el evento `dominitialize` para establecer un puente de entorno limitado:

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

El siguiente documento `child.html` muestra cómo el contenido secundario puede acceder al puente de entorno limitado principal:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Para obtener más información, consulte "Uso de scripts entre contenidos en diferentes entornos limitados de seguridad" en la página 1038 y "Seguridad HTML en Adobe AIR" en la página 1127.

Controladores de eventos de elementos HTML

Adobe AIR 1.0 y posterior

Los objetos DOM en AIR y WebKit distribuyen algunos eventos que no se encuentran en el modelo de eventos DOM estándar. En la tabla siguiente se enumeran los atributos de eventos relacionados que se pueden utilizar para especificar los controladores para estos eventos:

Nombre de atributo de callback	Descripción
oncontextmenu	Se llama cuando se invoca un menú contextual, por ejemplo cuando se hace clic en texto seleccionado con el botón derecho o con la tecla Comando pulsada.
oncopy	Se llama al copiar una selección en un elemento.
oncut	Se llama al cortar una selección en un elemento.
ondominitialize	Se llama al crear el DOM de un documento cargado en un fotograma o iframe, pero antes de crear los elementos del DOM o de analizar los scripts.
ondrag	Se llama al arrastrar un elemento.
ondragend	Se llama al soltar un elemento arrastrado.
ondragenter	Se llama cuando un gesto de arrastrar entra en un elemento.
ondragleave	Se llama cuando un gesto de arrastrar sale de un elemento.
ondragover	Se llama continuamente cuando un gesto de arrastrar se encuentra dentro de los límites de un elemento.
ondragstart	Se llama al iniciarse un gesto de arrastrar.
ondrop	Se llama al soltarse el gesto de arrastrar estando sobre un elemento.
onerror	Se llama cuando se produce un error al cargar un elemento.
oninput	Se llama al introducir texto en un elemento de formulario.
onpaste	Se llama al pegar un artículo en un elemento.
onscroll	Se llama al desplazarse el contenido de un elemento desplazable.
onselectstart	Se llama al iniciarse una selección.

Atributo HTML contentEditable

Adobe AIR 1.0 y posterior

El atributo `contentEditable` puede añadirse a cualquier elemento HTML para permitir que los usuarios modifiquen el contenido del elemento. Por ejemplo, en el siguiente ejemplo el código HTML define todo el documento como editable, a excepción del primer elemento `p`:

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

Nota: *si la propiedad `document.designMode` se define en `on`, todos los elementos del documento serán modificables, independientemente de la configuración de `contentEditable` para un elemento individual. Sin embargo, la definición de `designMode` en `off`, no inhabilita la edición de elementos para los que el atributo `contentEditable` está definido en `true`. Para obtener más información, consulte "Propiedad Document.designMode" en la página 1008.*

URL data:

Adobe AIR 2 y posterior

AIR admite URL `data:` para los siguientes elementos:

- img
- input type="image"
- Reglas CSS que admitan imágenes (como background-image)

Las URL data: permiten insertar datos de imagen binarios directamente en un documento CSS o HTML como cadena con codificación base64. El siguiente ejemplo utiliza una URL data: como fondo repetido:

```
<html>
<head>
<style>
body {
background-
image:url('
0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAA
BF0lEQVR42uzZQQ7CMAxE0e%2F7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShX
BgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0FRmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCd
m1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBctAKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bS
qHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMiQsSFiQ8vRscTjlQOI2iHZwtpHuf
%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

Cuando utilice URL data:, no olvide que un espacio en blanco tiene importancia. Por ejemplo, la cadena de datos debe introducirse como una sola línea sin saltos. En caso contrario, los saltos de línea se tratan como parte de los datos y no es posible descodificar la imagen.

CSS en AIR

Adobe AIR 1.0 y posterior

WebKit admite varias propiedades de CSS ampliadas. Muchas de estas extensiones utilizan el prefijo: `-webkit`. Tenga en cuenta que algunas de estas extensiones son experimentales y podrían desaparecer en futuras versiones de WebKit. Para obtener más información sobre la compatibilidad de Webkit con CSS y sus extensiones en CSS, consulte la referencia de CSS de Safari.

Funciones de WebKit no admitidas en AIR

Adobe AIR 1.0 y posterior

AIR no admite las siguientes funcionalidades en WebKit ni en Safari 4:

- Mensajes entre dominios mediante window.postMessage (AIR suministra sus propias API de comunicación entre dominios)
- Variables CSS
- Fuentes SVG y Web Open Font Format (WOFF).
- Vídeo HTML y etiquetas de audio
- Consultas de dispositivos multimedia
- Caché de aplicaciones sin conexión
- Impresión (AIR proporciona su propia API PrintJob)
- Correctores ortográfico y gramatical
- SVG
- WAI-ARIA
- WebSockets (AIR proporciona sus propias API de socket)
- Trabajadores web
- API SQL de WebKit (AIR suministra su propia API)
- API de geolocalización de WebKit (AIR suministra su propia API de geolocalización en dispositivos que así lo admitan)
- API de carga de varios archivos de WebKit
- Eventos táctiles de WebKit (AIR suministra sus propios eventos táctiles)
- Lenguaje de marcado inalámbrico (WML)

A continuación se detallan API concretas de JavaScript, elementos HTML y propiedades CSS, así como valores no admitidos por AIR:

Miembros de objetos Window de JavaScript no admitidos:

- applicationCache()
- console

- openDatabase()
- postMessage()
- document.print()

Etiquetas HTML no admitidas:

- audio
- video

Atributos HTML no admitidos:

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate
- ontouchstart
- ontouchmove
- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern
- required
- sandbox

Eventos de JavaScript no admitidos:

- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove

- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

Propiedades CSS no admitidas:

- background-clip
- background-origin (usar -webkit-background-origin)
- background-repeat-x
- background-repeat-y
- background-size (usar -webkit-background-size)
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip
- -webkit-color-correction
- -webkit-font-smoothing

Valores CSS no admitidos:

- valores de la propiedad appearance:
  - media-volume-slider-container
  - media-volume-slider
  - media-volume-sliderthumb
  - outer-spin-button
- border-box (background-clip y background-origin)
- contain (background-size)
- content-box (background-clip y background-origin)
- cover (background-size)
- valores de la propiedad list:
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem

  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant
  - lower-norwegian
  - oromo
  - sidama
  - somali
  - tigre
  - tigrinya-er
  - tigrinya-er-abegede
  - tigrinya-et
  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee (propiedad display)

Capítulo 59: Programación con HTML y JavaScript en AIR

Adobe AIR 1.0 y posterior

Una serie de temas de programación son imprescindibles para desarrollar las aplicaciones de Adobe® AIR® con HTML y JavaScript. La siguiente información es importante si programa una aplicación de AIR basada en HTML o basada en SWF que ejecuta HTML y JavaScript utilizando la clase HTMLLoader (o el componente mx:HTML Flex™).

Información sobre la clase HTMLLoader

Adobe AIR 1.0 y posterior

La clase HTMLLoader de Adobe AIR define el objeto de visualización que puede mostrar contenido HTML en una aplicación de AIR. Las aplicaciones basadas en SWF pueden añadir un control HTMLLoader a una ventana existente o crear una ventana HTML que automáticamente contiene un objeto HTMLLoader con `HTMLLoader.createRootWindow()`. Se puede acceder al objeto HTMLLoader a través de la propiedad `window.htmlLoader` de JavaScript dentro de la página HTML cargada.

Carga del contenido HTML desde una dirección URL

Adobe AIR 1.0 y posterior

El siguiente código carga una dirección URL en un objeto HTMLLoader (añada HTMLLoader como elemento secundario del escenario u otro contenedor de objetos de visualización para ver el contenido HTML en su aplicación):

```
import flash.html.HTMLLoader;

var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
```

Las propiedades `width` y `height` de un objeto HTMLLoader se definen en 0 como valor predeterminado. Se deben definir estas dimensiones cuando se añade un objeto HTMLLoader al escenario. El HTMLLoader distribuye varios eventos cuando se carga una página. Se pueden utilizar estos eventos para determinar cuándo es seguro interactuar con la página cargada. Estos eventos se describen en "Gestión de eventos relacionados con HTML en AIR" en la página 1062.

Nota: *en la arquitectura de Flex, solo se pueden añadir las clases que amplían la clase UIComponent como elementos secundarios de los componentes de contenedor Flex. Por esta razón, no se puede añadir directamente un HTMLLoader como un elemento secundario de un componente de contenedor Flex; sin embargo, se puede utilizar el control Flex mx:HTML, se puede crear una clase personalizada que amplía UIComponent y contiene un HTMLLoader como un elemento secundario de UIComponent o se puede añadir el HTMLLoader como un elemento secundario de un UIComponent y añadir el UIComponent al contenedor Flex.*

También se puede representar texto HTML utilizando la clase TextField, pero sus prestaciones son limitadas. La clase TextField de Adobe® Flash® Player admite un subconjunto de marcado HTML, pero dadas las limitaciones de tamaño, las prestaciones son limitadas. (La clase HTMLLoader incluida en Adobe AIR no está disponible en Flash Player).

Carga del contenido HTML desde una cadena

Adobe AIR 1.0 y posterior

El método `loadString()` de un HTMLLoader carga una cadena de contenido HTML en el objeto HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
var htmlStr:String = "<html><body>Hello <b>world</b>.</body></html>";
html.loadString(htmlStr);
```

De forma predeterminada, el contenido cargado a través del método `loadString()` se sitúa en un entorno limitado ajeno a la aplicación con las siguientes características:

- Tiene acceso para cargar contenido desde la red (pero no desde el sistema de archivos).
- No puede cargar datos utilizando XMLHttpRequest.
- La propiedad `window.location` se establece en `"about:blank"`.
- El contenido no puede acceder a la propiedad `window.runtime` (al igual que puede el contenido de cualquier entorno limitado ajeno a la aplicación).

En AIR 1.5, la clase HTMLLoader incluye una propiedad `placeLoadStringContentInApplicationSandbox`. Cuando esta propiedad se establece en `true` para un objeto HTMLLoader, el contenido cargado a través del método `loadString()` se sitúa en el entorno limitado de la aplicación. (El valor predeterminado es `false`.) Con ello se proporciona acceso al contenido cargado con el método `loadString()` a la propiedad `window.runtime` y a todas las API de AIR. Si esta propiedad se define como `true`, compruebe que el origen de datos de una cadena utilizada en una llamada al método `loadString()` sea de confianza. Las instrucciones de código de la cadena HTML se ejecutan con todos los privilegios de la aplicación si esta propiedad se establece en `true`. Únicamente defina esta propiedad como `true` si está seguro de que la cadena no contiene código dañino.

En las aplicaciones compiladas con los SDK de AIR 1.0 o AIR 1.1, el contenido cargado a través del método `loadString()` se sitúa en el entorno limitado de la aplicación.

Reglas importantes de seguridad cuando se utiliza HTML en aplicaciones de AIR

Adobe AIR 1.0 y posterior

Los archivos que se instalan con la aplicación de AIR tienen acceso a las API de AIR. Por razones de seguridad, el contenido de otras fuentes no lo tienen. Por ejemplo, esta restricción impide que el contenido de un dominio remoto (como http://example.com) lea el contenido en el directorio del escritorio del usuario (o peor).

Dado que existen vulnerabilidades en la seguridad que se pueden abusar llamando a la función`eval()` (y API relacionadas), de forma predeterminada, el contenido que se instala con la aplicación tiene restricciones para utilizar estos métodos. Sin embargo, algunas arquitecturas de Ajax utilizan el llamado de la función `eval()` y API relacionadas.

Para estructurar el contenido adecuadamente para que funcione en una aplicación de AIR, se deben tener en cuenta las reglas de las restricciones de seguridad en el contenido proveniente de diferentes fuentes. El contenido que procede de diferentes fuentes se sitúa en clasificaciones de seguridad independientes, denominadas entornos limitados (consulte "Entornos limitados de seguridad" en la página 1087). Como valor predeterminado, el contenido instalado con la aplicación se instala en un entorno limitado denominado entorno limitado de *aplicación* y este entorno le concede acceso a las API de AIR. El entorno limitado de la aplicación normalmente es el entorno limitado más seguro, con restricciones diseñadas para prevenir la ejecución de código sospechoso.

El motor de ejecución permite cargar el contenido instalado con la aplicación en un entorno limitado diferente del entorno limitado de la aplicación. El contenido en los entornos limitados que no pertenecen a la aplicación funciona en un entorno de seguridad similar al de un típico navegador web. Por ejemplo, el código en entornos limitados que no pertenecen a la aplicación puede utilizar el método `eval()` y métodos relacionados (pero a la vez no puede acceder a las API de AIR). El motor de ejecución incluye maneras para que el contenido en diferentes entornos limitados se comuniquen de modo seguro (sin exponer a las API de AIR a contenidos que no sean de la aplicación). Para más información, consulte "Uso de scripts entre contenidos en diferentes entornos limitados de seguridad" en la página 1038.

Si se llama al código que no se puede utilizar en un entorno limitado por razones de seguridad, el tiempo de ejecución distribuye un error de JavaScript: "Adobe AIR runtime security violation for JavaScript code in the application security sandbox" (Infracción de seguridad del motor de ejecución de Adobe AIR para el código JavaScript en el entorno limitado de seguridad de la aplicación).

Para evitar este error, siga las prácticas de codificación que se describen en la siguiente sección, "Cómo evitar errores de JavaScript relacionados con la seguridad" en la página 1021.

Para obtener más información, consulte "Seguridad HTML en Adobe AIR" en la página 1127.

Cómo evitar errores de JavaScript relacionados con la seguridad

Adobe AIR 1.0 y posterior

Si se llama al código que no se puede utilizar en un entorno limitado debido a estas restricciones de seguridad, el tiempo de ejecución distribuye un error de JavaScript: "Adobe AIR runtime security violation for JavaScript code in the application security sandbox" (Infracción de seguridad del motor de ejecución de Adobe AIR para el código JavaScript en el entorno limitado de seguridad de la aplicación). Para evitar este error, siga estas prácticas de codificación.

Causas de errores de JavaScript relacionados con la seguridad

Adobe AIR 1.0 y posterior

El código que se ejecuta en el entorno limitado de la aplicación está restringido de la mayoría de las operaciones que requieren evaluar y ejecutar cadenas una vez que el evento de documento `load` se ha desencadenado y se ha salido de cualquier controlador de evento `load`. Si se intentan utilizar los siguientes tipos de sentencias JavaScript que evalúan y ejecutan cadenas potencialmente no seguras, se generan errores de JavaScript:

- Función eval()
- setTimeout() y setInterval()

- Constructor Function

  Además, los siguientes tipos de sentencias JavaScript fallan sin generar un error JavaScript no seguro:

- URL de JavaScript
- Callbacks de eventos asignados a través de atributos onevent en sentencias innerHTML y outerHTML
- Carga de archivos JavaScript desde fuera del directorio de instalación de la aplicación
- document.write() y document.writeln()
- XMLHttpRequests sincrónicos antes del evento load o durante un controlador de evento load
- Elementos de script creados dinámicamente

  Nota: *en algunos casos restringidos, se permite la evaluación de cadenas. Para más información, consulte "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130.*

  Adobe dispone de una lista de arquitecturas de Ajax que admiten el entorno limitado de seguridad de aplicación, en http://www.adobe.com/go/airappsandboxframeworks_es.

  Las siguientes secciones describen la manera de reescribir los scripts para evitar estos errores de JavaScript no seguros y fallos sin mensaje para el código que se ejecuta en el entorno limitado de la aplicación.

Asignación de contenido de aplicación en un entorno limitado diferente

Adobe AIR 1.0 y posterior

En la mayoría de los casos, puede reescribir o reestructurar una aplicación para evitar errores de JavaScript relacionados con la seguridad. Sin embargo, cuando no se puede rescribir ni reestructurar, se puede cargar el contenido de la aplicación en un entorno limitado diferente utilizando la técnica descrita en "Carga de contenido de aplicación en un entorno limitado que no pertenece a la aplicación" en la página 1039. Si dicho contenido también debe acceder a las API de AIR, se puede crear un puente de entorno limitado, como se describe en "Definición de una interfaz de puente de entorno limitado" en la página 1040.

Función eval()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En el entorno limitado de la aplicación, la función `eval()` solo se puede utilizar antes de un evento `load` de página o durante un controlador de evento `load`. Una vez que la página se haya cargado, las llamadas a `eval()` no ejecutarán el código. Sin embargo, en los siguientes casos se puede reescribir el código para evitar el uso de `eval()`.

Asignación de propiedades a un objeto

Adobe AIR 1.0 y posterior

En lugar de analizar una cadena para crear el acceso a la propiedad:

```
eval("obj." + propName + " = " + val);
```

acceso a las propiedades con sintaxis de corchete:

```
obj[propName] = val;
```

Creación de una función con variables disponibles en el contexto

Adobe AIR 1.0 y posterior

Reemplace sentencias como la siguiente:

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

con:

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

Creación de un objeto utilizando el nombre de la clase como un parámetro de cadena

Adobe AIR 1.0 y posterior

Considere una clase hipotética de JavaScript definida en el siguiente código:

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

La manera más simple de crear una instancia sería utilizar `eval()`:

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

Sin embargo, puede evitar la llamada a `eval()` analizando cada componente de la clase name y crear un nuevo objeto utilizando la sintaxis de corchete:

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

Para crear la instancia, utilice:

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

setTimeout() y setInterval()

Adobe AIR 1.0 y posterior

Reemplace la cadena analizada como la función de controlador con un objeto o referencia de función. Por ejemplo, reemplace una sentencia como:

```
setTimeout("alert('Timeout')", 100);
```

con:

```
setTimeout(function(){alert('Timeout')}, 100);
```

O bien, cuando la función requiere que el autor de llamada defina el objeto `this`, reemplace una sentencia como:

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

con la siguiente:

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

Constructor Function

Adobe AIR 1.0 y posterior

Las llamadas a `new Function(param, body)` se pueden reemplazar con una declaración de función en línea o utilizar solo antes de que se haya controlado el evento de página `load`.

URL de JavaScript

Adobe AIR 1.0 y posterior

El código definido en un vínculo utilizando el esquema de URL javascript: se omite en el entorno limitado de la aplicación. No se genera ningún error de JavaScript de seguridad. Se pueden reemplazar vínculos utilizando URL javascript:, como:

```
<a href="javascript:code()">Click Me</a>
```

con:

```
<a href="#" onclick="code()">Click Me</a>
```

Callbacks de eventos asignados a través de atributos onevent en sentencias innerHTML y outerHTML

Adobe AIR 1.0 y posterior

Cuando se utiliza innerHTML o outerHTML para añadir elementos al DOM de un documento, se omite cualquier callback de evento dentro de la sentencia, como `onclick` o `onmouseover`. No se genera ningún error de seguridad. En cambio, se puede asignar un atributo `id` a los nuevos elementos y definir las funciones de callback de controlador de eventos utilizando el método `addEventListener()`.

Por ejemplo, dado un elemento de destino en un documento, como:

```
<div id="container"></div>
```

Reemplace las sentencias como:

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

con:

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

Carga de archivos JavaScript desde fuera del directorio de instalación de la aplicación

Adobe AIR 1.0 y posterior

No se permite la carga de archivos de script desde fuera del entorno limitado de la aplicación. No se genera ningún error de seguridad. Todos los archivos de script que se ejecutan en el entorno limitado de la aplicación se deben instalar en el directorio de la aplicación. Para utilizar scripts externos en una página, se debe asignar la página a un entorno limitado diferente. Consulte "Carga de contenido de aplicación en un entorno limitado que no pertenece a la aplicación" en la página 1039

document.write() y document.writeln()

Adobe AIR 1.0 y posterior

Se omiten las llamadas a `document.write()` o `document.writeln()` después de que se haya controlado el evento de página `load`. No se genera ningún error de seguridad. Como alternativa, se puede cargar un nuevo archivo o reemplazar el cuerpo del documento utilizando técnicas de manipulación DOM.

XMLHttpRequests sincrónicos antes del evento load o durante un controlador de evento load

Adobe AIR 1.0 y posterior

XMLHttpRequests sincrónicos iniciados antes del evento de página `load` o durante un controlador de eventos `load` no devuelven contenido. Se pueden iniciar XMLHttpRequests asíncronos pero no devuelven contenido hasta después del evento `load`. Después de que se haya controlado el evento `load`, los XMLHttpRequests sincrónicos se comportan normalmente.

Elementos de script creados dinámicamente

Adobe AIR 1.0 y posterior

Se omiten los elementos de script creados dinámicamente, como cuando se crean con el método `document.createElement()` o innerHTML.

Acceso a las clases de API de AIR desde JavaScript

Adobe AIR 1.0 y posterior

Además de los elementos estándar y ampliados de Webkit, el código HTML y JavaScript puede acceder a las clases host proporcionadas por el motor de ejecución. Estas clases permiten acceder a las funciones avanzadas que brinda AIR, incluyendo:

- Acceso al sistema de archivos
- Uso de las bases de datos SQL locales
- Control de los menús de aplicación y de ventana
- Acceso a los sockets para red
- Uso de clases y objetos definidos por el usuario
- Prestaciones de sonido

  Por ejemplo, la API del archivo de AIR incluye una clase File, contenida en el paquete flash.filesystem. Se puede crear un objeto File en JavaScript de la siguiente manera:

```
var myFile = new window.runtime.flash.filesystem.File();
```

El objeto `runtime` es un objeto especial de JavaScript, disponible para el contenido HTML ejecutándose en AIR en el entorno limitado de la aplicación. Permite acceder a las clases runtime desde JavaScript. La propiedad `flash` del objeto `runtime` proporciona acceso al paquete flash. A su vez, la propiedad `flash.filesystem` del objeto `runtime` proporciona acceso al paquete flash.filesystem (y este paquete incluye la clase File). Los paquetes son una manera de organizar las clases que se utilizan en ActionScript.

Nota: *la propiedad `runtime` no se añade automáticamente a los objetos window de las páginas en un fotograma o iframe. Sin embargo, mientras que el documento secundario se encuentra en el entorno limitado de la aplicación, el elemento secundario puede acceder a la propiedad `runtime` del elemento principal.*

Dado que la estructura del paquete de las clases runtime requieren que los desarrolladores escriban largas cadenas de código JavaScript para acceder a cada clase (como en `window.runtime.flash.desktop.NativeApplication`) AIR SDK incluye un archivo AIRAliases.js que permite acceder a las clases runtime de forma más fácil (por ejemplo, simplemente escribiendo `air.NativeApplication`).

Las clases API de AIR se describen en esta guía. Otras clases de la API de Flash Player, que pueden ser de interés para los desarrolladores de HTML, se describen en *Adobe AIR API Reference for HTML Developers* (solo disponible en inglés). ActionScript es el lenguaje que se utiliza en el contenido SWF (Flash Player). Sin embargo, las sintaxis de JavaScript y ActionScript son similares. (Ambas se basan en versiones del lenguaje ECMAScript.) Todas las clases incorporadas están disponibles en JavaScript (en el contenido HTML) y en ActionScript (en el contenido SWF).

Nota: *el código JavaScript no puede utilizar las clases Dictionary, XML y XMLList, que están disponibles en ActionScript.*

Uso del archivo AIRAliases.js

Adobe AIR 1.0 y posterior

Las clases runtime están organizadas en una estructura de paquete, como se muestra a continuación:

- `window.runtime.flash.desktop.NativeApplication`
- `window.runtime.flash.desktop.ClipboardManager`
- `window.runtime.flash.filesystem.FileStream`
- `window.runtime.flash.data.SQLDatabase`

En AIR SDK se incluye un archivo AIRAliases.js que proporciona definiciones de “alias” que permiten acceder a las clases runtime sin tener que escribir tanto. Por ejemplo, se puede acceder a las clases listadas arriba simplemente escribiendo:

- `air.NativeApplication`
- `air.Clipboard`
- `air.FileStream`
- `air.SQLDatabase`

Esta lista es solo un breve subconjunto de las clases en el archivo AIRAliases.js. La lista completa de las funciones a nivel de paquete y clases se suministra en *Adobe AIR API Reference for HTML Developers* (solo disponible en inglés).

Además de las clases runtime comúnmente utilizadas, el AIRAliases.js incluye aliases para funciones a nivel de paquete comúnmente utilizadas: `window.runtime.trace()`, `window.runtime.flash.net.navigateToURL()` y `window.runtime.flash.net.sendToURL()`, que forman alises como `air.trace()`, `air.navigateToURL()` y `air.sendToURL()`.

Para utilizar el archivo AIRAliases.js, se debe incluir la siguiente referencia `script` en la página HTML:

```
<script src="AIRAliases.js"></script>
```

Si es necesario, ajuste la ruta en la referencia `src`.

Importante: *excepto donde se notifica, el código de ejemplo de JavaScript en esta documentación supone que se ha incluido el archivo AIRAliases.js en la página HTML.*

Información sobre URL en AIR

Adobe AIR 1.0 y posterior

En el contenido HTML que se ejecuta en AIR, se puede utilizar cualquiera de los siguientes esquemas de URL para definir atributos `src` para las etiquetas `img`, `frame`, `iframe` y `script` en el atributo `href` de una etiqueta `link` o en cualquier lugar donde se puede suministrar una URL.

Esquema de URL	Descripción	Ejemplo
file	Una ruta relativa a la raíz del sistema de archivos.	`file:///c:/AIR Test/test.txt`
app	Una ruta relativa al directorio raíz de la aplicación instalada.	`app:/images`
app-storage	Una ruta relativa al directorio de almacenamiento de aplicación. Para cada aplicación instalada, AIR define un directorio de almacenamiento de aplicación exclusivo, que es un lugar útil para almacenar datos específicos a esa aplicación.	`app-storage:/settings/prefs.xml`
http	Una petición HTTP estándar.	`http://www.adobe.com`
https	Una petición HTTPS estándar.	`https://secure.example.com`

Para más información sobre el uso de esquemas de URL en AIR, consulte “Esquemas de URI” en la página 846.

Muchas de las API de AIR, incluyendo las clases File, Loader, URLStream y Sound, utilizan un objeto URLRequest en lugar de una cadena que contiene la URL. El objeto URLRequest mismo se inicializa con una cadena, que puede utilizar cualquiera de los mismos esquemas de URL. Por ejemplo, la siguiente sentencia crea un objeto URLRequest que se puede utilizar para solicitar la página de inicio de Adobe:

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

Para más información sobre objetos URLRequest, consulte “Comunicaciones HTTP” en la página 844.

Disponibilidad de objetos ActionScript en JavaScript

Adobe AIR 1.0 y posterior

JavaScript en la página HTML cargada por un objeto HTMLLoader puede llamar a las clases, objetos y funciones definidas en el contexto de ejecución de ActionScript utilizando las propiedades `window.runtime`, `window.htmlLoader` y `window.nativeWindow` de la página HTML. También se pueden hacer disponibles los objetos ActionScript y las funciones para el código JavaScript creando referencias a los mismos dentro del contexto de ejecución de JavaScript.

Ejemplo básico de acceso a los objetos JavaScript desde ActionScript

Adobe AIR 1.0 y posterior

En el siguiente ejemplo se muestra la manera de añadir propiedades haciendo referencia a objetos ActionScript al objeto global window de una página HTML:

```
var html:HTMLLoader = new HTMLLoader();
var foo:String = "Hello from container SWF."
function helloFromJS(message:String):void {
    trace("JavaScript says:", message);
}
var urlReq:URLRequest = new URLRequest("test.html");
html.addEventListener(Event.COMPLETE, loaded);
html.load(urlReq);

function loaded(e:Event):void{
    html.window.foo = foo;
    html.window.helloFromJS = helloFromJS;
}
```

El contenido HTML (en un archivo denominado test.html) cargado en el objeto HTMLLoader en el ejemplo anterior puede acceder a la propiedad `foo` y al método `helloFromJS()` definido en el archivo principal SWF:

```
<html>
    <script>
        function alertFoo() {
            alert(foo);
        }
    </script>
    <body>
        <button onClick="alertFoo()">
            What is foo?
        </button>
        <p><button onClick="helloFromJS('Hi.')">
            Call helloFromJS() function.
        </button></p>
    </body>
</html>
```

Cuando se accede al contexto JavaScript de un documento cargado, se puede utilizar el evento `htmlDOMInitialize` para crear objetos con suficiente anticipación en la secuencia de construcción de la página que cualquier script definido en la página puede acceder a los mismos. Si se espera para el evento `complete`, solo los scripts en la página que se ejecutan después del evento de página `load` pueden acceder a los objetos añadidos.

Disponibilidad de definiciones de clase en JavaScript

Adobe AIR 1.0 y posterior

Para hacer disponibles las clases ActionScript de la aplicación en JavaScript, se puede asignar el contenido HTML cargado al dominio de aplicación que contiene las definiciones de clase. El dominio de aplicación del contexto de ejecución de JavaScript se puede definir con la propiedad `runtimeApplicationDomain` del objeto HTMLLoader. Para establecer, por ejemplo, el dominio de aplicación al dominio de aplicación principal se debe definir `runtimeApplicationDomain` en `ApplicationDomain.currentDomain`, como se muestra en el siguiente código:

```
html.runtimeApplicationDomain = ApplicationDomain.currentDomain;
```

Una vez que la propiedad `runtimeApplicationDomain` se define, el contexto de JavaScript comparte las definiciones de clase con el dominio asignado. Para crear una instancia de una clase personalizada en JavaScript, se debe hacer referencia a la definición de clase a través de la propiedad `window.runtime` y utilizar el operador `new`:

```
var customClassObject = new window.runtime.CustomClass();
```

El contenido HTML debe ser de un dominio de seguridad compatible. Si el contenido HTML es de un dominio de seguridad diferente al del dominio de aplicación que asigna, la página utiliza un dominio de aplicación predeterminado. Por ejemplo, si carga una página remota de Internet, no podrá asignar ApplicationDomain.currentDomain como el dominio de aplicación de la página.

Eliminación de detectores de eventos

Adobe AIR 1.0 y posterior

Cuando se añaden detectores de eventos JavaScript a los objetos fuera de la página actual, incluyendo objetos runtime, objetos en contenido SWF cargado e inclusive objetos JavaScript ejecutándose en otras páginas, siempre se deben quitar esos detectores de eventos cuando se descarga la página. De lo contrario, el detector de evento distribuye el evento a una función de controlador que ya no existe. Si esto sucede, aparecerá el siguiente mensaje de error: “The application attempted to reference a JavaScript object in an HTML page that is no longer loaded", (La aplicación intentó realizar una referencia a un objeto JavaScript en una página HTML que ya no está cargada). Al quitar los detectores de eventos innecesarios también permite que AIR reclame la memoria asociada. Para más información, consulte “Eliminación de detectores de eventos de páginas HTML que permiten la navegación” en la página 1067.

Acceso a objetos JavaScript y DOM HTML desde ActionScript

Adobe AIR 1.0 y posterior

Una vez que el objeto HTMLLoader distribuye el evento `complete`, se puede acceder a todos los objetos en DOM HTML (modelo de objeto de documentos) para la página. Los objetos accesibles incluyen elementos de visualización (como objetos `div` y `p` en la página) así como funciones y variables de JavaScript. El evento `complete` corresponde al evento de página JavaScript `load`. Es posible que antes de distribuir el evento `complete`, los elementos DOM, variables, y funciones no se hayan analizado o creado. Si es posible, espere al evento `complete` antes de acceder al DOM HTML.

Por ejemplo, considere la siguiente página HTML:

```
<html>
    <script>
        foo = 333;
        function test() {
            return "OK.";
        }
    </script>
    <body>
        <p id="p1">Hi.</p>
    </body>
</html>
```

Esta simple página HTML define una variable JavaScript denominada *foo* y una función JavaScript denominada *test()*. Ambas son propiedades del objeto global `window` de la página. Asimismo, el objeto `window.document` incluye un elemento P (con el ID *p1*), que se puede acceder utilizando el método `getElementById()`. Una vez que se carga la página (cuando el objeto HTMLLoader distribuye el evento `complete`), se puede acceder a cada uno de estos objetos desde ActionScript, como se muestra en el siguiente código ActionScript:

```
var html:HTMLLoader = new HTMLLoader();
html.width = 300;
html.height = 300;
html.addEventListener(Event.COMPLETE, completeHandler);
var xhtml:XML =
    <html>
        <script>
            foo = 333;
            function test() {
                return "OK.";
            }
        </script>
        <body>
            <p id="p1">Hi.</p>
        </body>
    </html>;
html.loadString(xhtml.toString());

function completeHandler(e:Event):void {
    trace(html.window.foo); // 333
    trace(html.window.document.getElementById("p1").innerHTML); // Hi.
    trace(html.window.test()); // OK.
}
```

Para acceder al contenido de un elemento HTML, utilice la propiedad `innerHTML`. Por ejemplo, el código anterior utiliza `html.window.document.getElementById("p1").innerHTML` para obtener el contenido del elemento HTML denominado *p1*.

También se pueden definir propiedades de la página HTML desde ActionScript. Por ejemplo, en el siguiente ejemplo se define el contenido del elemento `p1` y el valor de la variable JavaScript `foo` en la página utilizando una referencia al objeto HTMLLoader que lo contiene:

```
html.window.document.getElementById("p1").innerHTML = "Goodbye";
html.window.foo = 66;
```

Incorporación de contenido SWF en HTML

Adobe AIR 1.0 y posterior

Se puede incorporar contenido SWF en el contenido HTML dentro de una aplicación de AIR de igual modo como se haría en un navegador. Incorpore el contenido SWF utilizando una etiqueta `object` una etiqueta `embed` o ambas.

Nota: *una práctica normal de desarrollo es utilizar ambas etiquetas `object` y `embed` para mostrar el contenido SWF en una página HTML. Esta práctica no tiene ninguna ventaja en AIR. Se puede utilizar la etiqueta `object` de estándar W3C por si sola en el contenido que se muestra en AIR. Al mismo tiempo, se puede continuar utilizando las etiquetas `object` y `embed` juntas, si es necesario, para el contenido HTML que también se muestra en un navegador.*

Si ha activado la transparencia en el objeto NativeWindow que muestra el contenido HTML y SWF, AIR no mostrará el contenido SWF si el modo de ventana (`wmode`) utilizado para incorporar el contenido se establece en el valor: `window`. Para ver contenido SWF en una página HTML de una ventana transparente, establezca el parámetro `wmode` como `opaque` o `transparent`. `window` es el valor predeterminado para `wmode`, por lo que si no especifica ningún valor, no podrá ver el contenido.

En el siguiente ejemplo se muestra el uso de la etiqueta HTML `object` para mostrar un archivo SWF dentro del contenido HTML. El parámetro `wmode` se establece como `opaque` para que el contenido se visualice, incluso si el objeto NativeWindow subyacente es transparente. El archivo SWF se carga desde el directorio de la aplicación, pero se puede utilizar cualquiera de los esquemas de URL admitidos por AIR. (La ubicación desde donde se carga el archivo SWF determina el entorno limitado de seguridad en el que AIR coloca el contenido).

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

También puede utilizar un script para cargar el contenido de forma dinámica. En el siguiente ejemplo se crea un nodo `object` para mostrar el archivo SWF especificado en el parámetro `urlString`. En el ejemplo se añade el nodo de un elemento secundario con el ID especificado por el parámetro `elementID`:

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

El contenido SWF no se visualiza si el objeto HTMLLoader se escala o se gira, o si la propiedad `alpha` se establece en un valor que no sea 1.0. Antes de AIR 1.5.2, el contenido SWF no se visualizaba en ventanas transparentes independientemente del valor dado `wmode`.

Nota: *cuando un objeto SWF incorporado intenta cargar un activo externo (por ejemplo, un archivo de vídeo), el contenido SWF podría no procesarse correctamente si no se proporciona una ruta absoluta al vídeo en el archivo HTML. Sin embargo, un objeto SWF incorporado sí puede cargar un archivo de imagen externa con una ruta relativa.*

El siguiente ejemplo describe cómo se pueden cargar activos externos a través de un objeto SWF incorporado en contenido HTML:

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

En el siguiente ejemplo de ActionScript, la ruta de imagen transferida por el archivo HTML se lee y la imagen se carga en el escenario

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

Uso de las bibliotecas de ActionScript en una página HTML

Adobe AIR 1.0 y posterior

AIR amplía el elemento de script HTML para que una página pueda importar clases ActionScript en un archivo SWF compilado. Por ejemplo, para importar una biblioteca llamada *myClasses.swf*, ubicada en el subdirectorio `lib` de la carpeta de aplicaciones raíz, se debe incluir la siguiente etiqueta de script dentro de un archivo HTML:

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

Importante: *el atributo type debe ser* `type="application/x-shockwave-flash"` *para que la biblioteca se cargue correctamente.*

Si el contenido SWF se compila como un archivo SWF de Flash Player 10 o AIR 1.5, debe establecer el espacio de nombres XML del archivo descriptor de la aplicación en el espacio de nombres de AIR 1.5.

El directorio `lib`y el archivo `myClasses.swf` también se deben incluir cuando se empaqueta el archivo de AIR.

Acceda a las clases importadas a través de la propiedad `runtime` del objeto Window de JavaScript:

```
var libraryObject = new window.runtime.LibraryClass();
```

Si las clases en el archivo SWF se organizan en paquetes, debe también incluir el nombre del paquete. Por ejemplo, si la definición LibraryClass está en un paquete denominado *utilities*, creará una instancia de la clase con la siguiente sentencia:

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

Nota: *para compilar una biblioteca SWF ActionScript para usar como parte de una página HTML en AIR, utilice el compilador* `acompc`*. La utilidad acompc forma parte de Flex SDK y se describe en la documentación de Flex SDK.*

Acceso a objetos JavaScript y DOM HTML desde un archivo importado ActionScript

Adobe AIR 1.0 y posterior

Para acceder a objetos en una página HTML desde ActionScript en un archivo SWF importado en la página utilizando la etiqueta `<script>`, pase una referencia a un objeto JavaScript, como `window` o `document`, a una función definida en el código ActionScript. Utilice la referencia dentro de la función para acceder al objeto JavaScript (u otros objetos accesibles a través de la referencia pasada).

Por ejemplo, considere la siguiente página HTML:

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

Esta simple página HTML tiene una variable JavaScript denominada *num* y una función JavaScript denominada *getStatus()*. Ambas son propiedades del objeto `window` de la página. Asimismo, el objeto `window.document` incluye un elemento P (con el ID *p1*).

La página carga un archivo ActionScript, "ASLibrary.swf," que contiene una clase, ASClass. ASClass define una función denominada `accessDOM()` que simplemente rastrea los valores de estos objetos JavaScript. El método `accessDOM()` toma al objeto Window de JavaScript como un argumento. Utilizando esta referencia Window, se puede acceder a otros objetos en la página incluyendo variables, funciones y elementos DOM como se muestra en la siguiente definición:

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

Se pueden obtener y definir las propiedades de la página HTML de una clase ActionScript importada. Por ejemplo, la siguiente función define el contenido del elemento `p1` en la página y define el valor de la variable `foo` de JavaScript en la página:

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

Conversión de los objetos Date y RegExp

Adobe AIR 1.0 y posterior

Los lenguajes JavaScript y ActionScript definen las clases Date y RegExp, pero los objetos de este tipo no se convierten automáticamente entre los dos contextos de ejecución. Se deben convertir los objetos Date y RegExp al tipo equivalente antes de utilizarlos para establecer propiedades o parámetros de función en el contexto de ejecución alternativo.

Por ejemplo, el siguiente código ActionScript convierte un objeto Date de JavaScript denominado `jsDate` a un objeto Date de ActionScript:

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

Por ejemplo, el siguiente código ActionScript convierte un objeto RegExp de JavaScript denominado `jsRegExp` a un objeto RegExp de ActionScript:

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

Manipulación de una hoja de estilo HTML de ActionScript

Adobe AIR 1.0 y posterior

Una vez que el objeto HTMLLoader ha distribuido el evento `complete`, puede examinar y manipular los estilos CSS en una página.

Por ejemplo, observe el siguiente documento HTML simple:

```
<html>
<style>
    .style1A { font-family:Arial; font-size:12px }
    .style1B { font-family:Arial; font-size:24px }
</style>
<style>
    .style2 { font-family:Arial; font-size:12px }
</style>
<body>
    <p class="style1A">
        Style 1A
    </p>
    <p class="style1B">
        Style 1B
    </p>
    <p class="style2">
        Style 2
    </p>
</body>
</html>
```

Una vez que un objeto HTMLLoader cargue este contenido, puede manipular los estilos CSS en la página a través del conjunto `cssRules` del conjunto `window.document.styleSheets`, tal y como se muestra a continuación:

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event):void {
    var styleSheet0:Object = html.window.document.styleSheets[0];
    styleSheet0.cssRules[0].style.fontSize = "32px";
    styleSheet0.cssRules[1].style.color = "#FF0000";
    var styleSheet1:Object = html.window.document.styleSheets[1];
    styleSheet1.cssRules[0].style.color = "blue";
    styleSheet1.cssRules[0].style.font-family = "Monaco";
}
```

Este código ajusta los estilos CSS para que el documento HTML resultante se asemeje al siguiente:

Style 1A

Style 1B

Style 2

Tenga en cuenta que el código puede añadir estilos a la página después de que el objeto HTMLLoader distribuye el evento `complete`.

Uso de scripts entre contenidos en diferentes entornos limitados de seguridad

Adobe AIR 1.0 y posterior

El modelo de seguridad de motor de ejecución aísla el código de diferentes orígenes. Al utilizar scripts entre contenidos en diferentes entornos limitados de seguridad, se puede permitir que el contenido de un entorno limitado de seguridad acceda a los métodos y propiedades seleccionadas en otro entorno limitado.

Entornos limitados de seguridad de AIR y código JavaScript

Adobe AIR 1.0 y posterior

AIR aplica una política de mismo origen que impide que el código en un dominio interactúe con el contenido en otro. Todos los archivos se colocan en un entorno limitado basado en su origen. Normalmente, el contenido en el entorno limitado de la aplicación no puede infringir el principio de mismo origen y uso de scripts entre contenidos cargados desde fuera del directorio de instalación de la aplicación. Sin embargo, AIR proporciona algunas técnicas que permiten el uso de scripts entre contenidos que no pertenecen a la aplicación.

Una técnica utiliza fotogramas o iframes para asignar el contenido de aplicación en un entorno limitado de seguridad diferente. Cualquier página que se carga desde el área del entorno limitado de la aplicación se comporta como si se cargara desde el dominio remoto. Por ejemplo, al asignar el contenido de aplicación al dominio *example.com*, dicho contenido podría hacer uso de scripts entre páginas desde example.com.

Dado que esta técnica coloca el contenido de aplicación en un entorno limitado diferente, el código dentro de dicho contenido tampoco está sujeto a las restricciones de ejecución de código en cadenas evaluadas. Se puede utilizar esta técnica de asignación de entorno limitado para facilitar estas restricciones aun cuando no necesita un contenido remoto de uso de scripts entre contenidos. La asignación de contenido en este modo puede ser especialmente útil cuando se trabaja con una de las muchas arquitecturas de JavaScript o con código existente que depende de cadenas de evaluación. Sin embargo, se debe considerar y estar atento al riesgo adicional de que el contenido sospechoso se podría insertar y ejecutar cuando el contenido se ejecuta fuera del entorno limitado de la aplicación.

Del mismo modo, el contenido de aplicación asignado a otro entorno limitado pierde el acceso a las API de AIR, por lo que la técnica de asignación de entorno limitado no se puede utilizar para exponer la funcionalidad de AIR al código ejecutado fuera del entorno limitado de la aplicación.

Otra técnica del uso de scripts entre contenidos permite crear una interfaz denominada *puente de entorno limitado* entre el contenido en un entorno limitado que no pertenece a la aplicación y el documento principal en el entorno limitado de la aplicación. El puente permite que el elemento secundario acceda a las propiedades y métodos definidos por el elemento principal, que el elemento principal acceda a las propiedades y a los métodos definidos por el elemento secundario o ambos.

También se puede realizar XMLHttpRequests entre dominios desde el entorno limitado de la aplicación y, opcionalmente, desde otros entornos limitados.

Para más información, consulte "Elementos frame e iframe de HTML" en la página 1011, "Seguridad HTML en Adobe AIR" en la página 1127 y "Objeto XMLHttpRequest" en la página 1005.

Carga de contenido de aplicación en un entorno limitado que no pertenece a la aplicación

Adobe AIR 1.0 y posterior

Para permitir que el contenido de aplicación use scripts entre contenidos cargados fuera del directorio de instalación de la aplicación, se pueden utilizar los elementos `frame` o `iframe` para cargar el contenido de aplicación en el mismo entorno limitado de seguridad que el contenido externo. Si no necesita usar el script entre contenido remoto, pero aún desea cargar una página de su aplicación fuera del entorno limitado de la aplicación, puede usar la misma técnica, especificando `http://localhost/` o cualquier otro valor inofensivo como el dominio de origen.

AIR añade los nuevos atributos, `sandboxRoot` y `documentRoot`, al elemento frame que permite especificar si el archivo de aplicación cargado en el fotograma se debe asignar a un entorno limitado que no pertenece a la aplicación. Los archivos que se resuelven en una ruta debajo de la URL `sandboxRoot` se cargan desde el directorio `documentRoot`. Por razones de seguridad, el contenido de aplicación cargado de este modo se lo trata como si se cargara desde una URL `sandboxRoot`.

La propiedad `sandboxRoot` especifica la URL que se debe utilizar para determinar el entorno limitado y dominio donde colocar el contenido del fotograma. Se deben utilizar los esquemas de URL `file:`, `http:` o `https:`. Si se especifica una URL relativa, el contenido permanece en el entorno limitado de la aplicación.

La propiedad `documentRoot` especifica el directorio desde donde cargar el contenido de fotograma. Se deben utilizar los esquemas de URL `file:`, `app:` o `app-storage:`.

En el siguiente ejemplo se asigna el contenido instalado en el subdirectorio `sandbox` de la aplicación para que se ejecute en el entorno limitado remoto y en el dominio `www.example.com`:

```
 <iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

La página `ui.html` podría cargar un archivo javascript desde la carpeta local `sandbox` utilizando la siguiente etiqueta de script:

```
<script src="http://www.example.com/local/ui.js"></script>
```

Asimismo, podría cargar el contenido desde un directorio en el servidor remoto utilizando una etiqueta de script como la siguiente:

```
<script src="http://www.example.com/remote/remote.js"></script>
```

La URL `sandboxRoot` enmascara cualquier contenido en la misma URL en el servidor remoto. En el ejemplo anterior, no se podría acceder a ningún contenido remoto en `www.example.com/local/` (ni a ninguno de los subdirectorios) dado que AIR vuelve a asignar la petición al directorio local de la aplicación. Se vuelven a asignar las peticiones independientemente si derivan de la navegación de una página, de XMLHttpRequest o de cualquier otro medio de carga de contenido.

Definición de una interfaz de puente de entorno limitado

Adobe AIR 1.0 y posterior

Puede utilizar un puente de entorno limitado cuando el contenido en el entorno limitado de la aplicación debe acceder a las propiedades o métodos definidos por el contenido en un entorno limitado que no pertenece a la aplicación o cuando el contenido que no pertenece a la aplicación debe acceder a las propiedades y métodos definidos por el contenido en el entorno limitado de la aplicación. Cree un puente con las propiedades `childSandboxBridge` y `parentSandboxBridge` del objeto `window` de cualquier documento secundario.

Definición de un puente de entorno limitado secundario

Adobe AIR 1.0 y posterior

La propiedad `childSandboxBridge` permite que el documento secundario exponga una interfaz al contenido en el documento principal. Para exponer una interfaz, se debe definir la propiedad `childSandbox` a una función u objeto en el documento secundario. Entonces puede acceder al objeto o función desde el contenido en el documento principal. En el siguiente ejemplo se muestra el modo en que un script ejecutándose en un documento secundario puede exponer al documento principal un objeto que contiene una función y una propiedad:

```
var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Si este contenido secundario se cargó en un iframe y se le asignó un ID de “elemento secundario”, se puede acceder a la interfaz desde el contenido principal leyendo la propiedad `childSandboxBridge` del fotograma:

```
var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

Definición de un puente de entorno limitado principal

Adobe AIR 1.0 y posterior

La propiedad `childSandboxBridge` permite que el documento secundario exponga una interfaz al contenido en el documento principal. Para exponer una interfaz, el documento principal define la propiedad `parentSandbox` del documento secundario a una función u objeto definido en el documento principal. Entonces puede acceder al objeto o función desde el contenido en el documento secundario. En el siguiente ejemplo se muestra el modo en que un script ejecutándose en un fotograma principal puede exponer al documento secundario un objeto que contiene una función:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

Utilizando esta interfaz, el contenido en un fotograma secundario puede guardar el texto en un archivo denominado `save.txt`, pero no tiene ningún otro acceso al sistema de archivos. El contenido secundario puede llamar a la función save de la siguiente manera:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

El contenido de aplicación debe exponer una interfaz lo más limitada posible a otros entornos limitados. Se debería considerar el contenido que no pertenece a la aplicación intrínsicamente sospechoso ya que puede estar sujeto a la inserción de código accidental o malintencionado. Se debe establecer una protección adecuada para evitar el mal uso de la interfaz que se expone a través del puente de entorno limitado principal.

Acceso de un puente de entorno limitado principal durante la carga de una página

Adobe AIR 1.0 y posterior

Para que un script en un documento secundario acceda a un puente de entorno limitado principal, se debe definir el puente antes de ejecutar el script. Los objetos Window, frame e iframe distribuyen un evento `dominitialize` cuando se crea un nuevo DOM de página, pero antes de que se haya analizado un script o se hayan añadido elementos DOM. Se puede utilizar el evento `dominitialize` para establecer el puente con suficiente anticipación en la secuencia de construcción de la página al que pueden acceder todos los scripts en documento secundario.

En el siguiente ejemplo se muestra la creación de un puente de entorno limitado principal en respuesta al evento `dominitialize` distribuido desde el fotograma secundario:

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

El siguiente documento `child.html` muestra cómo el contenido secundario puede acceder al puente de entorno limitado principal:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Para detectar el evento `dominitialize` en un objeto window secundario, en lugar de un objeto frame, se debe añadir el detector al nuevo objeto secundario window creado por la función `window.open()`:

```
var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

En este caso, no hay modo de asignar el contenido de aplicación en un entorno limitado que no pertenece a la aplicación. Esta técnica solo es útil cuando se carga `child.html` desde fuera del directorio de la aplicación. Aún se puede asignar el contenido de aplicación del objeto window a un entorno limitado que no pertenece a la aplicación, pero primero se debe cargar una página intermedia que utiliza fotogramas para cargar el documento secundario y asignarla al entorno limitado deseado.

Si se utiliza la función `createRootWindow()` de la clase HTMLLoader para crear una ventana, la nueva ventana no es un elemento secundario del documento desde donde se llama `createRootWindow()`. Por consiguiente, no se puede crear un puente de entorno limitado desde la ventana llamada al contenido que no pertenece a la aplicación cargado en la nueva ventana. En cambio, se debe cargar una página intermedia en la nueva ventana que utiliza fotogramas para cargar el documento secundario. Entonces se puede establecer el puente desde el documento principal de la nueva ventana al documento secundario cargado en el fotograma.

Capítulo 60: Uso de scripts en el contenedor HTML de AIR

Adobe AIR 1.0 y posterior

La clase HTMLLoader actúa como el contenedor para el contenido HTML en Adobe® AIR®. La clase proporciona muchas propiedades y métodos, heredados de la clase Sprite, para controlar el comportamiento y la apariencia del objeto en la lista de visualización de ActionScript™ 3.0. Además, la clase define la propiedades y los métodos para dichas tareas como cargar e interactuar con el contenido HTML y la gestión del historial.

La clase HTMLHost define un conjunto de comportamientos predeterminados para un objeto HTMLLoader. Cuando se crea un objeto HTMLLoader, no se proporciona ninguna implementación HTMLHost. Por consiguiente, cuando el contenido HTML activa uno de los comportamientos predeterminados, como cambiar la ubicación o el título de la ventana, no sucede nada. Puede extender la clase HTMLHost para definir los comportamientos deseados para la aplicación.

Se proporciona una implementación predeterminada de HTMLHost para las ventanas HTML creadas por AIR. Se puede asignar la implementación predeterminada HTMLHost a otro objeto HTMLLoader definiendo la propiedad `htmlHost` del objeto utilizando un nuevo objeto HTMLHost creado con el parámetro `defaultBehavior` definido en `true`.

Nota: *en la arquitectura de Adobe® Flex™, el objeto HTMLLoader está contenido por el componente mx:HTML. Cuando se utiliza Flex, se debe usar el componente HTML.*

Propiedades de visualización de objetos HTMLLoader

Adobe AIR 1.0 y posterior

Un objeto HTMLLoader hereda las propiedades de visualización de la clase Sprite de Adobe® Flash® Player. Por ejemplo, se puede cambiar el tamaño, mover, ocultar y cambiar el color de fondo. O bien se pueden aplicar efectos avanzados como filtros, máscaras, escala y rotación. Cuando se aplican los efectos, se debe tener en cuenta el impacto en la lectura. El contenido SWF y PDF cargado en una página HTML no se puede mostrar cuando se aplican ciertos efectos.

Las ventanas HTML tienen un objeto HTMLLoader que representa el contenido HTML. Este objeto está restringido dentro del área de la ventana, por lo que si se modifican las dimensiones, posición y rotación o el factor de escala no siempre se obtienen los resultados deseados.

Propiedades de visualización básicas

Adobe AIR 1.0 y posterior

Las propiedades de visualización básicas del HTMLLoader permiten colocar el control dentro del objeto de visualización principal, para establecer el tamaño y para mostrar u ocultar el control. No se deben cambiar estas propiedades para el objeto HTMLLoader de una ventana HTML.

Las propiedades básicas son:

Propiedad	Notas
`x`, `y`	Coloca el objeto dentro del contenedor principal.
`width`, `height`	Cambia las dimensiones del área de visualización.
`visible`	Controla la visibilidad del objeto y el contenido que tenga.

Fuera de una ventana HTML, las propiedades `width` y `height` de un objeto HTMLLoader tienen el valor predeterminado de 0. Se debe establecer la anchura y altura antes de que se pueda ver el contenido HTML cargado. El contenido HTML se dibuja acorde al tamaño de HTMLLoader, dispuesto según las propiedades HTML y CSS en el contenido. Si se cambia el tamaño de HTMLLoader se vuelve a fluir el contenido.

Cuando se carga el contenido en un nuevo objeto HTMLLoader (con `width` aún definido en 0), puede ser tentador definir las propiedades de visualización `width` y `height` del HTMLLoader usando las propiedades `contentWidth` y `contentHeight`. Esta técnica funciona para las páginas que tienen una anchura mínima razonable cuando se disponen según las reglas de flujo HTML y CSS. Sin embargo, algunas páginas fluyen con un diseño largo y angosto en la ausencia de una anchura razonable proporcionada por HTMLLoader.

Nota: *cuando se cambia la anchura y altura de un objeto HTMLLoader, los valores de scaleX y scaleY no cambian, como sucedería con la mayoría de los otros tipos de objetos de visualización.*

Transparencia del contenido de HTMLLoader

Adobe AIR 1.0 y posterior

La propiedad `paintsDefaultBackground` de un objeto HTMLLoader, que tiene el valor predeterminado de `true`, determina si el objeto HTMLLoader dibuja un fondo opaco. Cuando `paintsDefaultBackground` es `false`, el fondo es transparente. El contenedor del objeto de visualización u otros objetos de visualización debajo del objeto HTMLLoader están visibles detrás de los elementos en el primer plano del contenido HTML.

Si el elemento central o cualquier otro elemento del documento HTML especifica un color de fondo (usando por ejemplo `style="background-color:gray"`) entonces el fondo de esa parte del HTML es opaco y se representa con el color de fondo especificado. Si se define la propiedad `opaqueBackground` del objeto HTMLLoader, y `paintsDefaultBackground` es `false`, entonces el color definido para `opaqueBackground` es visible.

Nota: *se puede utilizar un gráfico transparente con formato PNG para proporcionar un fondo con mezcla con alfa para un elemento en un documento HTML. No se admite la configuración del estilo opaco para un elemento HTML.*

Ajuste de la escala del contenido HTMLLoader

Adobe AIR 1.0 y posterior

Se debe evitar ajustar la escala de un objeto HTMLLoader que exceda un factor de escala de 1,0. El texto en un contenido HTMLLoader se representa en una resolución específica y aparece pixelado si se aumenta el tamaño del objeto HTMLLoader. Para evitar que HTMLLoader, así como los contenidos, ajusten la escala cuando se cambia el tamaño de una ventana, se debe configurar la propiedad `scaleMode` del escenario en `StageScaleMode.NO_SCALE`.

Consideraciones al cargar el contenido SWF o PDF en una página HTML

Adobe AIR 1.0 y posterior

El contenido SWF y PDF cargado en un objeto HTMLLoader desaparece en las siguientes condiciones:

- Si se ajusta la escala del objeto HTMLLoader a un factor diferente de 1,0.
- Si se establece la propiedad alfa del objeto HTMLLoader a un valor diferente de 1,0.
- Si se rota el contenido HTMLLoader.

El contenido vuelve a aparecer si se quita el parámetro incorrecto de la propiedad y se quitan los filtros activos.

Además, el motor de ejecución no puede mostrar el contenido PDF en ventanas transparentes. El motor de ejecución solo muestra contenido SWF incorporado en una página HTML si el parámetro `wmode` del objeto o de la etiqueta incorporada se establece en `opaque` o `transparent`. Dado que el valor predeterminado de `wmode` es `window`, el contenido SWF no se visualiza en ventanas transparentes a no ser que se establezca el parámetro wmode de forma explícita.

Nota: *antes de AIR 1.5.2, el contenido SWF incorporado en HTML no se podía visualizar independientemente del valor usado en wmode.*

Para más información sobre la carga de estos tipos de medios en un HTMLLoader, consulte "Incorporación de contenido SWF en HTML" en la página 1032 y "Cómo añadir contenido PDF en AIR" en la página 575.

Propiedades de visualización avanzadas

Adobe AIR 1.0 y posterior

La clase HTMLLoader hereda varios métodos que se pueden usar para efectos especiales. En general, estos efectos tienen limitaciones cuando se utilizan con la visualización de HTMLLoader, pero pueden ser útiles para las transiciones u otros efectos temporales. Por ejemplo, si se muestra una ventana de diálogo para recopilar entradas del usuario, se puede desenfocar la visualización de la ventana principal hasta que el usuario cierre el diálogo. De igual modo, se puede desvanecer la visualización progresivamente cuando se cierra una ventana.

Las propiedades avanzadas de visualización son:

Propiedad	Limitaciones
`alpha`	Puede reducir la lectura del contenido HTML
`filters`	En una ventana HTML, los efectos exteriores están recortados por el borde de la ventana
`graphics`	Las formas dibujadas con los comandos gráficos aparecen debajo del contenido HTML, incluyendo el fondo predeterminado. La propiedad paintsDefaultBackground debe tener el valor false para que las formas dibujadas sean visibles.
`opaqueBackground`	No cambia el color del fondo predeterminado. La propiedad paintsDefaultBackground debe tener el valor false para que esta capa de color sea visible.

Propiedad	Limitaciones
`rotation`	Las esquinas del área HTMLLoader rectangular se pueden recortar por el borde de la ventana. El contenido SWF y PDF cargado en el contenido HTML no se muestra.
`scaleX`, `scaleY`	La visualización representada puede aparecer pixelada en factores de escala mayores que 1. El contenido SWF y PDF cargado en el contenido HTML no se muestra.
`transform`	Puede reducir la legibilidad del contenido HTML El borde de la ventana puede recortar la visualización HTML. El contenido SWF y PDF cargado en el contenido HTML no se muestra si la transformación involucra rotación, ajuste de escala o sesgado.

En el siguiente ejemplo se muestra la manera de establecer el conjunto de `filters` para desenfocar toda la visualización HTML:

```
 var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
html.width = 800;
html.height = 600;

var blur:BlurFilter = new BlurFilter(8);
var filters:Array = [blur];
html.filters = filters;
```

Desplazamiento de contenido HTML

Adobe AIR 1.0 y posterior

La clase HTMLLoader incluye las siguientes propiedades que permiten controlar el desplazamiento del contenido HTML:

Propiedad	Descripción
`contentHeight`	La altura, en píxeles, del contenido HTML.
contentWidth	La anchura, en píxeles, del contenido HTML.
scrollH	La posición de desplazamiento horizontal del contenido HTML en el objeto HTMLLoader.
scrollV	La posición de desplazamiento vertical del contenido HTML en el objeto HTMLLoader.

El siguiente código define la propiedad `scrollV` para que el contenido HTML se desplace a la parte inferior de la página:

```
 var html:HTMLLoader = new HTMLLoader();
html.addEventListener(Event.HTML_BOUNDS_CHANGE, scrollHTML);

const SIZE:Number = 600;
html.width = SIZE;
html.height = SIZE;

var urlReq:URLRequest = new URLRequest("http://www.adobe.com");
html.load(urlReq);
this.addChild(html);

function scrollHTML(event:Event):void
{
    html.scrollV = html.contentHeight - SIZE;
}
```

El HTMLLoader no incluye barras de desplazamiento horizontal y vertical. Se pueden implementar barras de desplazamiento en ActionScript o utilizando un componente Flex. El componente HTML de Flex automáticamente incluye barras de desplazamiento para el contenido HTML. Asimismo, se puede usar el método `HTMLLoader.createRootWindow()` para crear una ventana que contiene un objeto HTMLLoader con barras de desplazamiento (consulte "Creación de ventanas con contenido HTML de desplazamiento" en la página 1059).

Acceso a la lista del historial HTML

Adobe AIR 1.0 y posterior

A medida que se cargan nuevas páginas en un objeto HTMLLoader, el motor de ejecución mantiene una lista del historial del objeto. La lista del historial corresponde al objeto `window.history` en la página HTML. La clase HTMLLoader incluye las siguientes propiedades y métodos que permiten trabajar con la lista del historial HTML:

Miembro de clase	Descripción
`historyLength`	La longitud total de la lista del historial, incluyendo las entradas hacia atrás y hacia adelante.
`historyPosition`	La posición actual en la lista del historial. Los elementos del historial antes de esta posición representan la navegación hacia "atrás" y los elementos después de esta posición representan la navegación hacia "adelante".
`getHistoryAt()`	Devuelve el objeto URLRequest que corresponde a la entrada del historial en la posición específica en la lista del historial.
`historyBack()`	Navega hacia atrás en la lista del historial, si es posible.
`historyForward()`	Navega hacia adelante en la lista del historial, si es posible.
`historyGo()`	Navega la cantidad indicada de pasos en el historial del navegador. Navega hacia adelante si es positivo y hacia atrás si es negativo. La navegación a cero vuelve a cargar la página. La especificación de una posición más allá del final navega hasta el final de la lista.

Los elementos del historial se almacenan como objetos de tipo HTMLHistoryItem. La clase HTMLHistoryItem tiene las siguientes propiedades:

Propiedad	Descripción
`isPost`	Está definida en `true` si la página HTML incluye datos POST.
`originalUrl`	La URL original de la página HTML, antes de cualquier redirección.
`title`	El título de la página HTML.
`url`	La URL de la página HTML.

Configuración del agente de usuario que se utiliza al cargar contenido HTML

Adobe AIR 1.0 y posterior

La clase HTMLLoader tiene una propiedad `userAgent`, que le permite definir la cadena del agente de usuario que usa HTMLLoader. Se debe definir la propiedad `userAgent` del objeto HTMLLoader antes de llamar al método `load()`. Si define esta propiedad en la instancia de HTMLLoader, entonces la propiedad `userAgent` de URLRequest pasado al método `load()` *no* se utiliza.

Se puede definir la cadena del agente de usuario predeterminada usada por todos los objetos HTMLLoader en un dominio de aplicación definiendo la propiedad `URLRequestDefaults.userAgent`. Las propiedades estáticas URLRequestDefaults se aplican como valor predeterminado para todos los objetos URLRequest, no solo objetos URLRequest utilizados con el método `load()` de objetos HTMLLoader. La configuración de la propiedad `userAgent` de un HTMLLoader anula la configuración predeterminada `URLRequestDefaults.userAgent`.

Si no configura un valor de agente de usuario ya sea para la propiedad `userAgent` del objeto HTMLLoader o para `URLRequestDefaults.userAgent`, entonces se utiliza el valor predeterminado del agente de usuario de AIR. Este valor predeterminado varía según el sistema operativo basado en el motor de ejecución (como Mac OS o Windows), el lenguaje y la versión del motor de ejecución como en los siguientes dos ejemplos:

- `"Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"`
- `"Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"`

Configuración de la codificación de caracteres para utilizar con el contenido HTML

Adobe AIR 1.0 y posterior

Una página HTML puede especificar la codificación de caracteres que utiliza al incluir la etiqueta `meta`, como la siguiente:

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

Sustituya la configuración de la página para asegurar la utilización de una codificación de caracteres específica configurando la propiedad `textEncodingOverride` del objeto HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingOverride = "ISO-8859-1";
```

Especifique la codificación de caracteres que el contenido HTMLLoader utilizará cuando una página HTML no especifica una configuración con la propiedad `textEncodingFallback` del objeto HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingFallback = "ISO-8859-1";
```

La propiedad `textEncodingOverride` sustituye la configuración en la página HTML. Y la propiedad `textEncodingOverride` y la configuración en la página HTML sustituye la propiedad `textEncodingFallback`.

Configure la propiedad `textEncodingOverride` o la propiedad `textEncodingFallback` antes de cargar el contenido HTML.

Definición de interfaces de usuario del navegador para el contenido HTML

Adobe AIR 1.0 y posterior

JavaScript proporciona varias API para controlar la ventana que muestra el contenido HTML. En AIR, estas API se pueden anular implementando una clase HTMLHost personalizada.

Ampliación de la clase HTMLHost

Adobe AIR 1.0 y posterior

Si, por ejemplo, su aplicación presenta múltiples objetos HTMLLoader en una interfaz con fichas, tal vez desee que los cambios realizados por las páginas HTML cargadas cambien la etiqueta de la ficha, y no el título de la ventana principal. De igual manera, el código podría responder a una llamada `window.moveTo()` si se reposiciona el objeto HTMLLoader en el contenedor del objeto de visualización principal moviendo la ventana que contiene el objeto HTMLLoader, no haciendo nada o haciendo algo completamente diferente.

La clase HTMLHost de AIR controla las siguientes propiedades y métodos de JavaScript:

- `window.status`
- `window.document.title`
- `window.location`
- `window.blur()`
- `window.close()`
- `window.focus()`
- `window.moveBy()`
- `window.moveTo()`
- `window.open()`
- `window.resizeBy()`
- `window.resizeTo()`

Cuando se crea un objeto HTMLLoader usando `new HTMLLoader()`, las propiedades o métodos listados de JavaScript no se activan. La clase HTMLHost proporciona una implementación predeterminada de navegador de estas API de JavaScript. Asimismo se puede ampliar la clase HTMLHost para personalizar el comportamiento. Para crear un objeto HTMLHost que admite el comportamiento predeterminado, configure el parámetro `defaultBehaviors` en true en el constructor HTMLHost:

```
var defaultHost:HTMLHost = new HTMLHost(true);
```

Cuando se crea una ventana HTML en AIR con el método `createRootWindow()` de la clase HTMLLoader, se asigna inmediatamente una instancia HTMLHost que admite comportamientos predeterminados. Es posible cambiar el comportamiento del objeto host asignando una implementación HTMLHost diferente a la propiedad `htmlHost` de HTMLLoader o asignar `null` para desactivar las funciones completamente.

Nota: *AIR asigna un objeto HTMLHost predeterminado a la ventana inicial creada para una aplicación de AIR basada en HTML y cualquier ventana creada por la implementación predeterminada del método `window.open()` de JavaScript.*

Ejemplo: Ampliación de la clase HTMLHost

Adobe AIR 1.0 y posterior

En el siguiente ejemplo se muestra cómo personalizar la manera en que un objeto HTMLLoader afecta a la interfaz de usuario, ampliando la clase HTMLHost:

Ejemplo de Flex:

1. Cree una clase que amplía la clase HTMLHost (una subclase)
2. Anule los métodos de la nueva clase para gestionar cambios en los parámetros relacionados con la interfaz de usuario. Por ejemplo, la siguiente clase, CustomHost, define los comportamientos para las llamadas a `window.open()` y cambia a `window.document.title`. Las llamadas a `window.open()` abren la página HTML en una nueva ventana, y los cambios en `window.document.title` (incluyendo la configuración del elemento `<title>` de una página HTML) definen el título de dicha ventana.

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                         windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

3 En el código que contiene el HTMLLoader (no el código de la nueva subclase de HTMLHost), cree un objeto de la nueva clase. Asigne el nuevo objeto a la propiedad `htmlHost` de HTMLLoader. El siguiente código Flex usa la clase CustomHost definida en el paso anterior:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

Para probar el código descrito aquí, incluya un archivo HTML con el siguiente contenido en el directorio de la aplicación:

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

Ejemplo de Flash Professional:

1 Cree un archivo Flash para AIR. Defina la clase de documento a CustomHostExample y guarde el archivo como CustomHostExample.fla.

2 Cree un archivo ActionScript llamado CustomHost como que contiene una clase que amplía la clase HTMLHost (una subclase). Esta clase anula ciertos métodos de la nueva clase para gestionar cambios en los parámetros relacionados con la interfaz de usuario. Por ejemplo, la siguiente clase, CustomHost, define los comportamientos para las llamadas a `window.open()` y cambia a `window.document.title`. Las llamadas al método `window.open()` abren la página HTML en una nueva ventana, y los cambios en la propiedad `window.document.title` (incluyendo la configuración del elemento `<title>` de una página HTML) definen el título de dicha ventana.

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                        windowCreateOptions.y,
                                        windowCreateOptions.width,
                                        windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                    windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
```

```
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

3 Cree otro archivo ActionScript denominado CustomHostExample para que contenga la clase de documento para la aplicación. Esta clase crea un objeto HTMLLoader y define la propiedad del host a una instancia de la clase CustomHost definida en el paso anterior:

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

Para probar el código descrito aquí, incluya un archivo HTML con el siguiente contenido en el directorio de la aplicación:

```
<html>
    <head>
    <title>Test</title>
    <script>
    function openWindow()
    {
    document.title = "Test"
    window.open('Test.html');
    }
    </script>
    </head>
    <body bgColor="#EEEEEE">
    <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
    <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
    window.document.location = 'http://www.adobe.com'</a>
    <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
    <br/><a href="#" onclick="window.close()">window.close()</a>
    <br/><a href="#" onclick="window.blur()">window.blur()</a>
    <br/><a href="#" onclick="window.focus()">window.focus()</a>
    <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
    </body>
</html>
```

Gestión de cambios en la propiedad window.location

Adobe AIR 1.0 y posterior

Anule el método `locationChange()` para gestionar cambios de la URL de la página HTML. Se llama al método `locationChange()` cuando JavaScript en una página cambia el valor de `window.location`. En el siguiente ejemplo simplemente se carga la URL solicitada:

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

Nota: *puede utilizar la propiedad htmlLoader del objeto HTMLHost para hacer referencia al objeto HTMLLoader actual.*

Gestión de llamadas JavaScript a window.moveBy(), window.moveTo(), window.resizeTo(), window.resizeBy()

Adobe AIR 1.0 y posterior

Anule el método `set windowRect()` para gestionar cambios en los límites del contenido HTML. Se llama al método `set windowRect()` cuando JavaScript en una página llama a `window.moveBy()`, `window.moveTo()`, `window.resizeTo()` o `window.resizeBy()`. En el siguiente ejemplo simplemente se actualizan los límites de la ventana de escritorio:

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

Gestión de llamadas JavaScript a window.open()

Adobe AIR 1.0 y posterior

Anule el método `createWindow()` para gestionar llamadas JavaScript a `window.open()`. Las implementaciones del método `createWindow()` son responsables de la creación y devolución de un nuevo objeto HTMLLoader. Normalmente, se muestra HTMLLoader en una nueva ventana, pero no se requiere la creación de una ventana.

En el siguiente ejemplo se muestra cómo implementar la función `createWindow()` usando `HTMLLoader.createRootWindow()` para crear la ventana y el objeto HTMLLoader. Asimismo se puede crear un nuevo objeto NativeWindow de forma separada y añadir el HTMLLoader al escenario de la ventana.

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                 windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

Nota: *este ejemplo asigna la implementación HTMLHost personalizada a cualquier ventana nueva creada con `window.open()`. Asimismo, si se desea se puede usar una implementación diferente o definir la propiedad `htmlHost` en null para nuevas ventanas.*

El objeto transferido como parámetro al método `createWindow()` es un objeto HTMLWindowCreateOptions. La clase HTMLWindowCreateOptions incluye propiedades que informan sobre los valores definidos en la cadena de parámetro `features` en la llamada a `window.open()`:

Propiedad HTMLWindowCreateOptions	Parámetro correspondiente en la cadena de funciones en la llamada JavaScript a window.open()
`fullscreen`	`fullscreen`
`height`	`height`
`locationBarVisible`	`location`
`menuBarVisible`	`menubar`
`resizeable`	`resizable`
`scrollBarsVisible`	`scrollbars`
`statusBarVisible`	`status`
`toolBarVisible`	`toolbar`
`width`	`width`
`x`	`left` o `screenX`
`y`	`top` o `screenY`

La clase HTMLLoader no implementa todas las funciones que se pueden especificar en la cadena de función. La aplicación debe proporcionar barras de desplazamiento, barras de ubicación, barras de menús, barras de estado y barras de herramientas según corresponda.

Los otros argumentos al método `window.open()` JavaScript los gestiona el sistema. Una implementación `createWindow()` no debe cargar contenido en el objeto HTMLLoader ni definir el título de la ventana.

Gestión de llamadas JavaScript a window.close()

Adobe AIR 1.0 y posterior

Anule el método `windowClose()` para gestionar llamadas JavaScript a `window.close()`. En el siguiente ejemplo se cierra la ventana de escritorio cuando se llama al método `window.close()`:

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

Las llamadas JavaScript a `window.close()` no tienen que cerrar la ventana. Podría, por ejemplo, quitar el HTMLLoader de la lista de visualización, dejando la ventana abierta (que puede tener otro contenido), como en el siguiente código:

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

Gestión de cambios en la propiedad window.status

Adobe AIR 1.0 y posterior

Anule el método `updateStatus()` para gestionar los cambios de JavaScript al valor de `window.status`. En el siguiente ejemplo se rastrea el valor de estado:

```
 override public function updateStatus(status:String):void
{
    trace(status);
}
```

El estado requerido se pasa como una cadena al método `updateStatus()`.

El objeto HTMLLoader no proporciona una barra de estado.

Gestión de cambios en la propiedad document.title

Adobe AIR 1.0 y posterior

Anule el método `updateTitle()` para gestionar los cambios de JavaScript al valor de `window.document.title`. En el siguiente ejemplo se cambia el título de la ventana y se anexa la cadena, "Sample," al título:

```
 override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

Cuando se define `document.title` en una página HTML, el título solicitado se pasa como una cadena al método `updateTitle()`.

Los cambios en `document.title` no tienen que cambiar el título de la ventana que contiene el objeto HTMLLoader. Podría, por ejemplo, cambiar otro elemento de la interfaz como un campo de texto.

Gestión de llamadas JavaScript a window.blur() y window.focus()

Adobe AIR 1.0 y posterior

Anule los métodos `windowBlur()` y `windowFocus()` para gestionar las llamadas JavaScript a `window.blur()` y `window.focus()`, como se muestra en el siguiente ejemplo:

```
override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

Nota: *AIR no proporciona una API para desactivar una ventana o aplicación.*

Creación de ventanas con contenido HTML de desplazamiento

Adobe AIR 1.0 y posterior

La clase HTMLLoader incluye un método estático, `HTMLLoader.createRootWindow()`, que le permite abrir una nueva ventana(representada por un objeto NativeWindow) que contiene un objeto HTMLLoader y definir algunos parámetros de la interfaz de usuario para esa ventana. El método tiene cuatro parámetros, que permiten definir la interfaz de usuario:

Parámetro	Descripción
`visible`	Un valor Boolean que especifica si la ventana está inicialmente visible (`true`) o no (`false`).
`windowInitOptions`	Un objeto NativeWindowInitOptions. La clase NativeWindowInitOptions define las opciones de inicialización de un objeto NativeWindow, incluyendo: si se puede minimizar, maximizar o cambiar el tamaño de la ventana, si la ventana tiene fondo cromático del sistema o fondo cromático personalizado, si la ventana es transparente o no (para las ventanas que no usan fondo cromático) y el tipo de ventana.
`scrollBarsVisible`	Si hay barras de desplazamiento (`true`) o no (`false`).
`bounds`	Un objeto Rectangle que define la posición y el tamaño de la nueva ventana.

Por ejemplo, el siguiente código usa el método `HTMLLoader.createRootWindow()` para crear una ventana con contenido HTMLLoader que usa barras de desplazamiento.

```
var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
var bounds:Rectangle = new Rectangle(10, 10, 600, 400);
var html2:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2:URLRequest = new URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

Nota: *las ventanas creadas llamando a* `createRootWindow()` *directamente en JavaScript permanecen independientes de la ventana HTML abierta. Las propiedades de la ventana JavaScript* `opener` *y* `parent`*, por ejemplo, son* `null`*. Sin embargo, se llama a* `createRootWindow()` *indirectamente anulando el método* `createWindow()` *de HTMLHost para llamar a* `createRootWindow()`*, entonces las propiedades* `opener` *y* `parent` *sí hacen referencia a la ventana HTML abierta.*

Creación de subclases de la clase HTMLLoader

Adobe AIR 1.0 y posterior

Se puede crear una subclase de la clase HTMLLoader, para crear nuevos comportamientos. Por ejemplo, puede crear una subclase que define detectores de eventos predeterminados para los eventos HTMLLoader (como los eventos distribuidos cuando se representa HTML o cuando el usuario hace clic en un vínculo).

En el siguiente ejemplo se amplía la clase HTMLHost para proporcionar un comportamiento *normal* cuando se llama al método de JavaScript `window.open()`. Luego el ejemplo define una subclase de HTMLLoader que usa la clase de implementación HTMLHost personalizada:

```
package
{
       import flash.html.HTMLLoader;
    public class MyHTMLHost extends HTMLHost
       {
           public function MyHTMLHost()
           {
               super(false);
        }
        override public function createWindow(opts:HTMLWindowCreateOptions):void
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(opts.x, opts.y, opts.width, opts.height);
            var html:HTMLLoader = HTMLLoader.createRootWindow(true,
                                                    initOptions,
                                                    opts.scrollBarsVisible,
                                                    bounds);
            html.stage.nativeWindow.orderToFront();
            return html
       }
}
```

El siguiente ejemplo define una subclase de la clase HTMLLoader que asigna un objeto MyHTMLHost a la propiedad `htmlHost`:

```
package
{
       import flash.html.HTMLLoader;
    import MyHTMLHost;
    import HTMLLoader;
       public class MyHTML extends HTMLLoader
       {
           public function MyHTML()
           {
               super();
               htmlHost = new MyHTMLHost();
           }
       }
}
```

Para más información sobre la clase HTMLHost y el método `HTMLLoader.createRootWindow()` usado en este ejemplo, consulte "Definición de interfaces de usuario del navegador para el contenido HTML" en la página 1049.

Capítulo 61: Gestión de eventos relacionados con HTML en AIR

Adobe AIR 1.0 y posterior

Un sistema de gestión de eventos permite a los programadores responder a entradas del usuario y eventos del sistema de una forma conveniente. El modelo de eventos de Adobe® AIR® no solo es conveniente, sino que además cumple las normas pertinentes. Basado en la especificación de eventos del modelo de objetos de documento (DOM) de nivel 3, una arquitectura de gestión de eventos que es la norma del sector, el modelo de eventos ofrece una herramienta de gestión de eventos para programadores que es potente a la vez que intuitiva.

Eventos HTMLLoader

Adobe AIR 1.0 y posterior

Un objeto HTMLLoader distribuye los siguientes eventos de Adobe® ActionScript® 3.0:

Evento	Descripción
`htmlDOMInitialize`	Se distribuye al crearse el documento HTML, pero antes de que se analicen scripts o se añadan a la página nodos del DOM.
`complete`	Se distribuye cuando se ha creado el DOM de HTML como respuesta a una operación de cargar, inmediatamente después del evento `onload` en la página HTML.
`htmlBoundsChanged`	Se distribuye cuando ha cambiado al menos una de las propiedades `contentWidth` o `contentHeight`.
`locationChange`	Se distribuye cuando ha cambiado la propiedad de ubicación de HTMLLoader.
`locationChanging`	Se distribuye antes de que cambie la ubicación de HTMLLoader debido a la navegación del usuario, una llamada a JavaScript o un redireccionamiento. El evento `locationChanging` no se distribuye cuando se llama a los métodos `load()`, `loadString()`, `reload()`, `historyGo()`, `historyForward()` o `historyBack()`. Llamar al método `preventDefault()` del objeto de evento distribuido cancelará la navegación. Si se abre un vínculo en el navegador del sistema, no se distribuye un evento locationChanging, ya que la ubicación de HTMLLoader no cambia.
`scroll`	Se distribuye siempre que el motor de HTML cambia la posición de desplazamiento. Los eventos de desplazamiento pueden deberse a la navegación a vínculos de anclaje (vínculos `#`) en la página o a llamadas al método `window.scrollTo()`. Escribir texto en una zona de entrada de texto también puede dar lugar a un evento de desplazamiento.
`uncaughtScriptException`	Se distribuye cuando se produce una excepción de JavaScript en HTMLLoader y la excepción no se captura en código JavaScript.

También se puede registrar una función de ActionScript para un evento de JavaScript (por ejemplo, `onClick`). Para obtener más información, consulte "Gestión de eventos DOM con ActionScript" en la página 1063.

Gestión de eventos DOM con ActionScript

Adobe AIR 1.0 y posterior

Se pueden registrar funciones de ActionScript para responder a los eventos de JavaScript. Por ejemplo, tomemos el siguiente contenido HTML:

```
<html>
<body>
    <a href="#" id="testLink">Click me.</a>
</html>
```

Se puede registrar una función de ActionScript como controlador de cualquier evento de la página. En el siguiente ejemplo el código añade la función `clickHandler()` como detector del evento `onclick` del elemento `testLink` de la página HTML:

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event):void {
    html.window.document.getElementById("testLink").onclick = clickHandler;
}

function clickHandler( event:Object ):void {
    trace("Event of type: " + event.type );
}
```

El objeto de evento distribuido no es de tipo flash.events.Event ni una de las subclases Event. Utilice la clase Object para declarar un tipo para el argumento de función del controlador de eventos.

También se puede utilizar el método `addEventListener()` para registrarlo para estos eventos. Por ejemplo, se podría sustituir el método `completeHandler()` del ejemplo anterior con el código siguiente:

```
function completeHandler(event:Event):void {
    var testLink:Object = html.window.document.getElementById("testLink");
    testLink.addEventListener("click", clickHandler);
}
```

Cuando un detector hace referencia a un elemento DOM concreto, conviene esperar a que el objeto HTMLLoader principal distribuya el evento `complete` antes de añadir los detectores de eventos. Las páginas HTML cargan a menudo varios archivos y el DOM de HTML no se termina de crear hasta no haberse cargado y analizado todos los archivos. HTMLLoader distribuye el evento `complete` una vez creados todos los elementos.

Respuesta a excepciones en JavaScript sin capturar

Adobe AIR 1.0 y posterior

Tomemos el siguiente HTML:

```
<html>
<head>
    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

Contiene una función de JavaScript, `throwError()`, que hace referencia a una variable desconocida, `melbaToast`:

```
var x = 400 * melbaToast;
```

Si una operación de JavaScript se encuentra con una operación no válida que no es capturada en el código JavaScript con una estructura `try`/`catch`, el objeto HTMLLoader que contiene la página distribuye un evento HTMLUncaughtScriptExceptionEvent. Se puede registrar un controlador para este evento, como en el código siguiente:

```
var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.width = container.width;
html.height = container.height;
container.addChild(html);
html.addEventListener(HTMLUncaughtScriptExceptionEvent.UNCAUGHT_SCRIPT_EXCEPTION,
                          htmlErrorHandler);
function htmlErrorHandler(event:HTMLUncaughtJavaScriptExceptionEvent):void
{
    event.preventDefault();
    trace("exceptionValue:", event.exceptionValue)
    for (var i:int = 0; i < event.stackTrace.length; i++)
    {
        trace("sourceURL:", event.stackTrace[i].sourceURL);
        trace("line:", event.stackTrace[i].line);
        trace("function:", event.stackTrace[i].functionName);
    }
}
```

En JavaScript se puede controlar el mismo evento con la propiedad window.htmlLoader:

```
<html>
<head>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>

    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }

        function htmlErrorHandler(event) {
            event.preventDefault();
            var message = "exceptionValue:" + event.exceptionValue + "\n";
            for (var i = 0; i < event.stackTrace.length; i++){
                message += "sourceURL:" + event.stackTrace[i].sourceURL +"\n";
                message += "line:" + event.stackTrace[i].line +"\n";
                message += "function:" + event.stackTrace[i].functionName + "\n";
            }
            alert(message);
        }

        window.htmlLoader.addEventListener("uncaughtScriptException", htmlErrorHandler);
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

El controlador de eventos `htmlErrorHandler()` cancela el comportamiento predeterminado del evento (que es enviar el mensaje de error de JavaScript a la salida de la sentencia trace de AIR) y genera su propio mensaje de salida. Presenta el valor `exceptionValue` del objeto HTMLUncaughtScriptExceptionEvent. Presenta las propiedades de cada objeto del conjunto `stackTrace`:

```
exceptionValue: ReferenceError: Can't find variable: melbaToast
sourceURL: app:/test.html
line: 5
function: throwError
sourceURL: app:/test.html
line: 10
function: onclick
```

Gestión de eventos del motor de ejecución con JavaScript

Adobe AIR 1.0 y posterior

Las clases del motor de ejecución admiten añadir controladores de eventos con el método `addEventListener()`. Para añadir una función de controlador para un evento, llame al método `addEventListener()` del objeto que distribuye el evento, indicando el tipo de evento y la función de gestión. Por ejemplo, para detectar el evento `closing` distribuido cuando un usuario hace clic en el botón de cierre de la ventana situado en la barra del título, utilice la sentencia siguiente:

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

Creación de una función de controlador de eventos

Adobe AIR 1.0 y posterior

El código siguiente crea un archivo HTML sencillo que presenta información sobre la posición de la ventana principal. Una función de controlador denominada `moveHandler()` detecta si hay un evento de traslado (definido por la clase NativeWindowBoundsEvent) de la ventana principal.

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

Cuando el usuario desplaza la ventana, los elementos "textarea" muestran las posiciones X e Y actualizadas de la ventana:

Obsérvese que el objeto de evento se pasa como argumento al método `moveHandler()`. El parámetro de evento permite que la función de controlador examine el objeto de evento. En este ejemplo se utiliza la propiedad `type` del objeto de evento para notificar que se trata de un evento `move`.

Eliminación de detectores de eventos

Adobe AIR 1.0 y posterior

El método `removeEventListener()` sirve para eliminar un detector de eventos que ya no se necesita. Siempre conviene eliminar los detectores que no se vayan a usar más. Los parámetros obligatorios incluyen `eventName` y `listener`, los mismos que para el método `addEventListener()`.

Eliminación de detectores de eventos de páginas HTML que permiten la navegación

Adobe AIR 1.0 y posterior

Cuando el contenido HTML se desplaza, o se desecha porque se cierra la ventana que lo contiene, los detectores de eventos que hacen referencia a objetos en la página descargada no se eliminan automáticamente. Cuando un objeto distribuye un evento a un controlador que ya se ha descargado, aparece el siguiente mensaje de error: "The application attempted to reference a JavaScript object in an HTML page that is no longer loaded." (La aplicación intentó hacer referencia a un objeto JavaScript en una página HTML que ya no está cargada).

Para evitar este error, elimine los detectores de eventos de JavaScript que haya en una página HTML antes de que esta desaparezca. En el caso del desplazamiento por páginas (en un objeto HTMLLoader), elimine detector de eventos durante el evento `unload` del objeto `window`.

En el siguiente ejemplo, el código JavaScript elimina un detector de eventos para un evento `uncaughtScriptException`:

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Para evitar que se produzca el error al cerrar ventanas con contenido HTML, llame a una función "cleanup" (limpieza) como respuesta al evento `closing` del objeto NativeWindow (`window.nativeWindow`). En el siguiente ejemplo, el código JavaScript elimina un detector de eventos para un evento `uncaughtScriptException`:

```
 window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Este error se puede evitar eliminando un detector de eventos cuando se ejecute (si el evento solo se debe gestionar una vez). En el siguiente ejemplo, el código JavaScript crea una ventana html llamando al método `createRootWindow()` de la clase HTMLLoader y añade un detector de eventos para el evento `complete`. Cuando se llama al controlador de eventos `complete`, elimina su propio detector de eventos con la función `removeEventListener()`:

```
 var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

La eliminación de detectores de eventos que ya no se necesitan permite al recolector de datos innecesarios del sistema recuperar la memoria asociada con esos detectores.

Comprobación de detectores de eventos existentes

Adobe AIR 1.0 y posterior

El método `hasEventListener()` permite verificar si existe un detector de eventos en un objeto.

Capítulo 62: Visualización de contenido HTML en aplicaciones móviles

Adobe AIR 2.5 y posterior

La clase StageWebView muestra contenido HTML utilizando el control de navegador del sistema en los dispositivos móviles y el uso del control estándar HTMLLoader de Adobe® AIR® en los equipos de escritorio. Compruebe la propiedad `StageWebView.isSupported` para determinar si la clase se admite en el dispositivo actual. La compatibilidad no está garantizada en todos los dispositivos en el perfil móvil.

En todos los perfiles, la clase StageWebView solo admite la interacción limitada entre el contenido HTML y el resto de la aplicación. También se puede controlar la navegación, pero no se permite la reutilización de scripts ni el intercambio directo de datos. Es posible cargar contenido desde una URL local o remota o transmitir una cadena de HTML.

Objetos StageWebView

El objeto StageWebView no es un objeto de visualización y no se puede añadir a la lista de visualización. Funciona como ventana gráfica añadida directamente al escenario. El contenido de StageWebView se dibuja sobre la parte superior de cualquier contenido de la lista de visualización. No hay ninguna manera de controlar el orden de dibujo de diferentes objetos StageWebView.

Para mostrar un objeto StageWebView, se asigna el escenario en el que va a aparecer el objeto a la propiedad `stage` de StageWebView. Establezca el tamaño de la visualización utilizando la propiedad `viewPort`.

Establezca las coordenadas x e y de la propiedad `viewPort` entre -8192 y 8191. El valor máximo de la anchura y la altura del escenario es 8191. Si el tamaño sobrepasa los valores máximos, se emitirá una excepción.

En el siguiente ejemplo se crea un objeto StageWebView, se establecen las propiedades `stage` y `viewPort` y se muestra una cadena de HTML:

```
var webView:StageWebView = new StageWebView();
webView.viewPort = new Rectangle( 0, 0, this.stage.stageWidth, this .stage.stageHeight);
webView.stage = this.stage;
var htmlString:String = "<!DOCTYPE HTML>" +
                        "<html><body>" +
                        "<p>King Philip could order five good steaks.</p>" +
                        "</body></html>";
webView.loadString( htmlString );
```

Para ocultar un objeto StageWebView, establezca su propiedad `stage` en `null`. Para destruir todo el objeto, llame al método `dispose()`. La llamada a `dispose()` es opcional, pero ayuda al recolector de elementos no utilizados a recuperar antes la memoria utilizada por el objeto.

Contenido

Se puede cargar el contenido en un objeto StageWebView utilizando dos métodos: `loadURL()` y `loadString()`.

El método `loadURL()` carga un recurso en la URL especificada. Es posible utilizar cualquier esquema de URI admitido por el control de navegador web del sistema, incluyendo: data:, file:, http:, https: y javascript:. No se admiten los esquemas app: y app-storage:. AIR no valida la cadena URL.

El método `loadString()` carga una cadena literal que incluye contenido HTML. La ubicación de la página cargada con este método se expresa de la siguiente manera:

- En el escritorio: about:blank
- En iOS: *htmlString*
- En Android: el formato de URI de datos del *htmlString* codificado

El esquema de URI determina las reglas para cargar los datos o el contenido incorporado.

Esquema de URI	Carga de recursos locales	Carga de recursos remotos	XMLHttpRequest local	XMLHttpRequest remota
data:	No	Sí	No	No
file:	Sí	Sí	Sí	Sí
http:, https:	No	Sí	No	Mismo dominio
about: (loadString() method)	No	Sí	No	No

Nota: *si la propiedad `displayState` del escenario se establece en `FULL_SCREEN`, en Desktop, no se puede escribir en un campo de texto mostrado en StageWebView. No obstante, en iOS y Android, es posible escribir en un campo de texto en StageWebView aunque la propiedad `displayState` del escenario sea `FULL_SCREEN`.*

En el siguiente ejemplo se utiliza un objeto StageWebView para mostrar un sitio web de Adobe:

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );
        }
    }
}
```

En los dispositivos de Android, se debe especificar el permiso de INTERNET de Android para que la aplicación pueda cargar correctamente los recursos remotos.

En Android 3.0+, una aplicación debe activar la aceleración de hardware en el elemento manifestAdditions de Android del descriptor de la aplicación de AIR para mostrar contenido de plug-in en un objeto StageWebView. Consulte Activación de Flash Player y otros plug-ins en un objeto StageWebView.

URI de JavaScript

Se puede utilizar un URI de JavaScript para llamar a una función definida en la página HTML que se cargue mediante un objeto StageWebView. La función que se llama utilizando el URI de JavaScript se ejecuta en el contexto de la página web cargada. En el siguiente ejemplo se utiliza un objeto StageWebView para llamar a una función de JavaScript:

```
package {
    import flash.display.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    public class WebView extends Sprite
    {
        public var webView:StageWebView = new StageWebView();
        public function WebView()
        {
            var htmlString:String = "<!DOCTYPE HTML>" +
            "<html><script type=text/javascript>" +
            "function callURI(){" +
            "alert(\"You clicked me!!\");"+
            "}</script><body>" +
            "<p><a href=javascript:callURI()>Click Me</a></p>" +
            "</body></html>";
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadString( htmlString );
        }
    }
}
```

Más temas de ayuda

Sean Voisen: Making the Most of StageWebView

Eventos de navegación

Cuando un usuario hace clic en un vínculo en HTML, el objeto StageWebView distribuye un evento `locationChanging`. Se puede llamar al método `preventDefault()` del objeto de evento para detener la navegación. De lo contrario, el objeto StageWebView se desplaza a la nueva página y distribuye un evento `locationChange`. Una vez finalizada la carga de la página, StageWebView distribuye un evento `complete`.

Un evento `locationChanging` se distribuye en cada redirección HTML. Los eventos `locationChange` y `complete` se distribuyen en el momento adecuado.

En iOS, se distribuye un evento `locationChanging` antes de `locationChange`, excepto en el primer método `loadURL()` o `loadString()`. Un evento `locationChange` también se distribuye para los cambios de navegación mediante iFrames y Frames.

En el siguiente ejemplo se muestra cómo se puede evitar un cambio de ubicación y abrir la nueva página en el navegador del sistema.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.events.LocationChangeEvent;
    import flash.geom.Rectangle;
    import flash.net.navigateToURL;
    import flash.net.URLRequest;

    public class StageWebViewNavEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();

        public function StageWebViewNavEvents() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
        webView.addEventListener( LocationChangeEvent.LOCATION_CHANGING, onLocationChanging );
            webView.loadURL( "http://www.adobe.com" );
        }
        private function onLocationChanging( event:LocationChangeEvent ):void
        {
            event.preventDefault();
            navigateToURL( new URLRequest( event.location ) );
        }
    }
}
```

Historial

Conforme un usuario hace clic en los vínculos del contenido mostrado en un objeto StageWebView, el control guarda las pilas de historial de hacia atrás y hacia delante. El siguiente ejemplo muestra cómo desplazarse en las dos pilas del historial. En el ejemplo se utilizan las teclas programadas Back y Search.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample()
        {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );

            stage.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            if( event.keyCode == Keyboard.BACK && webView.isHistoryBackEnabled )
            {
                trace("back");
                webView.historyBack();
                event.preventDefault();
            }
            if( event.keyCode == Keyboard.SEARCH && webView.isHistoryForwardEnabled )
            {
                trace("forward");
                webView.historyForward();
            }
        }
    }
}
```

Selección

Aunque no es un objeto de visualización, la clase StageWebView contiene miembros que permiten administrar las transiciones seleccionadas dentro y fuera del control.

Cuando el objeto StageWebView se selecciona, distribuye un evento `focusIn`. Este evento se utiliza para administrar los elementos seleccionados en la aplicación, si es necesario.

Cuando StageWebView pierde la selección, distribuye un evento `focusOut`. Una instancia de StageWebView puede perder la selección si un usuario “pasa” el primer o el último control de la página web utilizando una bola de seguimiento de dispositivo o las flechas de dirección. La propiedad `direction` del objeto event permite saber si el flujo de selección sube y sobrepasa la parte superior de la página o desciende a la parte inferior. Utilice esta información para asignar la selección al objeto de visualización adecuado en la parte superior o inferior de StageWebView.

En iOS, la selección no se puede establecer mediante programación. StageWebView distribuye eventos `focusIn` y `focusOut` con la propiedad direction de `FocusEvent` establecida en `none`. Si el usuario toca dentro de StageWebView, se distribuye el evento `focusIn`. Si el usuario toca fuera de StageWebView, se distribuye el evento `focusOut`.

El siguiente ejemplo ilustra el modo en que la selección pasa del objeto StageWebView a los objetos de visualización de Flash:

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.FocusEvent;
    import flash.display.FocusDirection;
    import flash.events.LocationChangeEvent;

    public class StageWebViewFocusEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();
        var topControl:TextField = new TextField();
        var bottomControl:TextField = new TextField();

        public function StageWebViewFocusEvents()
        {
            trace("Starting");
            topControl.type = TextFieldType.INPUT;
            addChild( topControl );
            topControl.height = 60;
            topControl.width = stage.stageWidth;
            topControl.background = true;
            topControl.text = "One control on top.";
            topControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            topControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );

            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 60, stage.stageWidth, stage.stageHeight
- 120 );
            webView.addEventListener( FocusEvent.FOCUS_IN, webFocusIn );
            webView.addEventListener(FocusEvent.FOCUS_OUT, webFocusOut );
            webView.addEventListener(LocationChangeEvent.LOCATION_CHANGING,
                                     function( event:LocationChangeEvent ):void
                                     {
                                        event.preventDefault();
                                     } );
            webView.loadString("<form action='#'><input/><input/><input/></form>");
            webView.assignFocus();

            bottomControl.type = TextFieldType.INPUT;
            addChild( bottomControl );
            bottomControl.y = stage.stageHeight - 60;
            bottomControl.height = 60;
            bottomControl.width = stage.stageWidth;
            bottomControl.background = true;
            bottomControl.text = "One control on the bottom.";
            bottomControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            bottomControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );}

        private function webFocusIn( event:FocusEvent ):void
        {
```

```
            trace("Web focus in");
        }

        private function webFocusOut( event:FocusEvent ):void
        {
            trace("Web focus out: " + event.direction);
            if( event.direction == FocusDirection.TOP )
            {
                stage.focus = topControl;
            }
            else
            {
                stage.focus = bottomControl;
            }
        }

        private function flashFocusIn( event:FocusEvent ):void
        {
            trace("Flash focus in");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xff5566;
        }

        private function flashFocusOut( event:FocusEvent ):void
        {
            trace("Flash focus out");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xffffff;
        }

    }
}
```

Captura de mapas de bits

Un objeto StageWebView se representa sobre todo el contenido de la lista de visualización. No se puede añadir contenido sobre un objeto StageWebView. Por ejemplo, no se puede expandir un elemento desplegable sobre el contenido StageWebView. Para resolver este problema, capture una instantánea de StageWebView. A continuación, oculte StageWebView y añada la instantánea de mapa de bits.

El siguiente ejemplo muestra cómo capturar la instantánea de un objeto StageWebView utilizando el método `drawViewPortToBitmapData`. Oculta el objeto StageWebView estableciendo el escenario en null. Tras cargar por completo la página web, se llama a una función que captura el mapa de bits y lo muestra. Cuando se ejecuta, el código muestra dos etiquetas: Google y Facebook. Al hacer clic en la etiqueta, se captura la página web correspondiente y se muestra como instantánea en el escenario.

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    import flash.net.*;
    import flash.text.TextField;
    public class stagewebview extends Sprite
    {
        public var webView:StageWebView=new StageWebView();
        public var textGoogle:TextField=new TextField();
        public var textFacebook:TextField=new TextField();
        public function stagewebview()
        {
            textGoogle.htmlText="<b>Google</b>";
            textGoogle.x=300;
            textGoogle.y=-80;
            addChild(textGoogle);
            textFacebook.htmlText="<b>Facebook</b>";
            textFacebook.x=0;
            textFacebook.y=-80;
            addChild(textFacebook);
            textGoogle.addEventListener(MouseEvent.CLICK,goGoogle);
            textFacebook.addEventListener(MouseEvent.CLICK,goFaceBook);
            webView.stage = this.stage;
            webView.viewPort = new Rectangle(0, 0, stage.stageWidth, stage.stageHeight);
        }
    public function goGoogle(e:Event):void
        {
            webView.loadURL("http://www.google.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }

    public function goFaceBook(e:Event):void
        {
            webView.loadURL("http://www.facebook.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }
    public function handleLoad(e:Event):void
        {
            var bitmapData:BitmapData = new BitmapData(webView.viewPort.width,
webView.viewPort.height);
            webView.drawViewPortToBitmapData(bitmapData);
            var webViewBitmap:Bitmap=new Bitmap(bitmapData);
            addChild(webViewBitmap);
        }
    }
}
```

Capítulo 63: Uso de programas de trabajo para simultaneidad

Flash Player 11.4 y versiones posteriores, Adobe AIR 13.4 y versiones posteriores para plataformas de escritorio

Los programas de trabajo de ActionScript posibilitan la ejecución simultánea de código o, dicho de otro modo, permiten ejecutar código en segundo plano sin que esto interrumpa la ejecución del código principal.

Las API de simultaneidad de ActionScript están disponibles para plataformas de escritorio solo en Flash Player 11.4 y versiones posteriores, y en AIR 3.4 y versiones posteriores. La simultaneidad no se admite en AIR para plataformas móviles.

Descripción de los programas de trabajo y la simultaneidad

Flash Player 11.4 y versiones posteriores, Adobe AIR 13.4 y versiones posteriores para plataformas de escritorio

Cuando una aplicación no utiliza programas de trabajo, el código de la aplicación se ejecuta en único bloque lineal de pasos de ejecución denominado *subproceso* de ejecución. El subproceso ejecuta el código escrito por un desarrollador. También ejecuta gran parte del código que forma parte del motor de ejecución, sobre todo el código que actualiza la pantalla cuando cambian las propiedades de los objetos de visualización. Aunque el código se escribe en fragmentos como métodos y clases, en tiempo de ejecución el código ejecuta una línea cada vez, como si estuviese escrito en una única extensa serie de pasos. Considere este ejemplo hipotético de los pasos ejecutados por una aplicación:

1. Fotograma de entrada: el motor de ejecución llama a cualquier controlador de eventos `enterFrame` y ejecuta su código de uno en uno
2. Evento de ratón: el usuario mueve el ratón y el motor de ejecución llama a cualquier controlador de eventos mouse a medida que se suceden los distintos eventos rollover y rollout
3. Evento de carga finalizada: una solicitud de carga de un archivo xml desde una URL devuelve los datos del archivo cargado. Se llama al controlador de eventos y se ejecutan sus pasos, leyendo primero el contenido xml y creando después un conjunto de objetos a partir de los datos xml.
4. Evento de ratón: el ratón se vuelve a desplazar, por lo que el motor de ejecución llama a los controladores de eventos mouse correspondientes
5. Procesamiento: no se esperan más eventos, por lo que el motor de ejecución actualiza la pantalla en función de los cambios realizados en los objetos de visualización
6. Fotograma de entrada: el ciclo vuelve a comenzar

Tal y como se describe en el ejemplo, los pasos hipotéticos 1-5 se ejecutan secuencialmente dentro de un único bloque de tiempo denominado fotograma. Dado que se ejecutan uno después del otro en un único subproceso, el motor de ejecución no puede detener un paso del proceso para ejecutar otro. A una velocidad de 30 fotogramas por segundo, el motor de ejecución dispone de menos de 1/30 de segundo para ejecutar todas las operaciones. En muchos casos, este tiempo basta para ejecutar el código y el motor de ejecución simplemente espera durante el tiempo restante. No

obstante, supongamos que los datos xml cargados en el paso 3 son muy grandes y están muy anidados en la estructura xml. A medida que el código recorre una y otra vez el xml y crea objetos, puede tardar más de 1/30 de segundo en completar la operación. En este caso, los últimos pasos (respuesta del ratón y rediseño de la pantalla) no logran realizarse en el momento necesario. Esto hace que la pantalla se bloquee o que se produzcan fallos de visualización, ya que la pantalla no se rediseña lo suficientemente rápido como respuesta al desplazamiento del ratón del usuario.

Si todo el código se ejecuta en el mismo subproceso, solo hay una forma de evitar posibles retrasos y bloqueos de visualización: evitar operaciones de ejecución larga, como recorrer indefinidamente un conjunto extenso de datos. Los programas de trabajo de ActionScript ofrecen otra solución. Con un programa de trabajo, es posible ejecutar código de ejecución larga en un programa de trabajo independiente. Cada programa de trabajo se ejecuta en un subproceso distinto, con lo que el programa de trabajo en segundo plano es quien realiza la operación de ejecución larga en su propio subproceso. Esto libera el subproceso de ejecución del programa de trabajo principal y permite que redibuje la pantalla en cada fotograma sin interferir ni bloquear el resto del trabajo.

La capacidad de ejecutar múltiples operaciones de código al mismo tiempo de este modo se denomina *simultaneidad*. Cuando el programa de trabajo en segundo plano finaliza su trabajo, o en puntos de "progreso" durante el proceso, puede enviar notificaciones y datos al programa de trabajo principal. De este modo, se puede escribir código complejo o que consuma mucho tiempo, pero evitar la mala experiencia de usuario de ver cómo se bloquea la pantalla.

Los programas de trabajo son útiles porque reducen la posibilidad de pérdida de velocidad de datos debido a que otro código esté bloqueando el subproceso de procesamiento principal. No obstante, los programas de trabajo requieren memoria adicional del sistema y mayor uso de CPU, y esto puede afectar significativamente al rendimiento general de la aplicación. Dado que cada programa de trabajo utiliza su propia instancia de la máquina virtual del motor de ejecución, incluso la sobrecarga de un programa de trabajo trivial puede ser muy grande. Cuando utilice programas de trabajo, pruebe el código en todas las plataformas de destino para garantizar que no exigen demasiados recursos de uso del sistema. Adobe recomienda no utilizar más de uno o dos programas de trabajo en segundo plano en un escenario típico.

Creación y administración de programas de trabajo

Flash Player 11.4 y versiones posteriores, Adobe AIR 13.4 y versiones posteriores para plataformas de escritorio

El primer paso para utilizar un programa de trabajo para simultaneidad es crear un programa de trabajo en segundo plano. Se utilizan dos tipos de objetos para crear un programa de trabajo. El primero es una instancia de Worker, que es lo que se crea. El segundo es un objeto WorkerDomain, que crea el objeto Worker y gestiona los objetos Worker que se ejecutan en una aplicación.

Cuando se carga el motor de ejecución, crea automáticamente el objeto WorkerDomain. El motor de ejecución también crea automáticamente un programa de trabajo para el swf principal de la aplicación. Este primer programa de trabajo recibe el nombre de *programa de trabajo primordial*.

Dado que solo hay un objeto WorkerDomain por aplicación, se debe acceder a la instancia de WorkerDomain a través de la propiedad estática `WorkerDomain.current`.

En cualquier momento es posible acceder a la instancia actual de Worker (el programa de trabajo en el que se ejecuta el código) utilizando la propiedad estática `Worker.current`.

Creación de un objeto Worker a partir de un swf

Del mismo modo que el swf principal se ejecuta dentro del programa de trabajo primordial, un programa de trabajo en segundo plano ejecuta el código de un solo archivo swf. Para usar un programa de trabajo en segundo plano, debe crear y compilar el código del programa de trabajo como si fuera un archivo swf. Para crear el programa de trabajo en segundo plano, el programa de trabajo principal necesitará acceso a los bytes del archivo swf como un objeto ByteArray. Transfiera el objeto ByteArray al método `createWorker()` del objeto WorkerDomain para crear el programa de trabajo.

Se puede obtener el swf del programa de trabajo en segundo plano como un objeto ByteArray de tres formas distintas:

Incorporación del swf del programa de trabajo

Utilice la metaetiqueta [Embed] para incorporar el swf del programa de trabajo en el swf principal como un objeto ByteArray:

```
[Embed(source="../swfs/BgWorker.swf", mimeType="application/octet-stream")]
private static var BgWorker_ByteClass:Class;
private function createWorker():void
{
    var workerBytes:ByteArray = new BgWorker_ByteClass();
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

El swf del programa de trabajo se compila en el swf principal como una subclase de ByteArray denominada BgWorker_ByteClass. Crear una instancia de dicha clase le proporcionará un objeto ByteArray previamente lleno con los bytes del swf del programa de trabajo.

Carga de un swf de programa de trabajo externo

Utilice un objeto URLLoader para cargar un archivo swf externo. El archivo swf debe provenir del mismo dominio de seguridad (al igual que el archivo swf cargado debe provenir del mismo dominio de Internet) que el archivo swf principal o debe estar incluido en un paquete de aplicación de AIR.

```
var workerLoader:URLLoader = new URLLoader();
workerLoader.dataFormat = URLLoaderDataFormat.BINARY;
workerLoader.addEventListener(Event.COMPLETE, loadComplete);
workerLoader.load(new URLRequest("BgWorker.swf"));

private function loadComplete(event:Event):void
{
    // create the background worker
    var workerBytes:ByteArray = event.target.data as ByteArray;
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

Cuando el objeto URLLoader termina de cargar el archivo swf, los bytes del swf están disponibles en la propiedad `data` del objeto URLLoader (`event.target.data` en el ejemplo).

Uso del swf principal como swf del programa de trabajo

Puede utilizar un solo swf como swf principal y como swf del programa de trabajo. Utilice la propiedad `loaderInfo.bytes` de la clase principal de visualización para acceder a los bytes del swf.

```
// The primordial worker's main class constructor
public function PrimordialWorkerClass()
{
    init();
}

private function init():void
{
    var swfBytes:ByteArray = this.loaderInfo.bytes;

    // Check to see if this is the primordial worker or the background worker
    if (Worker.current.isPrimordial)
    {
        // create a background worker
        var bgWorker:Worker = WorkerDomain.current.createWorker(swfBytes);

        // ... set up worker communication and start the worker
    }
    else // entry point for the background worker
    {
        // set up communication between workers using getSharedProperty()
        // ... (not shown)

        // start the background work
    }
}
```

Si recurre a esta técnica, use una declaración `if` para dividir el código del archivo swf dentro del constructor de la clase principal o en un método al que llame. Para determinar si el código se está ejecutando en el programa de trabajo principal o en el programa de trabajo en segundo plano, compruebe la propiedad `isPrimordial` del objeto Worker actual, tal y como se muestra en el ejemplo.

Inicio de la ejecución de un programa de trabajo

Una vez creado un programa de trabajo, debe iniciar la ejecución del código mediante una llamada al método `start()` del objeto Worker. La operación de `start()` no se produce de forma inmediata. Para saber cuándo se está ejecutando el programa de trabajo, registre un detector para el evento `workerState` del objeto Worker. Este evento se distribuye cuando el objeto Worker cambia de estado en su ciclo de vida, por ejemplo, cuando inicia la ejecución del código. En el controlador de eventos `workerState`, compruebe que la propiedad `state` del objeto Worker es `WorkerState.RUNNING`. En dicho punto, el programa de trabajo se está ejecutando y el constructor de su clase principal se ha ejecutado. El siguiente código muestra un ejemplo del registro de un evento `workerState` y de una llamada al método `start()`:

```
// listen for worker state changes to know when the worker is running
bgWorker.addEventListener(Event.WORKER_STATE, workerStateHandler);
// set up communication between workers using
// setSharedProperty(), createMessageChannel(), etc.
// ... (not shown)
bgWorker.start();
private function workerStateHandler(event:Event):void
{
    if (bgWorker.state == WorkerState.RUNNING)
    {
        // The worker is running.
        // Send it a message or wait for a response.
    }
}
```

Gestión de la ejecución del programa de trabajo

Puede acceder al conjunto de programas de trabajo en ejecución de la aplicación en cualquier momento a través del método `listWorkers()` de la clase WorkerDomain. Este método devuelve el conjunto de programas de trabajo cuya propiedad `state` es `WorkerState.RUNNING`, incluido el programa de trabajo primordial. Si un programa de trabajo no se ha iniciado o ya se ha detenido su ejecución, no se incluirá.

Si ya no necesita un programa de trabajo, puede llamar al método `terminate()` del objeto Worker para cerrar el programa de trabajo y liberar su memoria, así como otros recursos del sistema.

Comunicación entre programas de trabajo

Flash Player 11.4 y versiones posteriores, Adobe AIR 13.4 y versiones posteriores para plataformas de escritorio

Aunque los programas de trabajo ejecutan su código en subprocesos de ejecución distintos, no supondrían ninguna ventaja si estuvieran totalmente aislados el uno del otro. La comunicación entre programas de trabajo en última instancia implica transferir datos entre programas de trabajo. Existen tres mecanismos principales para obtener datos entre programas de trabajo.

Al decidir qué técnicas de compartición de datos es la adecuada para una necesidad concreta de transferencia de datos, tenga en cuenta los dos importantes puntos que las diferencia. Una diferencia entre ambas reside en si hay un evento para notificar al receptor la disponibilidad de nuevos datos o si el programa de trabajo receptor debe buscar actualizaciones. Otra diferencia entre estas técnicas de compartición de datos está relacionada con el modo en que los datos se transfieren realmente. En algunos casos, el programa de trabajo receptor obtiene una copia de los datos compartidos, lo que significa que se han tenido que crear más objetos y se ha utilizado más memoria y ciclos de CPU. En otros casos, los programas de trabajo acceden a objetos que hacen referencia a la misma memoria subyacente del sistema, lo que implica una menor creación de objetos y menos uso de memoria en general. Estas diferencias se resumen a continuación:

Técnica de comunicación	Distribuye eventos al recibir datos	Comparte memoria con varios programas de trabajo
Propiedades compartidas del programa de trabajo	No	No, los objetos son copias, no referencias
MessageChannel	Sí	No, los objetos son copias, no referencias
ByteArray compartible	No	Sí, la memoria se comparte

Transferencia de datos con una propiedad compartida

La forma más básica de compartir datos entre programas de trabajo es usar una propiedad compartida. Cada programa de trabajo mantiene un diccionario interno con valores de propiedades compartidas. Las propiedades se almacenan con nombres de clave de cadena para distinguirlos de las propiedades. Para almacenar un objeto en un programa de trabajo como una propiedad compartida, llame al método `setSharedProperty()` del objeto Worker con dos argumentos, el nombre de la clave y el valor que se quiere almacenar:

```
// code running in the parent worker
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

Una vez definida la propiedad compartida, el valor se puede leer llamando al método `getSharedProperty()` del objeto Worker y transfiriendo el nombre de la clave:

```
// code running in the background worker
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

No hay restricciones respecto a qué programa de trabajo lee o establece el valor de la propiedad. Por ejemplo, el código de un programa de trabajo en segundo plano puede llamar a su método `setSharedProperty()` para almacenar un valor. El código que se ejecuta en el programa de trabajo principal puede utilizar a continuación el método `getSharedProperty()` para recibir los datos.

El valor transferido al método `setSharedProperty()` puede ser prácticamente cualquier tipo de objeto. Cuando se llama al método `getSharedProperty()`, el objeto que se devuelve es una copia del objeto transferido al método `setSharedProperty()` y no una referencia al mismo objeto, salvo en algunos casos excepcionales. La información específica sobre el modo en que se comparten los datos se detalla en "Referencias compartidas y valores copiados" en la página 1083.

La mayor ventaja de usar una propiedad compartida para transferir datos entre programas de trabajo es que está disponible incluso antes de que el programa de trabajo empiece a ejecutarse. Puede llamar al método `setSharedProperty()` de un objeto Worker en segundo plano para establecer una propiedad incluso antes de que el programa de trabajo se esté ejecutando. Cuando el programa de trabajo principal llama al método `start()` del objeto Worker, el motor de ejecución llama al constructor de la clase principal del programa de trabajo secundario. Cualquier propiedad compartida establecida antes de llamar a `start()` estará disponible para que el código del programa de trabajo secundario pueda leerla.

Transferencia de datos con un objeto MessageChannel

Un canal de mensajes proporciona un vínculo unidireccional de transferencia de datos entre dos programas de trabajo. Usar un objeto MessageChannel para transferir datos entre programas de trabajo tiene una ventaja clave. Cuando se envía un mensaje (un objeto) a través de un canal de mensajes, el objeto MessageChannel distribuye un evento `channelMessage`. El código del programa de trabajo receptor puede detectar un evento para saber cuándo están disponibles los datos. De este modo, el programa de trabajo receptor no necesita estar comprobando actualizaciones de datos continuamente.

Un canal de mensajes está asociado solamente a dos programas de trabajo: uno receptor y uno emisor. Para crear un objeto MessageChannel, llame al método `createMessageChannel()` del objeto Worker y transfiera el programa de trabajo receptor como un argumento:

```
// In the sending worker swf
var sendChannel:MessageChannel;
sendChannel = Worker.current.createMessageChannel(receivingWorker);
```

Ambos programas de trabajo necesitan tener acceso al objeto MessageChannel. El modo más sencillo de lograrlo es transferir el objeto MessageChannel mediante el método `setSharedProperty()`:

```
receivingWorker.setSharedProperty("incomingChannel", sendChannel);
```

En el programa de trabajo receptor, registre un detector para el evento `channelMessage` del objeto MessageChannel. El evento se distribuye cuando el programa de trabajo emisor envía datos a través del canal de mensajes.

```
// In the receiving worker swf
var incomingChannel:MessageChannel;
incomingChannel = Worker.current.getSharedProperty("incomingChannel");
incomingChannel.addEventListener(Event.CHANNEL_MESSAGE, handleIncomingMessage);
```

Para enviar los datos, en el programa de trabajo emisor llame al método `send()` del objeto MessageChannel:

```
// In the sending worker swf
sendChannel.send("This is a message");
```

En el programa de trabajo receptor, el objeto MessageChannel llama al controlador de eventos `channelMessage`. El programa de trabajo receptor entonces puede obtener datos mediante una llamada al método `receive()` del objeto MessageChannel.

```
private function handleIncomingMessage(event:Event):void
{
    var message:String = incomingChannel.receive() as String;
}
```

El objeto devuelto por el método receptor tiene el mismo tipo de datos que el objeto transferido al método `send()`. El objeto recibido es una copia del objeto transferido por el emisor, y no una referencia al objeto en el programa de trabajo emisor a no ser que sea uno de los pocos tipos de datos excepcionales que se describen en "Referencias compartidas y valores copiados" en la página 1083.

Compartir datos con un objeto ByteArray compartible

Cuando un objeto se transfiere entre dos programas de trabajo, el programa de trabajo receptor obtiene un nuevo objeto que es una copia del original. Los dos objetos se almacenan en distintas ubicaciones de la memoria del sistema. En consecuencia, cada copia del objeto que se recibe aumenta el uso total de memoria utilizada por el motor de ejecución. Además, cualquier cambio realizado en un objeto del programa de trabajo no afectará a la copia en el otro programa de trabajo. Para obtener más información sobre el modo en que se copian los datos, consulte "Referencias compartidas y valores copiados" en la página 1083.

De forma predeterminada, un objeto ByteArray se comporta de forma similar. Si transfiere una instancia de ByteArray a un método `setSharedProperty()` de un objeto Worker o un método `send()` de un objeto MessageChannel, el motor de ejecución crea un nuevo objeto ByteArray en la memoria del equipo y el programa de trabajo receptor obtiene una instancia de ByteArray que es una referencia a dicho nuevo objeto ByteArray. No obstante, puede cambiar este comportamiento del objeto ByteArray si establece su propiedad `shareable` en `true`.

Cuando un objeto ByteArray compartible se transfiere de un programa de trabajo a otro, la instancia de ByteArray del programa de trabajo es una referencia a la misma memoria subyacente del sistema operativo que utiliza la instancia de ByteArray del programa de trabajo emisor. Cuando el código de un programa de trabajo cambia el contenido del conjunto de bytes, dichos cambios están disponibles inmediatamente en los demás programas de trabajo que tengan acceso a dicho conjunto de bytes compartido.

Como los programas de trabajo ejecutan su código de forma simultánea, es posible que dos programas de trabajo intenten acceder al mismo tiempo a los mismos bytes de un conjunto de bytes. Esto podría provocar la pérdida o deterioro de los datos. Existen distintas API que puede utilizar para gestionar el acceso a recursos compartidos y evitar así estos problemas.

La clase ByteArray tiene métodos que permiten validar y modificar el contenido del conjunto de bytes en una única operación:

- Método atomicCompareAndSwapIntAt()
- Método atomicCompareAndSwapLength()

Además, el paquete flash.concurrent contiene clases que proporcionan control de acceso al trabajo con recursos compartidos:

- Clase Mutex
- Clase Condition

Referencias compartidas y valores copiados

En un caso normal, cuando se llama a los métodos `Worker.setSharedProperty()` o `MessageChannel.send()`, el objeto que se transfiere al programa de trabajo receptor se transfiere mediante su serialización en formato AMF. Esto tiene algunas consecuencias:

- El objeto creado en el programa de trabajo receptor cuando se llama al método `getSharedProperty()` se deserializa desde los bytes de AMF. Es una copia del objeto original, no una referencia al objeto. Cualquier cambio realizado en el objeto en cualquier programa de trabajo no se aplica a la copia en el otro programa de trabajo.
- Los objetos que no se pueden serializar en formato AMF (por ejemplo, objetos de visualización) no se pueden transferir a un programa de trabajo con `Worker.setSharedProperty()` o `MessageChannel.send()`.
- Para que una clase personalizada se pueda deserializar correctamente, la definición de la clase debe registrarse con la función `flash.net.registerClassAlias()` o los metadatos `[RemoteClass]`. Se debe utilizar el mismo alias para las dos versiones del programa de trabajo de la clase.

Existen cinco casos especiales de objetos que se comparten realmente, y que no se copian entre los programas de trabajo:

- Objetos Worker
- Objetos MessageChannel
- conjunto de bytes compartible (un objeto ByteArray cuya propiedad `shareable` es `true`)
- Objetos Mutex
- Objetos Condition

Cuando se transfiere una instancia de uno de estos objetos con los métodos `Worker.setSharedProperty()` o `MessageChannel.send()`, cada programa de trabajo tiene una referencia al mismo objeto subyacente. Los cambios realizados en la instancia de un programa de trabajo estarán disponibles inmediatamente en los demás programas de trabajo. Además, si transfiere la misma instancia de estos objetos más de una vez a un programa de trabajo, el motor de ejecución no creará una copia nueva del objeto en el programa de trabajo receptor. En su lugar, se reutilizará la misma referencia.

Técnicas adicionales de compartición de datos

Además de los mecanismos específicos de programas de trabajo para transferir datos, los programas de trabajo también pueden intercambiar datos mediante cualquiera de las API existentes que admiten compartición de datos entre dos aplicaciones swf, por ejemplo:

- objetos compartidos locales
- escribir datos en un archivo de un programa de trabajo y leer desde el archivo de otro programa de trabajo
- almacenar datos y leerlos en una base de datos de SQLite

Cuando se comparte un recurso entre dos o más programas de trabajo, generalmente es necesario evitar tener varios programas de trabajo que puedan acceder al recurso al mismo tiempo. Por ejemplo, si varios programas de trabajo acceden a un archivo del sistema de archivos local, podría causar pérdidas de datos o dañarlos, y podría no ser compatible con el sistema operativo.

Para evitar problemas de acceso simultáneo, utilice las clases Mutex y Condition del paquete flash.concurrent para proporcionar control de acceso al trabajo con recursos compartidos.

Al contrario de lo que sucede en otros mecanismos de compartición de datos, el motor de bases de datos de SQLite está diseñado para accesos simultáneos y tiene su propio soporte de transacciones integrado. Varios programas de trabajo pueden acceder a una base de datos de SQLite sin riesgo de dañar los datos. Dado que los programas de trabajo utilizan distintas instancias de SQLConnection, cada programa de trabajo accede a la base de datos en una transacción diferente. Las operaciones simultáneas de manipulación de datos no afectan a la integridad de los datos.

Véase también

"Trabajo con bases de datos SQL locales en AIR" en la página 744
Paquete flash.concurrent

Capítulo 64: Seguridad

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La seguridad es una de las principales preocupaciones de Adobe, los usuarios, los propietarios de sitios web y los desarrolladores de contenido. Por este motivo, Adobe® Flash® Player y Adobe® AIR™ incluyen un conjunto de reglas de seguridad y controles para proteger al usuario, al propietario del sitio web y al desarrollador de contenido. En este tema se analiza el modelo de seguridad para los archivos SWF publicados con ActionScript 3.0 y que se ejecutan en Flash Player 9.0.124.0 o posterior y los archivos SWF, HTML y JavaScript a no ser que se indique lo contrario.

El objetivo de este capítulo es proporcionar información general sobre la seguridad, por lo que no se intentará explicar de forma exhaustiva todos los detalles de implementación, escenarios de uso o ramificaciones para utilizar determinadas API. Para ver una descripción detallada de los conceptos de seguridad de Flash Player, consulte el tema de seguridad del Centro de desarrollo de Flash Player en la dirección www.adobe.com/go/devnet_security_es.

Información general sobre la seguridad de la plataforma Flash

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Gran parte del modelo de seguridad utilizado por los motores de ejecución de Flash Player y AIR se basa en el dominio de origen para los archivos SWF cargados, HTML, medios y otros recursos. El código ejecutable de un archivo de un dominio concreto de Internet, tal como www.ejemplo.com, podrá tener siempre acceso a todos los datos de dicho dominio. Estos activos se incluyen en el mismo grupo de seguridad, conocido como *entorno limitado de seguridad*. (Para obtener más información, consulte "Entornos limitados de seguridad" en la página 1087.)

Por ejemplo, el código ActionScript de un archivo SWF puede cargar archivos SWF, mapas de bits, audio, archivos de texto y otros activos de su propio dominio. Asimismo, la reutilización de scripts siempre está permitida entre dos archivos del mismo dominio, siempre que ambos archivos se hayan escrito utilizando ActionScript 3.0. La *reutilización de scripts* es la capacidad de un archivo para utilizar ActionScript para acceder a las propiedades, los métodos y los objetos definidos por el código en otro archivo.

La reutilización de scripts no es posible entre los archivos SWF escritos en ActionScript 3.0 y los escritos en versiones anteriores de ActionScript; sin embargo, estos archivos pueden comunicarse a través de la clase LocalConnection. Por otro lado, está prohibido de forma predeterminada que un archivo SWF reutilice los scripts de los archivos SWF de ActionScript 3.0 pertenecientes a otros dominios, así como que cargue datos de otros dominios. Sin embargo, este permiso se puede conceder si se llama al método `Security.allowDomain()` del archivo SWF cargado. Para obtener más información, consulte "Reutilización de scripts" en la página 1108.

Las reglas de seguridad básicas que siempre se aplican de forma predeterminada son:

- Los recursos del mismo entorno limitado de seguridad tienen acceso libre entre ellos.
- El código ejecutable de los archivos de un entorno limitado remoto no tienen nunca acceso a archivos y datos locales.

Los motores de ejecución de Flash Player y AIR consideran los siguientes componentes como dominios individuales y configuran entornos limitados de seguridad independientes para cada uno de ellos:

- `http://example.com`

- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

Incluso si un dominio con nombre, como http://example.com, corresponde a una dirección IP específica, como http://192.0.34.166, el motor de ejecución configura entornos limitados de seguridad independientes para cada uno de ellos.

Hay dos métodos básicos que puede utilizar un desarrollador para conceder a un archivo SWF el acceso a los activos de entornos limitados ajenos al del archivo SWF:

- El método `Security.allowDomain()` (consulte "Controles de autor (desarrollador)" en la página 1099)
- El archivo de política URL (consulte "Controles de sitio web (archivos de política)" en la página 1095)

En los modelos de seguridad del motor de ejecución de AIR y Flash Player, existe una diferencia entre cargar contenido y extraer o acceder a datos. El *contenido* se define como medios, incluidos los medios visuales que los motores de ejecución pueden mostrar, audio, vídeo o un archivo SWF que incluye medios mostrados. Se define como *datos* aquello que solo es accesible para el código El contenido y los datos se cargan de modos diferentes.

- Carga de contenido: el contenido se puede cargar utilizando clases como Loader, Sound y NetStream, mediante etiquetas MXML al utilizar Flex o con etiquetas HTML en una aplicación de AIR.
- Extracción de datos: los datos se pueden extraer de contenido de medios cargados a través de objetos Bitmap, los métodos `BitmapData.draw()` y `BitmapData.drawWithQuality()`, la propiedad `Sound.id3` o el método `SoundMixer.computeSpectrum()`. El método `drawWithQuality` está disponible en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores.
- Acceso a datos: se puede acceder a datos directamente cargándolos desde un archivo externo (como un archivo XML) a través de clases como URLStream, URLLoader, FileReference, Socket y XMLSocket. AIR proporciona clases adicionales para cargar datos como, por ejemplo, FileStream, y XMLHttpRequest.

El modelo de seguridad de Flash Player define reglas distintas para cargar contenido y acceder a los datos. En general, hay menos restricciones para cargar contenido que para acceder a los datos.

En general, el contenido (archivos SWF, mapas de bits, archivos MP3 y vídeos) puede cargarse desde cualquier origen, pero si procede de un dominio ajeno al del contenido o al código de carga, se ubicará en un entorno limitado de seguridad independiente.

Hay algunos obstáculos para cargar contenido:

- De forma predeterminada, los archivos SWF locales (cargados desde una dirección fuera de la red como, por ejemplo, el disco duro de un usuario) se clasifican en el entorno limitado local con sistema de archivos. Estos archivos no pueden cargar contenido de la red. Para obtener más información, consulte "Entornos limitados locales" en la página 1088.
- Los servidores RTMP (Real-Time Messaging Protocol) pueden limitar el acceso al contenido. Para obtener más información, consulte "Contenido proporcionado a través de servidores RTMP" en la página 1108.

Si el medio cargado es una imagen, audio o vídeo, sus datos (ya sean píxeles o sonidos) serán accesibles para un archivo SWF que no pertenezca a su entorno limitado de seguridad solo si el dominio de dicho archivo SWF se haya incluido en un archivo de política en el dominio de origen del medio. Para obtener información más detallada, consulte "Acceso a medios cargados como datos" en la página 1111.

Otros tipos de datos cargados contienen texto o archivos XML, que se cargan con un objeto URLLoader. De nuevo en este caso, para acceder a los datos de otro entorno limitado de seguridad, deben concederse permisos mediante un archivo de política URL en el dominio de origen. Para obtener información más detallada, consulte "Uso de URLLoader y URLStream" en la página 1114.

Nota: *los archivos de política nunca se requieren para el código que se ejecuta en el entorno limitado de la aplicación de AIR para cargar datos o contenido remoto.*

Entornos limitados de seguridad

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los equipos cliente pueden obtener archivos individuales que contienen código, contenido y datos de una serie de orígenes como, por ejemplo, sitios web externos, un sistema de archivos local o una aplicación instalada de AIR. Los motores de ejecución de Flash Player y AIR asignan individualmente archivos de código y otros recursos, como objetos compartidos, mapas de bits, sonidos, vídeos y archivos de datos, a entornos limitados de seguridad, en función de su origen, cuando se cargan. En las siguientes secciones se describen las reglas que aplican los motores de ejecución y que controlan dónde puede acceder el código o el contenido que se ejecuta en un determinado entorno limitado.

Para obtener más información sobre la seguridad de Flash Player, consulte el tema de seguridad del Centro de desarrollo de Flash Player en www.adobe.com/go/devnet_security_es.

Entornos limitados remotos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los motores de ejecución de Flash Player y AIR clasifican los recursos (incluyendo archivos SWF) procedentes de Internet en entornos limitados independientes que se corresponden con su dominio de origen. Por ejemplo, los recursos cargados desde *example.com* se situarán en un entorno limitado de seguridad diferente que los recursos cargados desde *foo.org*. De forma predeterminada, estos archivos tienen autorización para acceder a cualquier recurso de su propio servidor. Se puede permitir que los archivos SWF accedan a datos adicionales desde otros dominios. Para ello, es necesario conceder permisos de autor y sitio web explícitos, como los archivos de política URL y el método `Security.allowDomain()`. Para obtener información más detallada, consulte "Controles de sitio web (archivos de política)" en la página 1095 y "Controles de autor (desarrollador)" en la página 1099.

Los archivos SWF remotos no pueden cargar archivos ni recursos locales.

Para obtener más información sobre la seguridad de Flash Player, consulte el tema de seguridad del Centro de desarrollo de Flash Player en www.adobe.com/go/devnet_security_es.

Entornos limitados locales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un *archivo local* es un archivo al que se hace referencia a través del protocolo `file:` o una ruta UNC (convención de nomenclatura universal). Los archivos SWF locales se ubican en uno de los cuatro entornos limitados locales siguientes:

- Entorno limitado local con sistema de archivos: por motivos de seguridad y de forma predeterminada, los motores de ejecución de Flash Player y AIR colocan todos los archivos locales en el entorno limitado local con sistema de archivos. Desde este entorno limitado, el código ejecutable puede leer archivos locales (por ejemplo, mediante la clase URLLoader), pero no pueden comunicarse con la red de ningún modo. De este modo, el usuario tiene la seguridad de que no se filtran los datos locales a la red ni se comparten de forma indebida.
- El entorno limitado local con acceso a la red: al compilar un archivo SWF, se puede especificar que tenga acceso a la red cuando se ejecute como un archivo local (consulte "Configuración del tipo de entorno limitado de los archivos SWF locales" en la página 1090). Estos archivos se colocan en el entorno limitado local con acceso a la red. Los archivos SWF que se asignan al entorno limitado local con acceso a la red pierden el acceso al sistema de archivos local. A cambio, se les permite acceder a los datos de la red. Sin embargo, sigue sin permitirse que un archivo SWF local con acceso a la red lea los datos derivados de la red a menos que se disponga de permisos para ello, a través de un archivo de política URL o una llamada al método `Security.allowDomain()`. Para conceder dicho permiso, un archivo de política URL debe conceder permiso a *todos* los dominios a través de `<allow-access-from domain="*"/>` o `Security.allowDomain("*")`. Para obtener más información, consulte "Controles de sitio web (archivos de política)" en la página 1095 y "Controles de autor (desarrollador)" en la página 1099.
- El entorno limitado local de confianza: los archivos SWF locales que los usuarios o programas de instalación registran como archivos de confianza se colocan en el entorno limitado local de confianza. Los administradores del sistema y los usuarios también pueden reasignar (mover) un archivo SWF local al entorno limitado local de confianza, o quitarlo de él, en función de consideraciones de seguridad (consulte "Controles de administrador" en la página 1092 y "Controles de usuario" en la página 1094). Los archivos SWF asignados al entorno limitado local de confianza pueden interactuar con el resto de archivos SWF y cargar datos desde cualquier lugar (remoto o local).
- El entorno limitado de la aplicación AIR: este entorno limitado presenta contenido que se instaló con la aplicación AIR en ejecución. De forma predeterminada, el código que se ejecuta en el entorno limitado de la aplicación AIR puede reutilizar los scripts de los archivos de cualquier dominio. No obstante, a los archivos que se encuentran fuera del entorno limitado de la aplicación AIR no se les permite reutilizar el código de scripts en el entorno limitado de la aplicación. De forma predeterminada,el código y el contenido del entorno limitado de la aplicación de AIR pueden cargar contenido y datos desde cualquier dominio.

Se prohíbe estrictamente la comunicación entre entornos limitados locales con acceso a la red y entornos limitados locales con sistema de archivos, y entre entornos limitados locales con sistema de archivos y entornos limitados remotos. Una aplicación que se ejecute en Flash Player o un usuario o administrador no pueden conceder permiso para permitir dicha comunicación.

La creación de scripts en cualquier sentido entre archivos HTML locales y archivos SWF locales (por ejemplo, mediante la clase ExternalInterface) requiere que los archivos HTML y SWF implicados estén en el entorno limitado local de confianza. El motivo de ello es que los modelos de seguridad local de los navegadores difieren del modelo de seguridad local de Flash Player.

Los archivos SWF incluidos en el entorno limitado local con acceso a la red no pueden cargar archivos SWF en el entorno limitado local con sistema de archivos. Los archivos SWF incluidos en el entorno limitado local con sistema de archivos no pueden cargar archivos SWF en el entorno limitado local con acceso a la red.

Entorno limitado de la aplicación de AIR

Adobe AIR 1.0 y posterior

El motor de ejecución de Adobe AIR añade un entorno limitado adicional, denominado entorno limitado de la *aplicación*, al modelo del entorno limitado de seguridad de Flash Player. Los archivos instalados como parte de una aplicación de AIR se cargan en el entorno limitado de la aplicación. Los demás archivos cargados por la aplicación presentan restricciones de seguridad que se corresponden con las especificadas por el modelo de seguridad normal de Flash Player.

Cuando se instala una aplicación, todos los archivos incluidos en un paquete de AIR se instalan en el equipo del usuario en un directorio de la aplicación. Los desarrolladores pueden hacer referencia a este directorio en el código mediante el esquema de URL `app:/` (consulte "Esquemas de URI" en la página 846). Todos los archivos del árbol de directorios de la aplicación se asignan al entorno limitado de la aplicación cuando se ejecuta la aplicación. El contenido en el entorno limitado de la aplicación cuenta con todos los privilegios que tiene disponible una aplicación de AIR, incluyendo la interacción con el sistema de archivos local.

Muchas aplicaciones de AIR solo usan estos archivos instalados localmente para ejecutar la aplicación. Sin embargo, las aplicaciones de AIR no están restringidas solo a los archivos dentro del directorio de la aplicación; pueden cargar cualquier tipo de archivo de cualquier origen. Esto incluye archivos locales del equipo del usuario así como archivos de orígenes externos disponibles, como los de una red local o en Internet. El tipo de archivo no tiene ningún impacto en las restricciones de seguridad; los archivos HTML cargados tienen los mismos privilegios de seguridad que los archivos SWF cargados desde el mismo origen.

El contenido en el entorno limitado de seguridad de la aplicación tiene acceso a las API de AIR pero los contenidos en otros entornos limitados no pueden utilizarlas. Por ejemplo, la propiedad `air.NativeApplication.nativeApplication.applicationDescriptor`, que devuelve los contenidos del archivo descriptor de la aplicación de la aplicación, está restringido al contenido en el entorno limitado de seguridad de la aplicación. Otro ejemplo de una API restringida es la clase FileStream, que contiene métodos para leer y escribir en el sistema de archivos local.

Las API de ActionScript que solo están disponibles para el contenido en el entorno limitado de seguridad de la aplicación se indican con el logotipo de AIR en *Referencia de ActionScript 3.0 para la plataforma de Adobe Flash*. El uso de estas API en otros entornos limitados hace que el motor de ejecución emita una excepción SecurityError.

Para el contenido HTML (en un objeto HTMLLoader) todas las API JavaScript de AIR (las que están disponibles mediante la propiedad `window.runtime` o mediante el objeto `air` cuando se usa el archivo AIRAliases.js) están disponibles para el contenido en el entorno limitado de seguridad de la aplicación. El contenido HTML en otro entorno limitado no tiene acceso a la propiedad `window.runtime`, por lo que este contenido no puede acceder a las API de AIR o Flash Player.

El contenido que se ejecuta en el entorno limitado de la aplicación de AIR cuenta con las siguientes restricciones adicionales:

- Para el contenido HTML en el entorno limitado de seguridad de la aplicación, hay limitaciones en el uso de API que pueden transformar dinámicamente cadenas en código ejecutable después de que se carga el código. Esta limitación es para evitar que la aplicación accidentalmente inserte (y ejecute) código desde orígenes que no pertenecen a la aplicación (como dominios de red potencialmente inseguros). Un ejemplo es el uso de la función `eval()`. Para más información, consulte "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130
- Para evitar posibles ataques de suplantación de identidad (phishing), las etiquetas `img` en el contenido HTML en objetos TextField de ActionScript se omiten en el contenido SWF en el entorno limitado de seguridad de la aplicación.

- El contenido en el entorno limitado de la aplicación no puede usar el protocolo `asfunction` en el contenido HTML en los campos de texto de ActionScript 2.0.
- El contenido SWF en el entorno limitado de la aplicación no puede usar la caché entre dominios, una función que se añadió a la actualización 3 de Flash Player 9. Esta función permite que Flash Player utilice la caché persistentemente en el contenido del componente de la plataforma de Adobe y vuelva a utilizarlo en el contenido SWF cargado bajo demanda (eliminando la necesidad de volver a cargar el contenido varias veces).

Restricciones para JavaScript dentro de AIR

Adobe AIR 1.0 y posterior

A diferencia del contenido en el entorno limitado de seguridad de la aplicación, el contenido JavaScript en un entorno limitado de seguridad que no pertenece a la aplicación *puede* llamar a la función `eval()` para que ejecute dinámicamente el código generado en cualquier momento. Sin embargo, existen restricciones en el código JavaScript que se ejecuta en AIR en un entorno limitado de seguridad ajeno a la aplicación. Se incluyen:

- El código JavaScript en un entorno limitado que no pertenece a la aplicación no tiene acceso al objeto `window.runtime` y como tal este código no puede ejecutar las API de AIR.
- Como valor predeterminado, el contenido en un entorno limitado de seguridad que no pertenece a la aplicación no puede usar llamadas XMLHttpRequest para cargar datos de otros dominios excepto del dominio que llama a la petición. Sin embargo, el código de aplicación puede conceder permiso al contenido que no pertenece a la aplicación configurando un atributo `allowCrossdomainXHR` en el fotograma o en el iframe. Para obtener más información, consulte "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130.
- Hay restricciones para llamar al método `window.open()` JavaScript. Para más información, consulte "Restricciones para llamar al método window.open() JavaScript" en la página 1133.
- El contenido HTML en los entornos limitados de seguridad remotos (red) solo puede cargar contenido CSS, `frame`, `iframe` e `img` de dominios remotos (de URL de red).
- El contenido HTML en los entornos limitados local con sistema de archivos, local con acceso a la red o local de confianza solo pueden cargar contenido CSS, `frame`, `iframe` e `img` de entornos limitados locales (no de URL de red o aplicaciones).

Para más información, consulte "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130

Configuración del tipo de entorno limitado de los archivos SWF locales

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un usuario final o el administrador de un equipo puede especificar que un archivo SWF local es de confianza y permitir que cargue datos de todos los dominios, tanto locales como de red. Esto se especifica en los directorios Global Flash Player Trust y User Flash Player Trust. Para obtener más información, consulte "Controles de administrador" en la página 1092 y "Controles de usuario" en la página 1094.

Para obtener más información sobre los entornos limitados locales, consulte "Entornos limitados locales" en la página 1088.

Adobe Flash Professional

Se puede configurar un archivo SWF para el entorno limitado local con sistema de archivos o para el entorno limitado local con acceso a la red estableciendo la configuración de publicación del documento en la herramienta de edición

---

Adobe Flex

Puede configurar un archivo SWF para el entorno limitado local con sistema de archivos o el entorno limitado local con acceso a la red definiendo el indicador `use-network` en el compilador Adobe Flex. Para obtener más información, consulte la sección sobre las opciones del compilador de aplicaciones en *Building and Deploying Adobe Flex 3 Applications* (Creación e implementación de aplicaciones Adobe Flex 3).

La propiedad Security.sandboxType

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El autor de un archivo SWF puede usar la propiedad `Security.sandboxType` estática de solo lectura para determinar el tipo de entorno limitado al que el motor de ejecución de AIR o Flash Player ha asignado el archivo SWF. La clase Security incluye constantes que representan los valores posibles de la propiedad `Security.sandboxType`, del siguiente modo:

- `Security.REMOTE`: este archivo SWF procede de un URL de Internet y se rige por reglas de entorno limitado basadas en dominios.
- `Security.LOCAL_WITH_FILE`: el SWF es un archivo local y no es de confianza para el usuario. No se ha publicado con una designación de acceso a la red. El archivo SWF puede leer de orígenes de datos locales pero no puede comunicarse con Internet.
- `Security.LOCAL_WITH_NETWORK`: el SWF es un archivo local y no es de confianza para el usuario, pero se ha publicado con una designación de acceso a la red. El archivo SWF se puede comunicar con Internet pero no puede leer en fuentes de datos locales.
- `Security.LOCAL_TRUSTED`: el SWF es un archivo local y el usuario ha determinado que es de confianza, mediante el Administrador de configuración o un archivo de configuración de confianza de Flash Player. El archivo SWF puede leer de orígenes de datos locales y puede comunicarse con Internet.
- `Security.APPLICATION`: el archivo SWF se ejecuta en una aplicación AIR, y se instaló con el paquete (archivo AIR) de la aplicación. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden reutilizar los scripts de los archivos de cualquier dominio. No obstante, a los archivos que se encuentran fuera del entorno limitado de la aplicación AIR no se les permite reutilizar los scripts del archivo AIR. De forma predeterminada, los archivos del entorno limitado de la aplicación AIR pueden cargar contenido y datos desde cualquier dominio.

Controles de permiso

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El modelo de seguridad de tiempo de ejecución del cliente Flash Player se ha diseñado en torno a los recursos, que son objetos como, por ejemplo, archivos SWF, datos locales y URL de Internet. Las *personas con un interés directo* son aquellas que poseen o utilizan esos recursos. Estas personas pueden ejercer controles (configuración de seguridad) sobre sus propios recursos y cada recurso tiene cuatro personas con un interés directo. Flash Player aplica de forma estricta una jerarquía de autoridad para estos controles, como se muestra en la siguiente ilustración:

Jerarquía de controles de seguridad

Esto significa, por ejemplo, que si un administrador restringe el acceso a un recurso, ninguna otra persona con un interés directo puede anular dicha restricción.

Para las aplicaciones de AIR, estos controles de permiso solo se aplican al contenido que se ejecuta fuera del entorno limitado de la aplicación de AIR.

Controles de administrador

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un usuario administrador de un equipo, que haya iniciado sesión con derechos administrativos, puede aplicar una configuración de seguridad de Flash Player que afecte a todos los usuarios del equipo. En un entorno no empresarial, como puede ser un equipo doméstico, suele haber un usuario que tiene además acceso administrativo. Incluso en un entorno empresarial, los usuarios individuales pueden tener derechos administrativos en el equipo.

Hay dos tipos de controles de usuario administrador:

- El archivo mms.cfg
- El directorio Global Flash Player Trust

El archivo mms.cfg

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El archivo mms.cfg es un archivo de texto que permite a los administradores conceder o restringir el acceso a numerosas capacidades. Cuando se inicia Flash Player, lee la configuración de seguridad de este archivo y la utiliza para limitar la funcionalidad. El archivo mms.cfg incluye valores de configuración que el administrador utiliza para gestionar capacidades como, por ejemplo, controles de privacidad, seguridad de archivos locales, conexiones de socket, etc.

Un archivo SWF puede acceder a información sobre las capacidades desactivadas, mediante una llamada a las propiedades `Capabilities.avHardwareDisable` y `Capabilities.localFileReadDisable`. Sin embargo, la mayor parte de la configuración del archivo mms.cfg no puede consultarse desde ActionScript.

Para aplicar las políticas de privacidad y seguridad independientes de la aplicación en un equipo, el archivo mms.cfg solo deben modificarlo los administradores del sistema. Los archivos de instalación de aplicaciones no deben utilizar el archivo mms.cfg. Aunque un archivo de instalación que se ejecutara con privilegios de administrador podría modificar el contenido del archivo mms.cfg, Adobe considera que dicho uso infringe la confianza del usuario e insta a los creadores de los archivos de instalación que no modifiquen nunca el archivo mms.cfg.

El archivo mms.cfg se almacena en la ubicación siguiente:

- Windows: *system*\Macromed\Flash\mms.cfg

  (por ejemplo, C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac: *app support*/Macromedia/mms.cfg

  (por ejemplo, /Library/Application Support/Macromedia/mms.cfg)

Para obtener más información sobre el archivo mms.cfg, consulte la guía de administración de Flash Player www.adobe.com/go/flash_player_admin_es.

El directorio Global Flash Player Trust

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los usuarios con derechos administrativos y las aplicaciones de instalación pueden registrar archivos locales SWF específicos como de confianza para todos los usuarios. Estos archivos SWF se asignan al entorno limitado local de confianza. Pueden interactuar con cualquier otro archivo SWF y pueden cargar datos de cualquier ubicación local o remota. Los archivos se designan como de confianza en el directorio Global Flash Player Trust, en la ubicación siguiente:

- Windows: *system*\Macromed\Flash\FlashPlayerTrust

  (por ejemplo, C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac: *app support*/Macromedia/FlashPlayerTrust

  (por ejemplo, /Library/Application Support/Macromedia/FlashPlayerTrust)

El directorio Flash Player Trust puede contener cualquier número de archivos de texto, cada uno de los cuales contiene listas de rutas de confianza, con una ruta por cada línea. Cada ruta puede ser un archivo SWF individual, un archivo HTML o un directorio. Las líneas de comentario empiezan por el símbolo `#`. Por ejemplo, un archivo de configuración de confianza de Flash Player que contenga el siguiente texto concede el estado "de confianza" a todos los archivos en el directorio especificado y en todos sus subdirectorios:

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

Las rutas incluidas en un archivo de configuración de confianza deben ser siempre rutas locales o rutas de red SMB. Las rutas HTTP incluidas en un archivo de configuración de confianza se omiten; solo los archivos locales pueden ser de confianza.

Para evitar conflictos, debe asignarse a cada archivo de configuración de confianza un nombre de archivo correspondiente a la aplicación de instalación y utilizar una extensión de archivo .cfg.

Un desarrollador que distribuye un archivo SWF ejecutado localmente a través de una aplicación de instalación puede hacer que la aplicación añada un archivo de configuración al directorio Global Flash Player Trust, concediendo así privilegios completos al archivo que distribuye. La aplicación de instalación debe ejecutarse como usuario con derechos administrativos. A diferencia del archivo mms.cfg, el directorio Global Flash Player Trust se incluye para que las aplicaciones de instalación puedan conceder permisos de confianza. Tanto los usuarios administradores como las aplicaciones de instalación pueden designar aplicaciones locales de confianza a través del directorio Global Flash Player Trust.

Hay también directorios Flash Player Trust para usuarios individuales (consulte "Controles de usuario" en la página 1094).

Controles de usuario

Flash Player 9 y posterior

Flash Player ofrece tres mecanismos de nivel de usuario diferentes para definir permisos: la interfaz de usuario de configuración y el Administrador de configuración, y el directorio User Flash Player Trust.

La interfaz de usuario Configuración y el Administrador de configuración

Flash Player 9 y posterior

La interfaz de usuario Configuración es un mecanismo rápido e interactivo para establecer la configuración de un determinado dominio. El Administrador de configuración presenta una interfaz más detallada y permite realizar cambios globales que afectan a los permisos de muchos de los dominios o de todos ellos. Además, cuando un archivo SWF solicita un nuevo permiso que requiere tomar decisiones en tiempo de ejecución que afectan a la seguridad o privacidad, se muestran cuadros de diálogo en los que los usuarios pueden ajustar algunos parámetros de configuración de Flash Player.

La interfaz de usuario Configuración y el Administrador de configuración proporcionan opciones relacionadas con la seguridad como, por ejemplo, configuración de micrófono y cámara y de almacenamiento de objetos compartidos, configuración relacionada con el contenido heredado, etc. Ni el Administrador de configuración ni la Interfaz de usuario Configuración se encuentran disponibles en las aplicaciones de AIR.

Nota: *las configuraciones establecidas en el archivo mms.cfg (consulte "Controles de administrador" en la página 1092) no se reflejan en el Administrador de configuración.*

Para obtener información detallada sobre el Administrador de configuración, consulte www.adobe.com/go/settingsmanager_es.

El directorio User Flash Player Trust

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los usuarios y las aplicaciones de instalación pueden registrar determinados archivos SWF locales como archivos de confianza. Estos archivos SWF se asignan al entorno limitado local de confianza. Pueden interactuar con cualquier otro archivo SWF y pueden cargar datos de cualquier ubicación local o remota. Un usuario designa un archivo como archivo de confianza en el directorio User Flash Player Trust, que es el mismo directorio donde se almacenan los objetos compartidos, en las siguientes ubicaciones (las ubicaciones son específicas del usuario actual):

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (por ejemplo, C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust en Windows XP o C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust en Windows Vista)

  En Windows, la carpeta Datos de programa está oculta de forma predeterminada. Para mostrar carpetas y archivos ocultos, seleccione Mi PC para abrir el Explorador de Windows, seleccione Herramientas > Opciones de carpeta y, a continuación, la ficha Ver. En la ficha Ver, seleccione el botón de opción Mostrar todos los archivos y carpetas ocultos.

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (por ejemplo, /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  Esta configuración solo afecta al usuario actual, no a los demás usuarios que inician sesión en el equipo. Si un usuario sin derechos administrativos instala una aplicación en su parte del sistema, el directorio User Flash Player Trust permite al archivo de instalación registrar la aplicación como "de confianza" para dicho usuario.

  Un desarrollador que distribuye un archivo SWF ejecutado localmente a través de una aplicación de instalación puede hacer que se añada un archivo de configuración al directorio User Flash Player Trust, concediendo así privilegios completos al archivo que distribuye. Incluso en esta situación, el archivo del directorio User Flash Player Trust se considera un control de usuario, porque se inicia como consecuencia de una acción del usuario (la instalación).

  También hay un directorio Global Flash Player Trust que el usuario administrador o el archivo de instalación utiliza para registrar una aplicación para todos los usuarios de un equipo (consulte “Controles de administrador” en la página 1092).

Controles de sitio web (archivos de política)

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para que los datos de su servidor web estén disponibles para los archivos SWF de otros dominios, se puede crear un archivo de política en el servidor. Un *archivo de política* es un archivo XML situado en una ubicación específica de su servidor.

Los archivos de política afectan a varios activos, incluidos los siguientes:

- Datos en mapas de bits, sonidos y vídeos
- Carga de archivos de texto y XML
- Importación de archivos SWF desde otros dominios de seguridad en el dominio de seguridad del archivo SWF que realiza la carga
- Acceso a conexiones de socket y conexiones de socket XML

Los objetos ActionScript crean una instancia de dos tipos diferentes de conexiones de servidor: conexiones de servidor basadas en documentos y conexiones de socket. Los objetos de ActionScript como Loader, Sound, URLLoader y URLStream crean instancias de conexiones de servidor basadas en documentos y estos objetos cargan un archivo de una URL. Los objetos Socket y XMLSocket de ActionScript realizan conexiones de socket, que funcionan con datos de transmisión y no con documentos cargados.

Debido a que Flash Player admite dos tipos de conexiones de servidor, existen dos tipos de archivos de política: archivos de política URL y archivos de política de socket.

- Las conexiones basadas en documentos requieren *archivos de política URL*. Estos archivos permiten al servidor indicar que sus datos y documentos están disponibles para los archivos SWF de dominios determinados o de todos los dominios.
- Las conexiones de socket requieren *archivos de política de socket*, que permiten establecer redes directamente en el nivel inferior de socket TCP a través de las clases Socket y XMLSocket.

Flash Player requiere que los archivos de política se transmitan a través del mismo protocolo que pretende utilizar la conexión que se desea establecer. Por ejemplo, cuando se incluye un archivo de política en el servidor HTTP, los archivos SWF de otros dominios pueden cargar datos de él como un servidor HTTP. Sin embargo, si no se proporciona un archivo de política de socket en el mismo servidor, se prohibirá a los archivos SWF de otros dominios que se conecten con el servidor en el socket. Dicho de otro modo, la vía por la cual se recupera un archivo de política debe coincidir con la vía empleada en la conexión.

En el resto de esta sección se describen brevemente el uso y la sintaxis de los archivos de política, ya que se aplican a los archivos SWF publicados para Flash Player 10. (La implementación de archivos de política es ligeramente diferente en las versiones anteriores de Flash Player, ya que las versiones posteriores han reforzado la seguridad del programa.) Para obtener información más detallada sobre los archivos de política, consulte el tema sobre los cambios de los archivos de política en Flash Player 9 del Centro de desarrollo de Flash Player en la dirección www.adobe.com/go/devnet_security_es.

El código que se ejecuta en el entorno limitado de la aplicación de AIR no requiere ningún archivo de directiva para acceder a los datos de una URL o socket. El código de una aplicación de AIR que se ejecutan en un entorno limitado ajeno a la aplicación requiere un archivo de política.

Archivos maestros de política

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De forma predeterminada, Flash Player (y el contenido de AIR que no está en el entorno limitado de la aplicación AIR) busca un archivo de política URL denominado `crossdomain.xml` en el directorio raíz del servidor y un archivo de política de socket en el puerto 843. Los archivos que se encuentran en cualquiera de estas dos ubicaciones se denominan *archivos maestros de política*. (En el caso de las conexiones de socket, Flash Player también busca un archivo de política de socket en el mismo puerto que utiliza la conexión principal. No obstante, los archivos de política que se encuentran en dicho puerto no se consideran archivos maestros de política.)

Además de especificar los permisos de acceso, el archivo maestro de política puede contener también una sentencia *meta-policy*. Una metapolítica especifica las ubicaciones que pueden contener archivos de política. La metapolítica predeterminada de los archivos de política URL es “master-only”, es decir, /crossdomain.xml es el único archivo de política permitido en el servidor. La metapolítica predeterminada de los archivos de política de socket es “all”, es decir, cualquier socket del host puede servir un archivo de política de socket.

Nota: *en Flash Player 9 y versiones anteriores, la metapolítica predeterminada de los archivos de política URL era "all", es decir, cualquier directorio podía contener un archivo de política. Si ha implementado aplicaciones que cargan archivos de política desde ubicaciones diferentes al archivo predeterminado /crossdomain.xml, y dichas aplicaciones pudieran ejecutarse en Flash Player 10, asegúrese de que usted (o el administrador del servidor) modifica el archivo maestro de política para permitir archivos de política adicionales. Para obtener más información sobre cómo especificar una metapolítica diferente, consulte el tema sobre los cambios de los archivos de política en Flash Player 9 del Centro de desarrollo de Flash Player en la dirección www.adobe.com/go/devnet_security_es.*

Un archivo SWF puede comprobar un nombre de archivo de política o una ubicación de directorio diferentes llamando al método `Security.loadPolicyFile()`. Sin embargo, si el archivo maestro de política no especifica que la ubicación de destino puede servir archivos de política, la llamada a `loadPolicyFile()` no tendrá efecto, incluso si hay un archivo de política en dicha ubicación. Llame a `loadPolicyFile()` antes de intentar cualquier operación de red que requiera el archivo de política. Flash Player pone en cola automáticamente las peticiones de red detrás de sus correspondientes intentos de archivo de política. De este modo, es aceptable, por ejemplo, llamar a `Security.loadPolicyFile()` inmediatamente antes de iniciar una operación de red.

Al comprobar un archivo maestro de política, Flash Player espera durante tres segundos una respuesta del servidor. Si no recibe ninguna respuesta, Flash Player asumirá que no existe ningún archivo maestro de política. Sin embargo, no hay un valor de tiempo de espera predeterminado para las llamadas a `loadPolicyFile()`; Flash Player asume que el archivo al que se llama existe, y esperará el tiempo necesario para cargarlo. Por tanto, si desea asegurarse de que se carga un archivo maestro de política, utilice `loadPolicyFile()` para cargarlo explícitamente.

Aunque el método se denomina `Security.loadPolicyFile()`, no se cargará ningún archivo de política hasta que no se realice una llamada de red que requiera un archivo de política. Las llamadas a `loadPolicyFile()` solo indican a Flash Player la ubicación en la que buscar archivos de política cuando estos se necesitan.

No puede recibir una notificación del momento en el que se inicia o completa una solicitud de archivo de política, y no hay razón para ello. Flash Player realiza comprobaciones de política de forma asincrónica, y espera automáticamente a iniciar las conexiones hasta que las comprobaciones de archivos de política se hayan realizado correctamente.

Las secciones siguientes incluyen información que se aplica solo a los archivos de política URL. Para obtener más información sobre los archivos de política de socket, consulte "Conexión a sockets" en la página 1114.

Ámbito de los archivos de política URL

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un archivo de política URL solo se aplica al directorio desde el cual se carga y a sus directorios secundarios. Un archivo de política en el directorio raíz se aplica a todo el servidor. Un archivo de política cargado desde un subdirectorio arbitrario solo se aplica a dicho directorio y a sus subdirectorios.

Un archivo de política solo afecta al acceso al servidor concreto en el que reside. Por ejemplo, un archivo de política ubicado en https://www.adobe.com:8080/crossdomain.xml solo se aplica a las llamadas para cargar datos realizadas a www.adobe.com en el puerto 8080 utilizando el protocolo HTTPS.

Especificación de los permisos de acceso en un archivo de política URL

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Un archivo de política contiene una sola etiqueta `<cross-domain-policy>`, la cual no contiene ninguna o contiene varias etiquetas `<allow-access-from>`. Cada etiqueta `<allow-access-from>` contiene un atributo, `domain`, el cual especifica una dirección IP exacta, un dominio exacto o un dominio comodín (cualquier dominio). Los dominios comodín se indican de una de las dos formas siguientes:

- Mediante un solo asterisco (*), que coincide con todos los dominios y todas las direcciones IP
- Mediante un asterisco seguido de un sufijo, que coincide solo con los dominios que terminan con el sufijo especificado

Los sufijos deben empezar por un punto. Sin embargo, los dominios comodín con sufijos pueden incluir los dominios formados únicamente por el sufijo sin el punto inicial. Por ejemplo, xyz.com se considera parte de *.xyz.com. Los comodines no se pueden utilizar en las especificaciones de dominio IP.

En el ejemplo siguiente se muestra un archivo de política URL que permite el acceso a archivos SWF procedentes de *.example.com, www.friendOfExample.com y 192.0.34.166:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

Si especifica una dirección IP, solo se otorgará acceso a los archivos SWF cargados desde esa dirección IP mediante la sintaxis de IP (por ejemplo, http://65.57.83.12/flashmovie.swf). No se concederá acceso a los archivos SWF a través de la sintaxis de nombre de dominio. Flash Player no lleva a cabo la resolución DNS.

Se puede permitir el acceso a documentos procedentes de cualquier dominio, tal y como se muestra en el siguiente ejemplo:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Cada etiqueta `<allow-access-from>` tiene además el atributo opcional `secure`, que tiene el valor predeterminado `true`. Si su archivo de política se encuentra en un servidor HTTPS y desea permitir que los archivos SWF de un servidor que no sea HTTPS puedan cargar datos del servidor HTTPS, puede establecer el atributo en `false`.

Establecer el atributo `secure` en `false` puede poner en peligro la seguridad proporcionada por HTTPS. En concreto, establecer este atributo en `false` deja el contenido seguro expuesto a posibles ataques de fisgones y suplantadores. Adobe recomienda encarecidamente no establecer el atributo `secure` en `false`.

Si los datos que se van a cargar se encuentran en un servidor HTTPS, pero el archivo SWF que los carga se encuentra en un servidor HTTP, Adobe recomienda mover el archivo SWF de carga a un servidor HTTPS. De este modo, se podrán mantener todas las copias de sus datos seguros bajo la protección de HTTPS. Sin embargo, si se decide mantener el archivo SWF que realiza la carga en un servidor HTTP, deberá añadirse el atributo `secure="false"` a la etiqueta `<allow-access-from>`, como se muestra en el código siguiente:

```
<allow-access-from domain="www.example.com" secure="false" />
```

Otro elemento que se puede utilizar para permitir el acceso es la etiqueta `allow-http-request-headers-from`. Este elemento permite a un cliente que aloja contenido de otro dominio de permisos enviar a su dominio encabezados definidos por el usuario. Mientras que la etiqueta `<allow-access-from>` concede permiso a otros dominios para extraer datos de su dominio, la etiqueta `allow-http-request-headers-from` concede permiso a otros dominios para introducir datos en el mismo, en la forma de encabezados. En el ejemplo siguiente, se permite a cualquier dominio enviar el encabezado SOAPAction al dominio actual:

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Si la sentencia `allow-http-request-headers-from` se encuentra en el archivo maestro de política, se aplicará a todos los directorios del host. De lo contrario, solo se aplicará al directorio y los subdirectorios del archivo de política que contiene la sentencia.

Precarga de archivos de política

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La carga de datos de un servidor o la conexión a un socket es una operación asincrónica. Flash Player simplemente espera a que el archivo de política acabe de descargarse para comenzar la operación principal. Sin embargo, la extracción de datos de píxeles de imágenes o la extracción de datos de ejemplo de sonidos es una operación sincrónica. Para poder extraer datos, primero se debe cargar el archivo de política. Al cargar un medio, especifique que busque un archivo de política:

- Si utiliza el método `Loader.load()`, establezca la propiedad `checkPolicyFile` del parámetro `context`, que es un objeto LoaderContext.
- Si incorpora una imagen en un campo de texto mediante la etiqueta `<img>`, establezca el atributo `checkPolicyFile` de la etiqueta `<img>` en `"true"`, como en el siguiente ejemplo:

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- Si utiliza el método `Sound.load()`, establezca la propiedad `checkPolicyFile` del parámetro `context`, que es un objeto SoundLoaderContext.
- Si utiliza la clase NetStream, establezca la propiedad `checkPolicyFile` del objeto NetStream.

Si se establece uno de estos parámetros, Flash Player comprueba primero si hay algún archivo de política que ya se haya descargado para dicho dominio. A continuación, busca el archivo de política en la ubicación predeterminada del servidor, comprobando las sentencias `<allow-access-from>` y la presencia de una metapolítica. Por último, considera las llamadas pendientes al método `Security.loadPolicyFile()` para comprobar si se encuentran dentro del ámbito correspondiente.

Controles de autor (desarrollador)

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La principal API de ActionScript que se utiliza para conceder privilegios de seguridad es el método `Security.allowDomain()`, que concede privilegios a archivos SWF en los dominios especificados. En el siguiente ejemplo, un archivo SWF concede acceso a archivos SWF que se encuentran disponibles en el dominio www.example.com:

```
Security.allowDomain("www.example.com")
```

Este método concede permisos para lo siguiente:

- Reutilización de scripts entre archivos SWF (consulte "Reutilización de scripts" en la página 1108)
- Acceso a la lista de visualización (consulte "Recorrido de la lista de visualización" en la página 1110)
- Detección de eventos (consulte "Seguridad de eventos" en la página 1111)
- Acceso completo a las propiedades y métodos del objeto Stage (consulte "Seguridad del objeto Stage" en la página 1109)

El principal objetivo de llamar al método `Security.allowDomain()` es conceder permiso para que los archivos SWF de un dominio exterior puedan manipular mediante script el archivo SWF, a través de una llamada al método `Security.allowDomain()`. Para obtener más información, consulte "Reutilización de scripts" en la página 1108.

Cuando se especifica una dirección IP como parámetro en el método `Security.allowDomain()`, no se permite el acceso de todas las partes que tienen su origen en la dirección IP especificada. Solo se permite el acceso de la parte que contiene la dirección IP especificada en su URL, y no un nombre de dominio que corresponda a la dirección IP. Por ejemplo, si el nombre de dominio www.example.com corresponde a la dirección IP 192.0.34.166, una llamada a `Security.allowDomain("192.0.34.166")` no concede el acceso a www.example.com.

Se puede pasar el comodín `"*"` al método `Security.allowDomain()` para permitir el acceso desde todos los dominios. Como se concede permiso a los archivos SWF de *todos* los dominios para que manipulen mediante script el archivo SWF que realiza la llamada, se debe utilizar el comodín `"*"` con precaución.

ActionScript incluye una segunda API de permiso, denominada `Security.allowInsecureDomain()`. Este método realiza lo mismo que el método `Security.allowDomain()` y además, cuando se llama desde un archivo SWF que se encuentra disponible en una conexión HTTPS segura, permite el acceso al archivo SWF que realiza la llamada por parte de otros archivos SWF que se encuentran disponibles en un protocolo no seguro, como HTTP. Sin embargo, no es seguro permitir la reutilización de scripts entre archivos desde un protocolo seguro (HTTPS) y desde protocolos no seguros (como HTTP); si se permite, se deja el contenido seguro expuesto a posibles ataques de fisgones y suplantadores. Este es el modo en que pueden funcionar estos ataques: como el método `Security.allowInsecureDomain()` permite que los archivos SWF proporcionados a través de conexiones HTTP puedan acceder a los datos de HTTPS seguro, un atacante interpuesto entre el servidor HTTP y los usuarios podría sustituir el archivo SWF de HTTP por uno propio y acceder así a los datos HTTPS.

Importante: *no se permite que el código que se ejecuta en el entorno limitado de la aplicación de AIR llame a los métodos* `allowDomain()` *o* `allowInsecureDomain()` *de la clase Security.*

Otro método importante relacionado con la seguridad es el método `Security.loadPolicyFile()`, que hace que Flash Player compruebe si hay un archivo de política en una ubicación no estándar. Para obtener más información, consulte "Controles de sitio web (archivos de política)" en la página 1095.

Restricción de las API de red

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las API de red se pueden restringir de dos formas. Para evitar actividad malintencionada, se bloquea el acceso a los puertos que suelen estar reservados; estos bloques no se pueden sustituir en el código. Para controlar el acceso de un archivo SWF a la funcionalidad de red con respecto a otros puertos, puede utilizar la configuración `allowNetworking`.

Puertos bloqueados

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player y Adobe AIR, al igual que los navegadores, tienen restricciones para el acceso HTTP a determinados puertos. No se permiten las peticiones HTTP a determinados puertos estándar que se utilizan tradicionalmente para tipos de servidores que no son HTTP.

Las API que acceden a una URL de red están sujetas a estas restricciones de bloqueo de puertos. La única excepción son las API que llaman directamente a sockets, como `Socket.connect()` y `XMLSocket.connect()`, o las llamadas a `Security.loadPolicyFile()`, donde se carga un archivo de política de socket. Las conexiones de socket se permiten o deniegan a través del uso de archivos de política de socket en el servidor de destino.

La lista siguiente muestra las API de ActionScript 3.0 donde se aplica el bloqueo de puertos:

`FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `Loader.loadBytes()`, `navigateToURL()`, `NetConnection.call()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `sendToURL()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`

El bloqueo de puertos también se aplica a la importación de biblioteca compartida, al uso de la etiqueta `<img>` en campos de texto y a la carga de archivos SWF en páginas HTML a través de las etiquetas `<object>` y `<embed>`.

El bloqueo de puertos también se aplica al uso de la etiqueta `<img>` en campos de texto y a la carga de archivos SWF en páginas HTML a través de las etiquetas `<object>` y `<embed>`.

Las listas siguientes muestran los puertos que están bloqueados:

HTTP: 20 (datos ftp), 21 (control ftp)

HTTP y FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

Uso del parámetro allowNetworking

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede controlar el acceso de un archivo SWF a la funcionalidad de red configurando el parámetro `allowNetworking` en las etiquetas `<object>` y `<embed>` de la página HTML que aloja el contenido SWF.

Los valores posibles de `allowNetworking` son:

- `"all"` (valor predeterminado): se admiten todas las API de red en el archivo SWF.
- `"internal"`: el archivo SWF no puede llamar a las API de navegación o de interacción con el navegador que se muestran a continuación, pero puede llamar a las otras API de red.
- `"none"`: el archivo SWF no puede llamar a ninguna de las API de navegación o de interacción con el navegador que se muestran a continuación, y no puede usar ninguna de las API de comunicación entre archivos SWF, incluidas también en la lista siguiente.

El parámetro `allowNetworking` se ha diseñado principalmente para los casos en los que el archivo SWF y la página HTML que lo incluye pertenecen a dominios diferentes. No se recomienda el uso del valor de `"internal"` ni `"none"` cuando el archivo SWF que se carga pertenece al mismo dominio que las páginas HTML que lo incluyen, porque no se puede garantizar que un archivo SWF se cargue siempre con la página HTML deseada. Las partes que no son de confianza podrían cargar un archivo SWF de su dominio sin ninguna página HTML que lo incluya, en cuyo caso la restricción `allowNetworking` no funcionará del modo esperado.

Al llamar a una API no permitida, se emite una excepción SecurityError.

Añada el parámetro `allowNetworking` y establezca su valor en las etiquetas `<object>` y `<embed>` de la página HTML que contiene una referencia al archivo SWF, como se muestra en el ejemplo siguiente:

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code
base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,
0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una página HTML también puede utilizar un script para generar etiquetas que incorporan SWF. Necesita alterar el script para que inserte el valor `allowNetworking` adecuado. Las páginas HTML generadas por Adobe Flash Professional y Adobe Flash Builder utilizan la función `AC_FL_RunContent()` para incorporar referencias a archivos SWF. Añada la configuración del parámetro `allowNetworking` al script, como en el siguiente ejemplo:

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Las siguientes API no se admiten cuando `allowNetworking` está establecido en `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

Además de las API anteriores, no se admiten las siguientes API cuando `allowNetworking` se establece en `"none"`:

`sendToURL()`, `FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `LocalConnection.connect()`, `LocalConnection.send()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `SharedObject.getLocal()`, `SharedObject.getRemote()`, `Socket.connect()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`, `XMLSocket.connect()`

Aunque la configuración de `allowNetworking` seleccionada permita a un archivo SWF usar una API de red, puede haber otras restricciones basadas en las limitaciones de un entorno limitado de seguridad (consulte "Entornos limitados de seguridad" en la página 1087).

Cuando `allowNetworking` se establece en `"none"`, no se puede hacer referencia a medios externos en una etiqueta `<img>` en la propiedad `htmlText` de un objeto TextField (se emite una excepción SecurityError).

Si se establece `allowNetworking` en `"none"`, un símbolo de una biblioteca compartida importada añadido en la herramienta de edición Flash (no mediante ActionScript) se bloqueará en tiempo de ejecución.

Seguridad del modo de pantalla completa

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En Player 9.0.27.0 y versiones posteriores se admite el modo de pantalla completa, en el que el contenido que se ejecuta en Flash Player puede llenar toda la pantalla. Para entrar en el modo de pantalla completa, se establece la constante `StageDisplayState.FULL_SCREEN` como valor de la propiedad `displayState` de Stage. Para obtener más información, consulte "Trabajo con el modo de pantalla completa" en la página 171.

Hay que tener en cuenta algunas consideraciones de seguridad relacionadas con los archivos SWF que se ejecutan en un entorno limitado remoto.

Para activar el modo de pantalla completa, en las etiquetas `<object>` y `<embed>` de la página HTML que contiene una referencia al archivo SWF, añada el parámetro `allowFullScreen`, con su valor establecido en `"true"` (el valor predeterminado es `"false"`), tal y como se muestra en el siguiente ejemplo:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una página HTML también puede utilizar un script para generar etiquetas que incorporan SWF. Es necesario modificar el script para insertar la configuración adecuada de `allowFullScreen`. Las páginas HTML generadas por Flash Professional y Flex Builder utilizan la función `AC_FL_RunContent()` para incorporar referencias a archivos SWF y es necesario añadir la configuración del parámetro `allowFullScreen` al script, como en el siguiente ejemplo:

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

El código ActionScript que inicia el modo de pantalla completa solo puede llamarse como respuesta a un evento de ratón o de teclado. Si se llama en otras situaciones, Flash Player emite una excepción.

Cuando se entra en el modo de pantalla completa, aparece un mensaje que indica al usuario cómo puede salir y volver al modo normal. El mensaje aparece durante unos segundos y luego desaparece progresivamente.

Para el contenido que se está ejecutando en un navegador, el uso del teclado se limita al modo de pantalla completa. En Flash Player 9, solo se admiten los métodos abreviados de teclado que devuelven la aplicación a modo normal como, por ejemplo, presionar la tecla Esc. Los usuarios no pueden introducir texto en los campos de texto ni desplazarse por la pantalla. En Flash Player 10 y versiones posteriores, se admiten determinadas teclas que no afectan a la impresión (concretamente, las teclas de flecha, la barra espaciadora y el tabulador). Sin embargo, la entrada de texto aún está prohibida.

El modo de pantalla completa siempre se permite en el reproductor autónomo o en un archivo de proyector. Asimismo, el uso del teclado (incluyendo la entrada de texto) es totalmente compatible en estos entornos.

Al llamar a la propiedad `displayState` de un objeto Stage, se emite una excepción para cualquier origen de llamada que no esté en el mismo entorno limitado de seguridad que el propietario del objeto Stage (el archivo SWF principal). Para obtener más información, consulte "Seguridad del objeto Stage" en la página 1109.

Los administradores pueden desactivar el modo de pantalla completa en los archivos SWF que se ejecutan en navegadores. Para ello, deben establecer `FullScreenDisable = 1` en el archivo mms.cfg. Para obtener más información, consulte "Controles de administrador" en la página 1092.

En un navegador, un archivo SWF debe estar contenido en una página HTML para poder verlo en el modo de pantalla completa.

Seguridad del modo de pantalla completa

Flash Player 11.3 y posterior, Adobe AIR 1.0 y posterior

En Flash Player 11.3 y versiones posteriores se admite el modo de pantalla completa, en el que el contenido que se ejecuta en Flash Player puede llenar toda la pantalla *y aceptar introducción de texto*. Para entrar en el modo de pantalla completa, la propiedad `displayState` de Stage se establece en la constante `StageDisplayState.FULL_SCREEN_INTERACTIVE`. Para obtener más información, consulte "Trabajo con el modo de pantalla completa" en la página 171.

Hay que tener en cuenta algunas consideraciones de seguridad relacionadas con los archivos SWF que se ejecutan en un entorno limitado remoto.

Para activar el modo de pantalla completa, en las etiquetas `<object>` y `<embed>` de la página HTML que contiene una referencia al archivo SWF, añada el parámetro `allowFullScreenInteractive`, con su valor establecido en `"true"` (el valor predeterminado es `"false"`), tal y como se muestra en el siguiente ejemplo:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreenInteractive" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Una página HTML también puede utilizar un script para generar etiquetas que incorporan SWF. Es necesario modificar el script para insertar la configuración adecuada de `allowFullScreenInteractive`. Las páginas HTML generadas por Flash Professional y Flex Builder utilizan la función `AC_FL_RunContent()` para incorporar referencias a archivos SWF y es necesario añadir la configuración del parámetro `allowFullScreenInteractive` al script, como en el siguiente ejemplo:

```
AC_FL_RunContent( ... "allowFullScreenInteractive", "true", ...)
```

El código ActionScript que inicia el modo de pantalla completa solo puede llamarse como respuesta a un evento de ratón o de teclado. Si se llama en otras situaciones, Flash Player emite una excepción.

Aparece un mensaje de superposición si el contenido accede al modo de pantalla completa. El mensaje muestra el dominio de la página a pantalla completa, instrucciones sobre cómo salir del modo de pantalla completa y un botón **Permitir**. El mensaje no desaparece hasta que el usuario hace clic en **Permitir**, confirmando así que accede al modo de pantalla completa.

Los administradores pueden desactivar el modo de pantalla completa en los archivos SWF que se ejecutan en navegadores. Para ello, deben establecer `FullScreenInteractiveDisable = 1` en el archivo mms.cfg. Para obtener más información, consulte "Controles de administrador" en la página 1092.

En un navegador, un archivo SWF debe estar contenido en una página HTML para poder verlo en el modo de pantalla completa.

Carga de contenido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El contenido de Flash Player y AIR puede cargar distintos tipos de contenido, incluyendo los siguientes:

- Archivos SWF
- Imágenes
- Sonido
- Vídeo
- Archivos HTML (solo AIR)
- JavaScript (solo AIR)

Carga de imágenes y archivos SWF con la clase Loader

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase Loader se utiliza para cargar archivos SWF e imágenes (archivos JPG, GIF o PNG). Cualquier archivo SWF que no se encuentre en el entorno limitado local con sistema de archivos puede cargar archivos SWF e imágenes desde cualquier dominio de red. Solo los archivos SWF de entornos limitados locales pueden cargar archivos SWF e imágenes del sistema de archivos local. Sin embargo, los archivos del entorno limitado local con acceso a la red solo pueden cargar archivos SWF que se encuentren en el entorno limitado local de confianza o en el entorno limitado local con acceso a la red. Los archivos SWF del entorno limitado local con acceso a la red cargan contenido local que no sean archivos SWF (por ejemplo, imágenes), pero no pueden acceder a los datos del contenido cargado.

Al cargar un archivo SWF de un origen que no es de confianza (por ejemplo, un dominio distinto al del archivo SWF raíz del objeto Loader), es aconsejable definir una máscara para el objeto Loader para evitar que el contenido cargado (que es un elemento secundario del objeto Loader) se dibuje en partes del escenario situadas fuera de la máscara, como se muestra en el siguiente código:

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Cuando se llama al método `load()` del objeto Loader, se puede especificar un parámetro `context`, que es un objeto LoaderContext. La clase LoaderContext incluye tres propiedades que permiten definir el contexto de uso del contenido cargado:

- `checkPolicyFile`: utilice esta propiedad solo si carga un archivo de imagen (no un archivo SWF). Esta propiedad se especifica en un archivo de imagen de un dominio ajeno al del archivo que contiene el objeto Loader. Si establece esta propiedad en `true`, Loader comprueba el servidor de origen de un archivo de política URL (consulte "Controles de sitio web (archivos de política)" en la página 1095). Si el servidor concede permiso al dominio de Loader, el código ActionScript de los archivos SWF del dominio de Loader puede acceder a los datos de la imagen cargada. Dicho de otro modo, se puede utilizar la propiedad `Loader.content` para obtener una referencia al objeto Bitmap que representa la imagen cargada, o los métodos `BitmapData.draw()` o `BitmapData.drawWithQuality()` para acceder a los píxeles de la imagen cargada. El método `drawWithQuality` está disponible en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores.
- `securityDomain`: utilice esta propiedad solo si carga un archivo SWF (no una imagen). Esta propiedad se especifica en un archivo SWF de un dominio ajeno al del archivo que contiene el objeto Loader. Actualmente, la propiedad `securityDomain` solo admite dos valores: `null` (el valor predeterminado) y `SecurityDomain.currentDomain`. Si se especifica `SecurityDomain.currentDomain`, el archivo SWF cargado debe *importarse* en el entorno limitado del objeto SWF que realiza la carga, lo que significa que funciona como si se hubiera cargado del servidor del archivo SWF que realiza la carga. Esto solo se permite si se encuentra un archivo de política URL en el servidor del archivo SWF cargado que permita el acceso por parte del dominio del archivo SWF que realiza la carga. Si se encuentra el archivo de política necesario, el cargador y el contenido cargado pueden manipularse mutuamente mediante script en cuanto se inicia la carga, ya que se encuentran en el mismo entorno limitado. El entorno limitado donde se importa el archivo puede sustituirse a través de una carga normal, seguida de una llamada del archivo SWF cargado al método `Security.allowDomain()`. Es posible que este último método sea más sencillo, pues el archivo SWF cargado estará entonces en su entorno limitado natural y, por lo tanto, podrá acceder a los recursos de su propio servidor real.
- `applicationDomain`: utilice esta propiedad solamente si carga un archivo SWF escrito en ActionScript 3.0 (no una imagen ni un archivo SWF escritos en ActionScript 1.0 o 2.0). Al cargar el archivo, puede especificar que se incluya el archivo en un dominio de aplicación concreto y no en un nuevo dominio de aplicación que sea un elemento secundario del dominio de aplicación del archivo SWF que realiza la carga, que es lo que sucede de forma predeterminada. Tenga en cuenta que los dominios de aplicación son subunidades de los dominios de seguridad y, por lo tanto, puede especificar un dominio de aplicación de destino únicamente si el archivo SWF que está cargando procede de su propio dominio de seguridad, ya sea porque corresponde a su servidor propio o porque lo ha importado en su dominio de seguridad a través de la propiedad `securityDomain`. Si especifica un dominio de aplicación pero el archivo SWF cargado forma parte de un dominio de seguridad distinto, el dominio especificado en `applicationDomain` se omite. Para obtener más información, consulte "Trabajo con dominios de aplicación" en la página 150.

Para obtener más detalles, consulte "Especificación del contexto de carga" en la página 207.

Una propiedad importante de un objeto Loader es `contentLoaderInfo`, que es un objeto LoaderInfo. A diferencia de lo que ocurre con la mayoría de los objetos, un objeto LoaderInfo se comparte entre el archivo SWF que realiza la carga y el contenido cargado, y siempre es accesible para ambas partes. Cuando el contenido cargado es un archivo SWF, este puede acceder al objeto LoaderInfo a través de la propiedad `DisplayObject.loaderInfo`. Los objetos LoaderInfo contienen información como el progreso de carga, los URL del cargador y del contenido cargado, o la relación de confianza entre ambos. Para obtener más información, consulte "Supervisión del progreso de carga" en la página 205.

Carga de sonido y vídeos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cualquier contenido, excepto el contenido que se encuentre en el entorno limitado local con sistema de archivos, puede cargar sonido y vídeo de orígenes de red, a través de los métodos `Sound.load()`, `NetConnection.connect()` y `NetStream.play()`.

Solo el contenido de los entornos limitados local con sistema de archivos y de la aplicación de AIR pueden cargar medios desde el sistema de archivos local. Solo el contenido del entorno limitado local con sistema de archivos, del entorno local e la aplicación de AIR o del entorno limitado local de confianza puede acceder a los datos de estos archivos cargados.

Hay otras restricciones relativas al acceso de datos desde medios cargados. Para obtener información más detallada, consulte "Acceso a medios cargados como datos" en la página 1111.

Carga de archivos SWF e imágenes mediante la etiqueta <img> de un campo de texto

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Puede cargar archivos SWF y mapas de bits en un campo de texto mediante la etiqueta `<img>`, como se muestra en el código siguiente:

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Para acceder al contenido cargado de este modo, puede utilizar el método `getImageReference()` de la instancia de TextField, como se muestra en el código siguiente:

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

Sin embargo, tenga en cuenta que los archivos SWF e imágenes que se cargan de este modo se incluyen en el entorno limitado correspondiente a su origen.

Si se carga un archivo de imagen mediante una etiqueta `<img>` en un campo de texto, puede ser que haya un archivo de política URL que permita el acceso a datos de la imagen. Para comprobar si hay un archivo de política, añada un atributo `checkPolicyFile` a la etiqueta `<img>`, como se muestra en el código siguiente:

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Si carga un archivo SWF mediante una etiqueta `<img>` en un campo de texto, puede permitir el acceso a los datos de dicho archivo SWF a través de una llamada al método `Security.allowDomain()`.

Si utiliza una etiqueta `<img>` en un campo de texto para cargar un archivo externo (en lugar de usar una clase Bitmap incorporada en el archivo SWF), se crea automáticamente un objeto Loader como elemento secundario del objeto TextField y el archivo externo se carga en dicho objeto Loader, tal y como sucedería si hubiera utilizado un objeto Loader en ActionScript para cargar el archivo. En este caso, el método `getImageReference()` devuelve el objeto Loader que se creó automáticamente. No es necesario realizar ninguna comprobación de seguridad para acceder a este objeto Loader porque se encuentra en el mismo entorno limitado de seguridad que el código que realiza la llamada.

Sin embargo, si se hace referencia a la propiedad `content` del objeto Loader para acceder al medio cargado, sí es preciso aplicar las reglas de seguridad. Si el contenido es una imagen, deberá implementar un archivo de política URL y, si es un archivo SWF, deberá hacer que el código del archivo SWF llame al método `allowDomain()`.

Adobe AIR

En el entorno limitado de la aplicación, las etiquetas <img> de un campo de texto se omiten para evitar ataques de suplantación de identidad. Asimismo, no se permite que el código que se ejecuta en el entorno limitado de la aplicación llame al método `allowDomain()` de Security.

Contenido proporcionado a través de servidores RTMP

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Media Server utiliza el protocolo RTMP (Real-Time Media Protocol) para proporcionar datos, audio y vídeo. Puede cargar estos medios mediante el método `connect()` de la clase NetConnection y pasa una URL RTMP como parámetro. Flash Media Server puede restringir las conexiones y evitar la descarga del contenido, en función del dominio del archivo que realiza la solicitud. Para obtener más información, consulte la documentación de Flash Media Server en línea en www.adobe.com/go/learn_fms_docs_es.

Para utilizar los métodos `BitmapData.draw()`, `BitmapData.drawWithQuality()` y `SoundMixer.computeSpectrum()` para extraer gráficos en tiempo de ejecución y datos de sonido de los flujos RTMP, debe permitir el acceso en el servidor. Utilice las propiedades `Client.videoSampleAccess` y `Client.audioSampleAccess` de ActionScript del servidor para permitir acceso a directorios específicos en Flash Media Server. Para obtener más información, consulte la referencia del lenguaje ActionScript de Flash Media Server. (El método `drawWithQuality` está disponible en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores.)

Reutilización de scripts

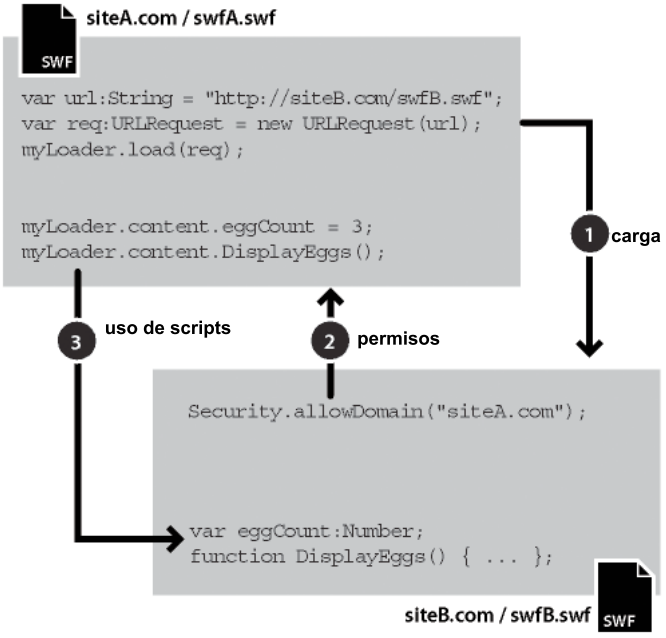
Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Si hay dos archivos escritos en ActionScript 3.0 o dos archivos HTML que se ejecutan en el mismo dominio (por ejemplo, el URL de un archivo SWF es http://www.example.com/swfA.swf y el URL del otro es http://www.example.com/swfB.swf), el código definido en un archivo puede examinar y modificar las variables, objetos, propiedades, métodos, etc. en el otro archivo y viceversa. Esto se denomina *reutilización de scripts*.

Si dos archivos proceden de diferentes dominios como, por ejemplo, http://siteA.com/swfA.swf y http://siteB.com/swfB.swf, de forma predeterminada, Flash Player y AIR no permiten que swfA.swf determine swfB.swf, ni que swfB.swf determine swfA.swf. Un archivo SWF concede permiso a archivos SWF de otros dominios llamando a `Security.allowDomain()`. Si se llama a `Security.allowDomain("siteA.com")`, swfB.swf concede permiso a los archivos SWF de siteA.com para utilizarlo en scripts.

La reutilización de scripts no se permite entre archivos SWF AVM1 y archivos SWF AVM2. Un archivo SWF AVM1 es aquel que se crea en ActionScript 1.0 o ActionScript 2.0. (AVM1 y AVM2 se refieren a la máquina virtual ActionScript.) Sin embargo, es posible utilizar la clase LocalConnection para enviar datos entre AVM1 y AVM2.

En cualquier operación entre dominios es importante tener claro qué dos partes están involucradas. En esta sección, llamaremos *parte que accede* a la que lleva a cabo la reutilización de scripts (normalmente el archivo SWF que accede a otro) y *parte a la que se accede* a la otra (por lo general, el archivo SWF al que se accede). Si siteA.swf usa scripts en siteB.swf, siteA.swf será la parte que accede y siteB.swf será la parte a la que se accede, tal y como se indica en la siguiente ilustración:



Los permisos entre dominios establecidos con el método `Security.allowDomain()` son asimétricos. En el ejemplo anterior, siteA.swf puede manipular mediante script a siteB.swf, pero siteB.swf no puede hacerlo con siteA.swf, ya que siteA.swf no ha llamado al método `Security.allowDomain()` para dar permiso a los archivos SWF de siteB.com para manipularlo mediante script. Para configurar permisos simétricos, es necesario que ambos archivos SWF llamen al método `Security.allowDomain()`.

Flash Player protege los archivos SWF no solo de los scripts creados en varios dominios por otros archivos SWF, sino también de los originados por archivos HTML. El uso de scripts de HTML en SWF puede producirse mediante funciones callback establecidas a través del método `ExternalInterface.addCallback()`. Cuando el uso de scripts de HTML en SWF traspasa los dominios, el archivo SWF debe llamar al método `Security.allowDomain()` para evitar que la operación falle, tanto si es la parte que accede como si es la parte a la que se accede. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 1099.

Además, Flash Player proporciona controles de seguridad para el uso de scripts de SWF en HTML. Para obtener más información, consulte "Control del acceso URL saliente" en la página 1119.

Seguridad del objeto Stage

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Algunas propiedades y métodos del objeto Stage solo están disponibles para los objetos Sprite o MovieClip de la lista de visualización.

No obstante, se dice que el objeto Stage tiene un propietario: el primer archivo SWF cargado. De forma predeterminada, las siguientes propiedades y métodos del objeto Stage solo están disponibles para los archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el propietario de Stage:

Propiedades	Métodos
`align`	`addChild()`
displayState	`addChildAt()`
frameRate	`addEventListener()`
`height`	`dispatchEvent()`
`mouseChildren`	`hasEventListener()`
`numChildren`	`setChildIndex()`
`quality`	`willTrigger()`
`scaleMode`	
`showDefaultContextMenu`	
`stageFocusRect`	
`stageHeight`	
`stageWidth`	
`tabChildren`	
`textSnapshot`	
`width`	

Para que un archivo SWF de un entorno limitado ajeno al del propietario de Stage pueda acceder a estas propiedades y métodos, el archivo SWF del propietario de Stage debe llamar al método `Security.allowDomain()` para permitir el dominio del entorno limitado externo. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 1099.

La propiedad `frameRate` es especial, ya que cualquier archivo SWF puede leerla. Sin embargo, solo pueden cambiar la propiedad los archivos que se encuentran en el entorno limitado de seguridad del propietario de Stage (o los que han obtenido permiso a través de una llamada al método `Security.allowDomain()`).

También hay restricciones relativas a los métodos `removeChildAt()` y `swapChildrenAt()` del objeto Stage, pero son distintas a las demás restricciones. En lugar de tener que estar en el mismo dominio que el propietario de Stage, para llamar a estos métodos, el código debe estar en el mismo dominio que el propietario de los objetos secundarios afectados, o bien los objetos secundarios pueden llamar al método `Security.allowDomain()`.

Recorrido de la lista de visualización

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Existen restricciones con respecto a la capacidad de un archivo SWF para acceder a los objetos de visualización cargados desde otros entornos limitados. Para que un archivo SWF pueda acceder a un objeto de visualización creado por otro archivo SWF en un entorno limitado distinto, el archivo SWF al que se accede debe llamar al método `Security.allowDomain()` para permitir el acceso desde el dominio del archivo SWF que accede. Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 1099.

Para acceder a un objeto Bitmap cargado por un objeto Loader, debe existir un archivo de política URL en el servidor de origen del archivo de imagen y debe conceder permiso al dominio del archivo SWF que intenta acceder al objeto Bitmap (consulte "Controles de sitio web (archivos de política)" en la página 1095).

El objeto LoaderInfo correspondiente a un archivo cargado (y al objeto Loader) contiene las tres propiedades siguientes, que definen la relación entre el objeto cargado y el objeto Loader: `childAllowsParent`, `parentAllowsChild` y `sameDomain`.

Seguridad de eventos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los eventos relacionados con la lista de visualización tienen limitaciones de acceso de seguridad que dependen del entorno limitado donde se encuentre el objeto de visualización que distribuye el evento. Un evento de la lista de visualización tiene fases de propagación y captura (descritas en "Gestión de eventos" en la página 127). Durante las fases de propagación y captura, un evento migra del objeto de visualización de origen a través de los objetos de visualización principales en la lista de visualización. Si un objeto principal se encuentra en un entorno limitado de seguridad distinto al del objeto de visualización de origen, la fase de captura y propagación se detiene debajo de dicho objeto principal, a menos que exista una confianza mutua entre el propietario del objeto principal y el propietario del objeto de origen. Esta confianza mutua se obtiene del siguiente modo:

1. El archivo SWF propietario del objeto principal debe llamar al método `Security.allowDomain()` para confiar en el dominio del archivo SWF propietario del objeto de origen.
2. El archivo SWF propietario del objeto de origen debe llamar al método `Security.allowDomain()` para confiar en el dominio del archivo SWF propietario del objeto principal.

El objeto LoaderInfo correspondiente a un archivo cargado (y al objeto Loader) contiene las dos propiedades siguientes, que definen la relación entre el objeto cargado y el objeto Loader: `childAllowsParent` y `parentAllowsChild`.

En los eventos distribuidos desde objetos que no sean objetos de visualización, no se realizan comprobaciones de seguridad.

Acceso a medios cargados como datos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para acceder a datos de carga, utilice los métodos `BitmapData.draw()`, `BitmapData.drawWithQuality()` y `SoundMixer.computeSpectrum()`. De forma predeterminada, no se pueden obtener datos de píxeles o datos de audio en objetos gráficos o de audio procesados o reproducidos por medios cargados en un entorno limitado diferente. No obstante, se pueden utilizar los métodos siguientes para otorgar permiso para acceder a dichos datos en los límites del entorno limitado:

- En el contenido que procesa o reproduce los datos a los que se va a acceder, llame al método `Security.allowDomain()` para permitir el acceso a los datos al contenido del resto de dominios.
- En una imagen, sonido o vídeo cargado, añada un archivo de política URL en el servidor del archivo cargado. Este archivo de política debe conceder acceso al dominio del archivo SWF que intenta llamar a los métodos `BitmapData.draw()`, `BitmapData.drawWithQuality()` o `SoundMixer.computeSpectrum()` para extraer datos del archivo. El método `drawWithQuality` está disponible en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores.

En las siguientes secciones se proporcionan detalles sobre el acceso a datos de mapa de bits, sonido y vídeo.

Acceso a datos de mapa de bits

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Los métodos `draw()` y `drawWithQuality()` (Flash Player 11.3; AIR 3.3) de un objeto BitmapData permiten dibujar en el objeto BitmapData los píxeles de cualquier objeto de visualización que se están mostrando. Podrían ser los píxeles de un objeto MovieClip, un objeto Bitmap o cualquier objeto de visualización. Se deben cumplir las siguientes condiciones para que estos métodos dibujen píxeles en el objeto BitmapData:

- En el caso de un objeto de origen que no sea un mapa de bits cargado, el objeto de origen y (en el caso de un objeto Sprite o MovieClip) todos sus objetos secundarios deben proceder del mismo dominio que el objeto que realiza la llamada al método draw(), o bien deben incluirse en un archivo SWF que sea accesible para el llamador mediante una llamada al método `Security.allowDomain()`.
- En el caso de un objeto de origen de mapa de bits cargado, el objeto de origen debe proceder del mismo dominio que el objeto que realiza la llamada al método draw() o su servidor de origen debe incluir un archivo de política URL que conceda permiso al dominio que realiza la llamada.

Si no se cumplen estas condiciones, se emite una excepción SecurityError.

Cuando se carga una imagen mediante el método `load()` de la clase Loader, se puede especificar un parámetro `context`, que es un objeto LoaderContext. Si se establece la propiedad `checkPolicyFile` del objeto LoaderContext en `true`, Flash Player comprueba si hay un archivo de política entre dominios en el servidor desde el cual se carga la imagen. Si hay un archivo de política y este admite el dominio del archivo SWF que realiza la carga, el archivo podrá acceder a los datos del objeto Bitmap; en caso contrario, se denegará el acceso.

También se puede especificar una propiedad `checkPolicyFile` en una imagen cargada a través de una etiqueta `<img>` en un campo de texto. Para obtener información más detallada, consulte "Carga de archivos SWF e imágenes mediante la etiqueta <img> de un campo de texto" en la página 1107.

Acceso a datos de sonido

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Las siguientes API de ActionScript 3.0 relacionadas con el sonido tienen restricciones de seguridad:

- El método `SoundMixer.computeSpectrum()`: siempre se permite en el código que se ejecuta en el mismo entorno limitado de seguridad que el archivo de sonido. Para el código de otros entornos limitados existen comprobaciones de seguridad.
- El método `SoundMixer.stopAll()`: siempre se permite en el código que se ejecuta en el mismo entorno limitado de seguridad que el archivo de sonido. Para los archivos de otros entornos limitados existen comprobaciones de seguridad.
- La propiedad `id3` de la clase Sound: siempre se permite en archivos SWF que se encuentran en el mismo entorno limitado de seguridad que el archivo de sonido. Para el código de otros entornos limitados existen comprobaciones de seguridad.

Cada sonido tiene dos tipos de entornos limitados asociados, que son un entorno limitado de contenido y un entorno limitado de propietario:

- El dominio de origen del sonido determina el entorno limitado de contenido que, a su vez, determina si los datos pueden extraerse del sonido a través de la propiedad `id3` del sonido y a través del método `SoundMixer.computeSpectrum()`.

- El objeto que inició el sonido que se reproduce determina el entorno limitado de propietario que, a su vez, determina si el sonido puede detenerse a través del método `SoundMixer.stopAll()`.

Cuando se carga el sonido mediante el método `load()` de la clase Sound, se puede especificar un parámetro `context`, que es un objeto SoundLoaderContext. Si establece la propiedad `checkPolicyFile` del objeto SoundLoaderContext como `true`, el motor de ejecución comprobará la existencia de un archivo de política URL en el servidor desde el que se carga el sonido. Si hay un archivo de política y este admite el dominio del código que realiza la carga, el código podrá acceder a la propiedad `id` del objeto Sound; en caso contrario, no será posible. Además, si se establece un valor de la propiedad `checkPolicyFile`, se puede activar el método `SoundMixer.computeSpectrum()` para sonidos cargados.

Se puede utilizar el método `SoundMixer.areSoundsInaccessible()` para saber si una llamada al método `SoundMixer.stopAll()` no va a detener todos los sonidos porque el que realiza la llamada no puede acceder al entorno limitado de uno o varios propietarios de sonidos.

Llamar al método `SoundMixer.stopAll()` permite detener estos sonidos cuyo entorno limitado de propietario es el mismo que el que realiza la llamada a `stopAll()`. También detiene los sonidos que han empezado a reproducirse porque unos archivos SWF realizaron una llamada al método `Security.allowDomain()` para permitir el acceso por parte del dominio del archivo SWF que realiza la llamada al método `stopAll()`. Los demás sonidos no se detienen y su presencia puede revelarse mediante una llamada al método `SoundMixer.areSoundsInaccessible()`.

Para llamar al método `computeSpectrum()`, es necesario que cada sonido que se esté reproduciendo se encuentre en el mismo entorno limitado que el objeto que realiza la llamada al método o que proceda de un origen que haya concedido permiso al entorno limitado del que realiza la llamada; en caso contrario, se emite una excepción SecurityError. En el caso de los sonidos cargados desde sonidos incorporados en una biblioteca de un archivo SWF, el permiso se concede a través de una llamada al método `Security.allowDomain()` en el archivo SWF cargado. En el caso de los sonidos que no proceden de archivos SWF (procedentes de archivos MP3 cargados o archivos de vídeo) un archivo de política URL en el servidor de origen concede acceso a los datos de los medios cargados.

Para obtener más información, consulte "Controles de autor (desarrollador)" en la página 1099 y "Controles de sitio web (archivos de política)" en la página 1095.

Para acceder a datos de sonido desde flujos RTMP, debe permitir el acceso en el servidor. Utilice la propiedad `Client.audioSampleAccess` de ActionScript de servidor para permitir acceso a directorios específicos de Flash Media Server. Para obtener más información, consulte la referencia del lenguaje ActionScript de Flash Media Server.

Acceso a datos de vídeo

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se puede utilizar el método `BitmapData.draw()` o `BitmapData.drawWithQuality()` para capturar los datos de píxeles del fotograma actual de un vídeo. (El método `drawWithQuality` está disponible en Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores.)

Hay dos tipos distintos de vídeo:

- Transmisión de vídeo en RTMP desde Flash Media Server
- Vídeo progresivo, que se carga desde un archivo FLV o F4V

Para utilizar los métodos de dibujo y extraer gráficos de flujos RTMP en tiempo de ejecución, debe permitir el acceso en el servidor. Utilice la propiedad `Client.videoSampleAccess` de ActionScript de servidor para permitir el acceso a directorios específicos de Flash Media Server. Para obtener más información, consulte la referencia del lenguaje ActionScript de Flash Media Server.

Cuando se llama al método de dibujo con vídeo progresivo como valor del parámetro `source`, quien realiza la llamada al método debe encontrarse en el mismo entorno limitado que el archivo FLV, o el servidor del archivo FLV debe tener un archivo de política que conceda permiso al dominio del archivo SWF que realiza la llamada. Para solicitar la descarga del archivo de política, debe establecerse la propiedad `checkPolicyFile` del objeto NetStream en `true`.

Carga de datos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El contenido de Flash Player y de AIR puede intercambiar datos con servidores. La carga de datos es una operación distinta de la carga de medios, ya que la información cargada aparece directamente como objetos de programación en lugar de mostrarse como medios. Generalmente, el contenido puede cargar datos desde el mismo dominio que el que los originó. No obstante, el contenido requerir archivos de política para cargar datos desde otros dominios (consulte "Controles de sitio web (archivos de política)" en la página 1095).

Nota: *el contenido que se ejecuta en el entorno limitado de la aplicación de AIR nunca se sirve desde un dominio remoto (a no ser que el desarrollador importe intencionadamente contenido remoto en el entorno limitado de la aplicación), por lo que no puede participar en este tipo de ataques que quedan protegidos por los archivos de política. El contenido de AIR en el entorno limitado de la aplicación no está limitado a la carga de datos en sus archivos de política. Sin embargo, el contenido de AIR en otros entornos limitados está sujeto a las limitaciones aquí descritas.*

Uso de URLLoader y URLStream

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Se pueden cargar datos como, por ejemplo, un archivo XML o un archivo de texto. Los métodos `load()` de las clases URLLoader y URLStream se rigen por los permisos del archivo de política URL.

Si utiliza el método `load()` para cargar contenido desde un dominio distinto al dominio desde el cual llama el método, el motor de ejecución busca un archivo de política en el servidor de los activos cargados. Si hay un archivo de política y concede acceso al dominio del contenido que realiza la carga, se pueden cargar los datos.

Conexión a sockets

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

De forma predeterminada, el motor de ejecución busca un archivo de política de socket disponible en el puerto 843. Tal y como sucede con los archivos de política URL, este archivo se denomina*archivo maestro de política*.

Cuando se introdujeron por primera vez los archivos de política en Flash Player 6, no se admitían los archivos de política de socket. Las conexiones a servidores de socket se autorizaban a través de un archivo de política desde la ubicación predeterminada en un servidor HTTP del puerto 80 del mismo host que el servidor de socket. Flash Player 9 todavía admite esta capacidad, pero Flash Player 10 no. En Flash Player 10, solo los archivos de política de socket pueden autorizar las conexiones de socket.

Al igual que los archivos de política URL, los archivos de política de socket admiten una sentencia de metapolítica que especifica los puertos que pueden servir los archivos de política. Sin embargo, en lugar de "master-only", la metapolítica predeterminada de los archivos de política de socket es "all". Es decir, a no ser que el archivo maestro de política especifique una configuración más restrictiva, Flash Player asumirá que cualquier socket del host puede servir un archivo de política de socket.

De forma predeterminada, el acceso a conexiones de socket y socket XML está desactivado, incluso si el socket al que se conecta se encuentra en el dominio al que pertenece el archivo SWF. El acceso a nivel de socket es posible si se sirve un archivo de política de socket desde cualquiera de las ubicaciones siguientes:

- Puerto 843 (la ubicación del archivo maestro de política)
- El mismo puerto que la conexión de socket principal
- un puerto diferente a la conexión de socket principal

De forma predeterminada, Flash Player busca un archivo de política de socket en el puerto 843 y en el mismo puerto que utiliza la conexión de socket principal. Para servir un archivo de política de socket desde un puerto diferente, el archivo SWF debe llamar a `Security.loadPolicyFile()`.

Los archivos de política de socket presentan la misma sintaxis que los archivos de política URL, salvo por el hecho de que también deben especificar los puertos a los que concede acceso. Cuando un archivo de política de socket procede de un número de puerto inferior a 1024, dicho archivo puede conceder acceso a cualquier puerto; cuando un archivo de política procede del puerto 1024 o superior, solo puede conceder acceso a otros puertos 1024 y superiores. Los puertos permitidos se especifican en el atributo `to-ports` de la etiqueta `<allow-access-from>`. Se aceptan como valor los números de puerto únicos, los intervalos de puertos y los comodines.

A continuación se muestra un ejemplo de un archivo de política de socket:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

Para recuperar un archivo de política de socket del puerto 843 o del mismo puerto que utiliza la conexión de socket principal, llame al método `Socket.connect()` o `XMLSocket.connect()`. Flash Player comprueba en primer lugar la presencia del archivo maestro de política en el puerto 843. Si encuentra alguno, comprobará si este contiene una sentencia de metapolítica que prohíba los archivos de política de socket en el puerto de destino. Si el acceso no está prohibido, Flash Player buscará en primer lugar la sentencia `allow-access-from` en el archivo maestro de política. Si no encuentra ninguno, entonces buscará un archivo de política de socket en el mismo puerto que utiliza la conexión de socket principal.

Para recuperar un archivo de política de socket de una ubicación diferente, primero llame al método `Security.loadPolicyFile()` con la sintaxis especial `"xmlsocket"`, como en el ejemplo siguiente:

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Llame al método `Security.loadPolicyFile()` antes de llamar al método `Socket.connect()` o `XMLSocket.connect()`. Flash Player espera entonces hasta completar la solicitud del archivo de política antes de decidir si permite o no la conexión principal. Sin embargo, si el archivo maestro de política especifica que la ubicación de destino no puede servir archivos de política, la llamada a `loadPolicyFile()` no será efectiva, incluso si hay un archivo de política en dicha ubicación.

Si se implementa un servidor de socket y se necesita proporcionar un archivo de política de socket, se debe decidir entre proporcionar el archivo de política a través del mismo puerto que acepta conexiones principales o utilizar otro puerto. En cualquier caso, para poder enviar una respuesta, el servidor deberá esperar a la primera transmisión de su cliente.

Cuando Flash Player solicita un archivo de política, siempre transmite la siguiente cadena en cuanto se establece una conexión:

```
<policy-file-request/>
```

Cuando el servidor recibe esta cadena, puede transmitir el archivo de política. La petición de Flash Player siempre termina con un byte null, y la respuesta del servidor debe terminar con otro byte del mismo tipo.

No cabe esperar que se pueda reutilizar la misma conexión para una solicitud de archivo de política y una conexión principal; cierre la conexión después de transmitir el archivo de política. De lo contrario, Flash Player cierra la conexión del archivo de política antes de volver a conectar para configurar la conexión principal.

Protección de datos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para proteger los datos frente a pérdidas y modificaciones mientras viajan por Internet, puede utilizar los protocolos TLS o SSL en el servidor desde el que se originan los datos. Seguidamente, puede conectarse al servidor mediante el protocolo HTTPS.

En aplicaciones creadas para AIR 2 o posterior, también puede proteger las comunicaciones de socket TCP. La clase SecureSocket permite iniciar una conexión de socket con un servidor de sockets que use TLS (versión 1) o SSL (versión 4).

Envío de datos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El envío de datos se produce cuando el código envía los datos a un servidor o recursos. En envío de datos se permite siempre para el contenido de un dominio de red. Un archivo SWF local puede enviar datos a direcciones de la red únicamente si se encuentra en el entorno limitado local de confianza, en el entorno limitado local con acceso a la red o en un entorno limitado de la aplicación de AIR. Para obtener más información, consulte "Entornos limitados locales" en la página 1088.

Se puede utilizar la función `flash.net.sendToURL()` para enviar datos a un URL. Otros métodos también envían peticiones a URL. Algunos de estos métodos son los métodos de carga como `Loader.load()` y `Sound.load()`, y los métodos de carga de datos como `URLLoader.load()` y `URLStream.load()`.

Carga y descarga de archivos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

El método `FileReference.upload()` inicia la carga de un archivo seleccionado por un usuario en un servidor remoto. Se debe llamar al método `FileReference.browse()` o `FileReferenceList.browse()` antes de llamar al método `FileReference.upload()`.

El código que inicia el método `FileReference.browse()` o `FileReferenceList.browse()` solo puede llamarse como respuesta a un evento de ratón o de teclado. Si se llama en otras situaciones, Flash Player 10 y versiones posteriores emitirán una excepción. No obstante, no se requiere ningún evento iniciado por el usuario para llamar a estos métodos desde el entorno limitado de la aplicación de AIR.

Al llamar al método `FileReference.download()`, se abre un cuadro de diálogo en el que el usuario puede descargar un archivo desde un servidor remoto.

Nota: *si el servidor requiere autenticación del usuario, solo los archivos que se ejecutan en un navegador, es decir que utilizan el plug-in de navegador o controles ActiveX, pueden mostrar un cuadro de diálogo para pedir al usuario un nombre de usuario y una contraseña para la autenticación, y solo para las descargas. Flash Player no permite realizar cargas en servidores que requieran autenticación de usuario.*

Las cargas y descargas no se permiten si el archivo SWF que realiza la llamada se encuentra en el entorno limitado local con sistema de archivos.

De forma predeterminada, un archivo SWF no puede realizar cargas ni descargas en un servidor ajeno. Un archivo SWF puede realizar cargas y descargas en otro servidor, si dicho servidor proporciona un archivo de política que conceda permiso al dominio del archivo SWF que realiza la llamada.

Carga de contenido incorporado de archivos SWF importados en un dominio de seguridad

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando se carga un archivo SWF, se puede establecer el parámetro `context` del método `load()` del objeto Loader que se utiliza para cargar el archivo. Este parámetro es un objeto LoaderContext. Si se establece la propiedad `securityDomain` de este objeto LoaderContext como `Security.currentDomain`, Flash Player comprueba si hay un archivo de política URL en el servidor del archivo SWF cargado. Si hay un archivo de política y concede acceso al dominio del archivo SWF que realiza la carga, se puede cargar el archivo SWF como medios importados. De este modo, el archivo que realiza carga puede obtener acceso a los objetos de la biblioteca del archivo SWF.

Otra forma de que un archivo SWF pueda acceder a las clases de los archivos SWF cargados de otros entornos limitados de seguridad es hacer que el archivo SWF cargado llame al método `Security.allowDomain()` para conceder acceso al dominio del archivo SWF que realiza la llamada. Se puede añadir la llamada al método `Security.allowDomain()` al método constructor de la clase principal del archivo SWF cargado y luego hacer que el archivo SWF que realiza la carga añada un detector de eventos para responder al evento `init` distribuido por la propiedad `contentLoaderInfo` del objeto Loader. Cuando se distribuye este evento, el archivo SWF cargado ha llamado al método `Security.allowDomain()` en el método constructor y las clases del archivo SWF cargado están disponibles para el archivo SWF que realiza la carga. El archivo SWF de carga puede recuperar clases del archivo SWF cargado si llama a `Loader.contentLoaderInfo.applicationDomain.getDefinition()` o a Loader.contentLoaderInfo.applicationDomain.getQualifiedDefinitionNames()(Flash Player 11.3 y versiones posteriores; AIR 3.3 y versiones posteriores).

Trabajo con contenido heredado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

En Flash Player 6, el dominio utilizado para determinada configuración de Flash Player se basaba en el fragmento final del dominio del archivo SWF. Esta configuración incluye la configuración para permisos de cámaras y micrófonos, las cuotas de almacenamiento y el almacenamiento de objetos compartidos persistentes.

Si el dominio de un archivo SWF contiene más de dos segmentos, como en www.example.com, se quita el primer segmento del dominio (www) y se utiliza el fragmento restante. De este modo, en Flash Player 6, tanto www.example.com como store.example.com utilizan example.com como dominio para esta configuración. De forma análoga, tanto www.example.co.uk como store.example.co.uk utilizan example.co.uk como dominio para esta configuración. Esto puede originar problemas cuando archivos SWF de dominios no relacionados, como example1.co.uk y example2.co.uk, tienen acceso a los mismos objetos compartidos.

En Flash Player 7 y posterior, la configuración del reproductor se elige de forma predeterminada según el dominio exacto de un archivo SWF. Por ejemplo, un archivo SWF del dominio www.example.com utilizará la configuración del reproductor para www.example.com y un archivo SWF del dominio store.example.com utilizará la configuración del reproductor específica para store.example.com.

En un archivo SWF escrito en ActionScript 3.0, cuando `Security.exactSettings` se establece en `true` (valor predeterminado), Flash Player utiliza dominios exactos para la configuración del reproductor. Cuando se establece en `false`, Flash Player utiliza la configuración de dominio utilizada en Flash Player 6. Si se cambia el valor predeterminado de `exactSettings`, debe hacerse antes de que se produzca algún evento que requiera que Flash Player elija la configuración del reproductor, por ejemplo, al utilizar una cámara o micrófono, o al recuperar un objeto compartido persistente.

Si publica un archivo SWF de la versión 6 y crea objetos compartidos persistentes a partir de él, para recuperar estos objetos desde un archivo SWF que utilice ActionScript 3.0, debe establecer `Security.exactSettings` en `false` antes de llamar a `SharedObject.getLocal()`.

Configuración de permisos de LocalConnection

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La clase LocalConnection permite enviar mensajes entre aplicaciones de Flash Player o de AIR. Los objetos LocalConnection se pueden comunicar únicamente en contenido de Flash Player o de AIR que se ejecute en el equipo cliente, pero puede hacerlo en distintas aplicaciones; por ejemplo, un archivo SWF que se ejecute en un navegador, un archivo SWF que se ejecute en un proyector y una aplicación de AIR pueden comunicarse gracias a la clase LocalConnection.

En cada comunicación LocalConnection, hay un emisor y un detector. De forma predeterminada, Flash Player permite la comunicación de LocalConnection entre código ejecutándose en el mismo dominio. En el caso de código ejecutándose en distintos entornos limitados, el detector debe dar permiso al emisor a través del método `LocalConnection.allowDomain()`. La cadena que pase como argumento al método `LocalConnection.allowDomain()` puede contener cualquiera de los elementos siguientes: nombres de dominio exactos, direcciones IP y el operador comodín `*`.

El método `allowDomain()` ha cambiado de la forma que tenía en ActionScript 1.0 y 2.0. En estas versiones anteriores `allowDomain()` era un método callback implementado por el usuario. En ActionScript 3.0, `allowDomain()` es un método incorporado de la clase LocalConnection que puede recibir llamadas. Con este cambio, el funcionamiento de `allowDomain()` es muy similar al de `Security.allowDomain()`.

Un archivo SWF puede utilizar la propiedad `domain` de la clase LocalConnection para determinar su dominio.

Control del acceso URL saliente

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

La creación de scripts y el acceso URL salientes (a través de URL HTTP, mailto:, etc.) se realizan a través de las siguientes API de

- La función `flash.system.fscommand()`
- El método `ExternalInterface.call()`
- La función `flash.net.navigateToURL()`

Para contenido cargado desde el sistema de archivos local, las llamadas a estos métodos son correctas únicamente si el código y la página web que lo contiene (si existe) se encuentran en el entorno limitado local de confianza o el de la aplicación de AIR. Las llamadas a estos métodos no se realizan correctamente si el contenido se encuentra en el entorno limitado local con acceso a la red o en el entorno limitado local con sistema de archivos.

En el caso de contenido que no se carga de forma local, todas estas API se pueden comunicar con la página web en la que están incorporadas, en función del valor del parámetro AllowScriptAccess que se describe a continuación. La función `flash.net.navigateToURL()` tiene la capacidad adicional de comunicarse con cualquier ventana o fotograma de navegador abierto, no solo con la página en la que está incorporado el archivo SWF. Para obtener más información sobre esta funcionalidad, consulte "Uso de la función navigateToURL()" en la página 1120.

El parámetro `AllowScriptAccess` del código HTML que carga un archivo SWF controla la capacidad de realizar un acceso a URL saliente desde un archivo SWF. Defina este parámetro dentro de la etiqueta PARAM o EMBED. Si no se define ningún valor para `AllowScriptAccess`, el archivo SWF y la página HTML solo se podrán comunicar si ambos se encuentran dentro del mismo dominio.

El parámetro `AllowScriptAccess` puede tener uno de los tres valores siguientes: `"always"`, `"sameDomain"` o `"never"`.

- Cuando `AllowScriptAccess` se establece como `"always"`, el archivo SWF se puede comunicar con la página HTML que lo incorpora, incluso cuando su dominio y el de la página son diferentes.
- Cuando `AllowScriptAccess` se establece como `"sameDomain"`, el archivo SWF se puede comunicar con la página HTML que lo incorpora solo cuando su dominio y el de la página son iguales. Este es el valor predeterminado de `AllowScriptAccess`. Utilice esta configuración, o no defina un valor para `AllowScriptAccess`, para evitar que un archivo SWF alojado en un dominio acceda a un script de una página HTML perteneciente a otro dominio.
- Cuando `AllowScriptAccess` se establece como `"never"`, el archivo SWF no se puede comunicar con ninguna página HTML. El uso de este valor se ha desfasado desde la versión de Adobe Flash CS4 Professional. No se recomienda y no debe ser necesario si no se proporcionan archivos SWF que no son de confianza desde su propio dominio. Si es preciso servir archivos SWF que no sean de confianza, Adobe recomienda crear un subdominio independiente y colocar en él todo ese contenido.

Este es un ejemplo de configuración de la etiqueta `AllowScriptAccess` en una página HTML para permitir el acceso URL saliente a un dominio diferente:

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0'
height='100%' width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

Uso de la función navigateToURL()

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Además de la configuración de seguridad especificada por el parámetro `allowScriptAccess` descrito anteriormente, la función `navigateToURL()` tiene un segundo parámetro opcional: `target`. El parámetro `target` se puede utilizar para especificar el nombre de una ventana o fotograma HTML al que enviar la petición de URL. Las restricciones de seguridad adicional se aplican a dichas peticiones y varían en función de si `navigateToURL()` se utiliza como una sentencia de creación de scripts o como una sentencia de no creación de scripts.

En el caso de las sentencias de creación de scripts, como `navigateToURL("javascript: alert('Hello from Flash Player.')")`, es necesario tener en cuenta las reglas siguientes.

- Si el archivo SWF es un archivo de confianza local, la solicitud funcionará correctamente.
- Si el destino es la página HTML en la que se incorpora el archivo SWF, se aplicarán las reglas `allowScriptAccess` descritas anteriormente.
- Si el destino tiene contenido cargado del dominio al que pertenece el archivo SWF, la solicitud funcionará correctamente.
- Si el destino tiene contenido cargado de un dominio distinto al que pertenece el archivo SWF, y no se cumple ninguna de las condiciones anteriores, la solicitud no funcionará correctamente.

En el caso de las sentencias que crean scripts (como HTTP, HTTPS y `mailto:`), la solicitud no funcionará correctamente si se cumplen todas las condiciones siguientes:

- El destino es una de las palabras clave especiales `"_top"` o `"_parent"`, y
- el archivo SWF se encuentra en una página web alojada en un dominio diferente, y
- el archivo SWF está incorporado con un valor para `allowScriptAccess` que no es `"always"`.

Para obtener más información

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Para obtener información sobre el acceso URL saliente, consulte las siguientes entradas en *Referencia de ActionScript 3.0 para la plataforma de Adobe Flash:*

- La función flash.system.fscommand()
- El método call() de la clase ExternalInterface
- La función `flash.net.navigateToURL()`

Objetos compartidos

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Flash Player proporciona la capacidad de utilizar *objetos compartidos*, que son objetos de ActionScript que persisten fuera de un archivo SWF, ya sea localmente en el sistema de archivos de un usuario o remotamente en un servidor RTMP. Los objetos compartidos, al igual que otros medios de Flash Player, se dividen en entornos limitados de seguridad. Sin embargo, el modelo de entorno limitado de los objetos compartidos es algo distinto, ya que los objetos compartidos no son recursos a los que se puede tener acceso más allá de los límites de los dominios. Por el contrario,

los objetos compartidos siempre se recuperan de un almacén de objetos compartidos que es específico del dominio de cada archivo SWF que llama a los métodos de la clase SharedObject. Los almacenes de objetos compartidos suelen ser incluso más específicos que los dominios de los archivos SWF: de forma predeterminada, cada archivo SWF utiliza un almacén de objetos compartidos específico de su URL de origen completa. Para obtener más información sobre los objetos compartidos, consulte "Objetos compartidos" en la página 731.

Un archivo SWF puede utilizar el parámetro `localPath` de los métodos `SharedObject.getLocal()` y `SharedObject.getRemote()` para usar un almacén de objetos compartidos asociado solamente a una parte de su URL. De este modo, el archivo SWF puede permitir el uso compartido con otros archivos SWF de otros URL. Incluso si se pasa el valor `'/'` para el parámetro `localPath`, se sigue designando un almacén de objetos compartidos específico de su propio dominio.

Los usuarios pueden restringir el acceso a objetos compartidos utilizando el cuadro de diálogo Configuración de Flash Player o el Administrador de configuración. De forma predeterminada, pueden crearse objetos compartidos de un máximo de 100 KB de datos por dominio. Los usuarios con derechos administrativos y los usuarios pueden también limitar la capacidad de escribir en el sistema de archivos. Para obtener más información, consulte "Controles de administrador" en la página 1092 y "Controles de usuario" en la página 1094.

Para especificar que un objeto compartido sea seguro, se debe establecer el valor `true` para el parámetro `secure` del método `SharedObject.getLocal()` o del método `SharedObject.getRemote()`. Tenga en cuenta las siguientes cuestiones relativas al parámetro `secure`:

- Si el valor de este parámetro es `true`, Flash Player crea un nuevo objeto compartido seguro u obtiene una referencia a un objeto compartido seguro existente. Solo pueden leer o escribir en este objeto compartido seguro archivos SWF enviados a través de HTTPS que llamen a `SharedObject.getLocal()` con el parámetro `secure` definido como `true`.
- Si el parámetro se establece en `false`, Flash Player crea un nuevo objeto compartido o bien obtiene una referencia a un objeto compartido existente que pueda leerse y escribirse con archivos SWF enviados mediante conexiones que no son HTTPS.

Si el archivo SWF que realiza la llamada no procede de un URL HTTPS, al especificar el valor `true` para el parámetro `secure` del método `SharedObject.getLocal()` o del método `SharedObject.getRemote()`, se emitirá una excepción SecurityError.

La elección de un almacén de objetos compartidos se basa en el URL de origen de un archivo SWF. Esto es cierto incluso en las dos situaciones en las que un archivo SWF no se origina en un simple URL: carga de importación y carga dinámica. La carga de importación hace referencia a la situación en la que se carga un archivo SWF con la propiedad `LoaderContext.securityDomain` establecida en `SecurityDomain.currentDomain`. En esta situación, el archivo SWF cargado tendrá un pseudoURL que empieza por el dominio del archivo SWF de carga, seguido del URL de origen real. La carga dinámica se refiere a la carga de un archivo SWF a través del método `Loader.loadBytes()`. En esta situación, el archivo SWF cargado tendrá un pseudoURL que empieza por el URL completo del archivo SWF que realiza la carga, seguido de un ID de entero. Tanto en la carga de importación como en la carga dinámica, el pseudoURL de un archivo SWF puede examinarse con la propiedad `LoaderInfo.url`. El pseudoURL se trata del mismo modo que un URL real, en lo que se refiere a la elección de un almacén de objetos compartidos. Se puede especificar un parámetro `localPath` de objeto compartido que utilice el pseudoURL parcialmente o en su totalidad.

Los usuarios y los administradores pueden optar por desactivar la utilización de *objetos compartidos de terceros*. Es la utilización de objetos compartidos por parte de cualquier archivo SWF que esté ejecutándose en un navegador web, cuando el URL de origen de dicho archivo SWF procede de un dominio distinto al del URL que se muestra en la barra de direcciones del navegador. Los usuarios y administradores pueden optar por desactivar la utilización de objetos compartidos de terceros por motivos de privacidad, con el fin de evitar el seguimiento entre dominios. Para evitar esta restricción, quizá desee asegurarse de que cualquier archivo SWF que utilice objetos compartidos solo se cargue en las

estructuras de página HTML que garantizan que el archivo SWF procede del mismo dominio que se muestra en la barra de direcciones del navegador. Cuando se intentan utilizar objetos compartidos desde un archivo SWF de terceros y la utilización de objetos compartidos de terceros está desactivada, los métodos `SharedObject.getLocal()` y `SharedObject.getRemote()` devuelven `null`. Para obtener más información, consulte www.adobe.com/es/products/flashplayer/articles/thirdpartylso.

Acceso a la cámara, el micrófono, el portapapeles, el ratón y el teclado

Flash Player 9 y posterior, Adobe AIR 1.0 y posterior

Cuando un archivo SWF intenta acceder a la cámara o micrófono de un usuario a través de los métodos `Camera.get()` o `Microphone.get()`, Flash Player muestra un cuadro de diálogo Privacidad que permite al usuario autorizar o denegar el acceso a la cámara o al micrófono. El usuario y el usuario administrador también pueden desactivar el acceso a la cámara para cada sitio o de forma global, a través de controles en el archivo mms.cfg, la interfaz de usuario Configuración y el Administrador de configuración (consulte "Controles de administrador" en la página 1092 y "Controles de usuario" en la página 1094). Con restricciones de usuario, cada uno de los métodos `Camera.get()` y `Microphone.get()` devuelve un valor `null`. Se puede utilizar la propiedad `Capabilities.avHardwareDisable` para determinar si el administrador ha prohibido (`true`) o permitido (`false`) el acceso a la cámara y el micrófono.

El método `System.setClipboard()` permite que un archivo SWF sustituya el contenido del portapapeles por una cadena de texto normal. Esto no supone ningún riesgo de seguridad. Para protegerse del riesgo que supone cortar o copiar contraseñas y otros datos confidenciales en el portapapeles, no existe ningún método `getClipboard()`.

Una aplicación que se ejecute en Flash Player solo puede controlar los eventos de teclado y ratón que sucedan en su ámbito. El contenido que se ejecuta en Flash Player no puede detectar eventos de teclado o ratón en otra aplicación.

Seguridad en AIR

Adobe AIR 1.0 y posterior

Aspectos básicos de la seguridad de AIR

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR se ejecutan con las mismas restricciones de seguridad que las aplicaciones nativas. En general, las aplicaciones de AIR, como las aplicaciones nativas, tienen un amplio acceso a las capacidades del sistema operativo como, por ejemplo, leer y escribir archivos, iniciar aplicaciones, dibujar en pantalla y la comunicación con la red. Las restricciones del sistema operativo que se aplican a las aplicaciones nativas, como privilegios específicos del usuario, se aplican de igual manera a las aplicaciones de AIR.

Aunque el modelo de seguridad de Adobe® AIR® es una evolución del modelo de seguridad de Adobe® Flash® Player, el contrato de seguridad es diferente del contrato de seguridad que se aplica al contenido en un navegador. Este contrato ofrece a los desarrolladores un medio seguro de funcionalidad más amplia para experiencias enriquecedoras con libertades que serían inapropiadas para una aplicación basada en un navegador.

Las aplicaciones de AIR están escritas usando ya sea código de bytes compilado (contenido SWF) o script interpretado (JavaScript, HTML) para que el motor de ejecución proporcione la administración de la memoria. Esto reduce las posibilidades de que las aplicaciones de AIR se vean afectadas por las vulnerabilidades relacionadas con la administración de la memoria, como desbordes de búfer y corrupción de memoria. Estas son algunas de las vulnerabilidades más comunes que afectan a las aplicaciones de escritorio escritas en código nativo.

Instalación y actualizaciones

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR se distribuyen mediante archivos de instalación de AIR que utilizan la extensión `air` o a través de los instaladores nativos, que usan la extensión y el formato de archivo de la plataforma nativa. Por ejemplo, el formato del archivo de instalación nativo de Windows es un archivo EXE y para Android el formato nativo es un archivo APK.

Cuando se instala Adobe AIR y se abre un archivo de instalación de AIR, el motor de ejecución de AIR administra el proceso de instalación. Cuando se utiliza un instalador nativo, el sistema operativo administra el proceso de instalación.

Nota: *los desarrolladores pueden especificar una versión y un nombre de aplicación y un origen de editor cuando se utiliza el formato de archivo de AIR, pero no se puede modificar el flujo de trabajo de instalación de la aplicación inicial. Esta restricción es una ventaja para los usuarios porque todas las aplicaciones de AIR comparten un procedimiento de instalación seguro, optimizado y coherente administrado por el motor de ejecución. Si es necesario personalizar la aplicación, se puede hacer cuando se ejecuta la aplicación por primera vez.*

Ubicación de la instalación del motor de ejecución

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR que utilizan el formato de archivo de AIR primero requieren la instalación del motor de ejecución en el equipo del usuario, al igual que los archivos SWF primero requieren la instalación del plug-in de navegador de Flash Player.

El motor de ejecución se instala en la siguiente ubicación en los equipos de escritorio:

- Mac OS: `/Library/Frameworks/`
- Windows: `C:\Archivos de programa\Archivos comunes\Adobe AIR`
- Linux: `/opt/Adobe AIR/`

En Mac OS, para instalar una versión actualizada de una aplicación, el usuario debe contar con adecuados privilegios del sistema para instalar en el directorio de la aplicación. En Windows y Linux, el usuario debe disponer de privilegios de administrador.

Nota: *en iOS, el motor de ejecución de AIR no se instala de forma independiente; cada aplicación de AIR es una aplicación de contenido propio.*

El motor de ejecución se puede instalar de dos maneras: usando la función de instalación integrada (instalando directamente desde un navegador web) o a través de una instalación manual. Las aplicaciones de AIR empaquetadas como instaladores nativos también pueden instalar el motor de ejecución de AIR como parte del proceso normal de instalación de la aplicación. (Distribuir el motor de ejecución de AIR de esta forma requiere la firma de un acuerdo de redistribución con Adobe.)

Instalación integrada (motor de ejecución y aplicación)

Adobe AIR 1.0 y posterior

La función de instalación integrada proporciona a los desarrolladores una experiencia de instalación optimizada para los usuarios que aún no tienen instalado Adobe AIR. En el método de instalación integrada, el desarrollador crea un archivo SWF que presenta la aplicación para instalar. Cuando un usuario hace clic en el archivo SWF para instalar la aplicación, el archivo SWF intenta detectar el motor de ejecución. Si el motor de ejecución no se puede detectar se instala, y el motor de ejecución se activa de forma inmediata con el proceso de instalación para la aplicación del desarrollador.

Instalación manual

Adobe AIR 1.0 y posterior

Como alternativa, el usuario puede descargar e instalar manualmente el motor de ejecución antes de abrir un archivo de AIR. Entonces el desarrollador puede distribuir un archivo de AIR mediante diferentes medios (por ejemplo, a través de correo electrónico o un vínculo HTML en un sitio web). Cuando se abre el archivo de AIR, el motor de ejecución comienza el proceso de instalación de la aplicación.

Flujo de instalación de la aplicación

Adobe AIR 1.0 y posterior

El modelo de seguridad de AIR permite que los usuarios decidan si quieren instalar una aplicación de AIR. La instalación de AIR proporciona varias mejoras con respecto a las tecnologías de instalación de las aplicaciones nativas que ayuda a los usuarios a la hora de tomar decisiones:

- El motor de ejecución proporciona una experiencia de instalación coherente en todos los sistemas operativos, aun cuando una aplicación de AIR se instala desde un vínculo en un navegador web. La mayoría de las experiencias de instalación de las aplicaciones nativas dependen del navegador u otra aplicación para proporcionar información de seguridad, si la hay.
- La instalación de la aplicación de AIR identifica el origen de la aplicación y la información sobre los privilegios disponibles para la aplicación (si el usuario permite continuar la instalación).
- El motor de ejecución administra el proceso de instalación de una aplicación de AIR. Una aplicación de AIR no puede manipular el proceso de instalación que utiliza el motor de ejecución.

En general, los usuarios no deberían instalar ninguna aplicación de escritorio que proviene de un origen que no conocen o que no se puede verificar. La comprobación de veracidad sobre la seguridad de las aplicaciones nativas es igual para las aplicaciones de AIR así como para otras aplicaciones que se instalan.

Destino de la aplicación

Adobe AIR 1.0 y posterior

El directorio de instalación se puede establecer usando una de las siguientes dos opciones:

1. El usuario personaliza el destino durante la instalación. La aplicación se instala donde lo especifica el usuario.
2. Si el usuario no cambia el destino de la instalación, la aplicación se instala en la ruta predeterminada como lo determina el motor de ejecución:
   - Mac OS: `~/Aplicaciones/`
   - Windows XP y anterior: `C:\Archivos de programa\`

- Windows Vista: `~/Aplicaciones/`
- Linux: /opt/

Si el desarrollador especifica un parámetro `installFolder` en el archivo descriptor de la aplicación, la aplicación se instala en una ruta secundaria de este directorio.

Sistema de archivos de AIR

Adobe AIR 1.0 y posterior

El proceso de instalación para las aplicaciones de AIR copia todos los archivos que el desarrollador ha incluido en el archivo de instalación de AIR en el equipo local del usuario. La aplicación instalada se compone de:

- Windows: un directorio que contiene todos los archivos incluidos en el archivo de instalación de AIR. Asimismo, el motor de ejecución crea un archivo exe durante la instalación de la aplicación de AIR.
- Linux: un directorio que contiene todos los archivos incluidos en el archivo de instalación de AIR. Asimismo, el motor de ejecución crea un archivo bin durante la instalación de la aplicación de AIR.
- Mac OS: un archivo `app` que contiene todo el contenido del archivo de instalación de AIR. Se puede examinar usando la opción "Show Package Contents" en Finder. El motor de ejecución crea este archivo app como parte de la instalación de la aplicación de AIR.

Una aplicación de AIR se ejecuta del siguiente modo:

- Windows: ejecutando el archivo .exe en la carpeta de instalación, o un método abreviado que corresponde a este archivo (como un método abreviado en el menú Inicio o en el escritorio)
- Linux: iniciando el archivo .bin en la carpeta de instalación, seleccionando la aplicación en el menú de aplicaciones o realizando la ejecución a partir de un alias o acceso directo en el escritorio.
- Mac OS: ejecutando el archivo .app o un alias que apunta al mismo.

Asimismo, el sistema de archivos de la aplicación incluye subdirectorios relacionados con la función de la aplicación. Por ejemplo, la información escrita en el almacenamiento local cifrado se guarda en un subdirectorio en un directorio nombrado a partir del identificador de la aplicación.

Almacenamiento de la aplicación de AIR

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR tienen privilegios para escribir en cualquier ubicación en el disco duro del usuario; sin embargo, se recomienda que los desarrolladores utilicen la ruta `app-storage:/` para el almacenamiento local relacionado con su aplicación. Los archivos que se escriben a `app-storage:/` desde una aplicación se encuentran en una ubicación estándar dependiendo del sistema operativo del usuario:

- En Mac OS: el directorio de almacenamiento de una aplicación varía según la versión de AIR:
  - **AIR 3.2 y versiones anteriores** - `<appData>/<appId>/Local Store/` donde `<appData>` es la “carpeta de preferencias” del usuario, normalmente: `/Users/<user>/Library/Preferences`
  - **AIR 3.3 y versiones posteriores** - `<ruta de acceso>/Library/Application Support/<appID>/Local Store`, donde `<ruta de acceso>` es `/Users/<user>/Library/Containers/<bundle-id>/Data` (dentro de entorno limitado de seguridad) o `/Users/<user>` (fuera de entorno limitado de seguridad)
- En Windows: el directorio de almacenamiento de una aplicación es `<appData>\<appId>\Local Store\` , donde `<appData>` es la “Carpeta especial” CSIDL_APPDATA del usuario, normalmente: `C:\Documents and Settings\<user>\Application Data`

- En Linux: `<appData>/<appID>/Local Store/`, donde `<appData>` es `/home/<user>/.appdata`

Puede acceder al directorio de almacenamiento de la aplicación a través de la propiedad `air.File.applicationStorageDirectory`. Puede acceder al contenido usando el método `resolvePath()` de la clase File. Para más información, consulte "Trabajo con el sistema de archivos" en la página 679.

Actualización de Adobe AIR

Adobe AIR 1.0 y posterior

Cuando el usuario instala una aplicación de AIR que requiere una versión actualizada del motor de ejecución, el tiempo de ejecución automáticamente instala la actualización del motor de ejecución requerida.

Para actualizar el motor de ejecución, el usuario debe tener privilegios administrativos para el equipo.

Actualización de aplicaciones de AIR

Adobe AIR 1.0 y posterior

El desarrollo y la implementación de actualizaciones de software es uno de los desafíos más grandes de seguridad que enfrentan las aplicaciones de código nativo. La API de AIR proporciona un mecanismo para mejorar esto: se puede invocar el método `Updater.update()` durante el inicio para verificar la ubicación remota de un archivo de AIR. Si se requiere una actualización, el archivo de AIR se descarga, se instala y la aplicación se reinicia. Los desarrolladores pueden usar esta clase no solo para proporcionar nueva funcionalidad sino también para responder a las vulnerabilidades potenciales de seguridad.

La clase Updater solo se puede utilizar para actualizar las aplicaciones distribuidas como archivos de AIR. Las aplicaciones distribuidas como aplicaciones nativas, deben utilizar los recursos de actualización, si existe alguno, del sistema operativo nativo.

Nota: *los desarrolladores pueden especificar la versión de una aplicación configurando la propiedad versionNumber del archivo descriptor de la aplicación.*

Desinstalación de una aplicación de AIR

Adobe AIR 1.0 y posterior

Al quitar una aplicación de AIR se quitan todos los archivos en el directorio de la aplicación. Sin embargo, no se quitan todos los archivos que la aplicación haya escrito fuera del directorio de la aplicación. Cuando se quitan las aplicaciones de AIR no se deshacen los cambios que la aplicación de AIR ha realizado en los archivos fuera del directorio de la aplicación.

Parámetros del registro de Windows para administradores

Adobe AIR 1.0 y posterior

En Windows, los administradores pueden configurar un equipo para impedir (o permitir) la instalación de una aplicación de AIR o actualizaciones del motor de ejecución. Estos parámetros están dentro del registro de Windows bajo la siguiente clave: HKLM\Software\Policies\Adobe\AIR. Incluyen lo siguiente:

Parámetro de registro	Descripción
AppInstallDisabled	Especifica que se permite la instalación y desinstalación de la aplicación de AIR. Configurada en 0 para "autorizado", y en 1 para "no autorizado".
UntrustedAppInstallDisabled	Especifica que se permite la instalación de aplicaciones de AIR que no son de confianza (aplicaciones que no incluyen un certificado de confianza). Configurada en 0 para "autorizado", y en 1 para "no autorizado".
UpdateDisabled	Especifica que se permite la actualización del motor de ejecución, ya sea como una tarea en segundo plano o como parte de una instalación explícita. Configurada en 0 para "autorizado", y en 1 para "no autorizado".

Seguridad HTML en Adobe AIR

Adobe AIR 1.0 y posterior

Este tema describe la arquitectura de seguridad HTML de AIR y cómo utilizar iframes, fotogramas y el puente de entorno limitado para configurar aplicaciones basadas en HTML e integrar con seguridad el contenido HTML en aplicaciones basadas en SWF.

El motor de ejecución impone reglas y proporciona mecanismos para superar posibles vulnerabilidades de seguridad en HTML y JavaScript. Se imponen las mismas reglas independientemente si la aplicación está principalmente escrita en JavaScript o si se carga el contenido HTML y JavaScript en una aplicación basada en SWF. El contenido en el entorno limitado de la aplicación y en el entorno limitado de seguridad que no pertenece a la aplicación tiene diferentes privilegios. Cuando se carga contenido en un iframe o en un fotograma, el motor de ejecución proporciona un mecanismo seguro de *puente de entorno limitado* que permite que el contenido en el fotograma o iframe se comunique de forma segura en el entorno limitado de seguridad de la aplicación.

El SDK de AIR proporciona tres clases para representar contenido HTML.

La clase HTMLLoader proporciona una estrecha integración entre el código JavaScript y las API de AIR.

La clase StageWebView es una clase de representación HTML y cuenta con una integración muy limitada con la aplicación host de AIR. El contenido cargado mediante la clase StageWebView nunca se sitúa en el entorno limitado de seguridad de la aplicación y no puede acceder a los datos ni llamar a las funciones en la aplicación host de AIR. En las plataformas de escritorio, la clase StageWebView emplea el motor incorporado HTML de AIR, basado en Webkit, que también utiliza la clase HTMLLoader. En las plataformas móviles, la clase StageWebView utiliza el control HTML proporcionado por el sistema operativo. Por lo tanto, en las plataformas móviles la clase StageWebView presenta las mismas vulnerabilidades y factores de seguridad que el navegador web del sistema.

La clase TextField puede mostrar cadenas de texto HTML. JavaScript no se puede ejecutar, pero el texto puede incluir vínculos e imágenes cargadas de forma externa.

Para más información, consulte "Cómo evitar errores de JavaScript relacionados con la seguridad" en la página 1021.

Información general sobre la configuración de la aplicación basada en HTML

Adobe AIR 1.0 y posterior

Los fotogramas e iframes proporcionan una estructura conveniente para organizar contenido HTML en AIR. Los fotogramas proporcionan un medio para mantener la persistencia de datos y para trabajar de forma segura con el contenido remoto.

Dado que HTML en AIR conserva la organización normal basada en la página, el entorno HTML se regenera completamente si el fotograma superior del contenido HTML "navega" a una página diferente. Se pueden usar fotogramas e iframes para conservar la persistencia de datos en AIR, al igual que en una aplicación web ejecutándose en un navegador. Defina los objetos de la aplicación principal en el fotograma superior y persistirán siempre que no permita que el fotograma navegue a una nueva página. Use iframes o fotogramas secundarios para cargar y visualizar las partes transitorias de la aplicación. (Hay varias maneras de mantener la persistencia de datos que se pueden usar además o en lugar de los fotogramas. Estas incluyen cookies, objetos compartidos locales, almacenamiento en archivo local, el almacén de archivo cifrado y el almacenamiento local en base de datos.)

Debido a que HTML en AIR conserva su línea difuminada normal entre el código ejecutable y los datos, AIR sitúa el contenido en el marco superior del entorno HTML en el entorno limitado de la aplicación. Tras el evento de página `load`, AIR restringe todas las operaciones como, por ejemplo, `eval()`, que puede convertir una cadena de texto en un objeto ejecutable. Esta restricción se impone aun cuando una aplicación no carga el contenido remoto. Para permitir que el contenido HTML ejecute estas operaciones restringidas, debe utilizar fotogramas o iframes para situar el contenido en un entorno limitado ajeno a la aplicación. (Puede ser necesaria la ejecución de contenido en un marco secundario de entorno limitado cuando se utilizan algunos marcos de aplicación de JavaScript que confían en la función `eval()`.) Para obtener una lista completa de las restricciones en JavaScript en el entorno limitado de la aplicación, consulte "Restricciones de código del contenido en entornos limitados diferentes" en la página 1130.

Dado que HTML en AIR conserva la habilidad de cargar contenido remoto posiblemente no seguro, AIR impone una política de mismo origen que evita que el contenido en un dominio interactúe con el contenido en otro. Para permitir la interacción entre el contenido de aplicación y el contenido en otro dominio, se puede configurar un puente que actúe como la interfaz entre un fotograma principal y uno secundario.

Configuración de una relación de entorno limitado principal y secundario

Adobe AIR 1.0 y posterior

AIR añade los atributos `sandboxRoot` y `documentRoot` al los elementos frame e iframe HTML. Estos atributos permiten tratar al contenido de aplicación como si proviniera de otro dominio:

Atributo	Descripción
sandboxRoot	La URL que se debe utilizar para determinar el entorno limitado y dominio donde colocar el contenido del fotograma. Se deben utilizar los esquemas de URL `file:`, `http:` o `https:`.
documentRoot	La URL desde donde cargar el contenido del fotograma. Se deben utilizar los esquemas URL `file:`, `app:` o `app-storage:`.

En el siguiente ejemplo se asigna el contenido instalado en el subdirectorio del entorno limitado de la aplicación para que se ejecute en el entorno limitado remoto y en el dominio www.example.com:

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

Configuración de un puente entre fotogramas principales y secundarios en diferentes entornos limitados o dominios

Adobe AIR 1.0 y posterior

AIR añade las propiedades `childSandboxBridge` y `parentSandboxBridge` al objeto `window` de cualquier fotograma secundario. Estas propiedades permiten definir puentes para que actúen como interfaces entre un fotograma principal y uno secundario. Cada puente va en una dirección:

`childSandboxBridge` — La propiedad `childSandboxBridge` permite que el fotograma secundario exponga una interfaz al contenido en el fotograma principal. Para exponer una interfaz, se debe establecer la propiedad `childSandbox` a una función u objeto en el fotograma secundario. Entonces puede acceder al objeto o función desde el contenido en el fotograma principal. En el siguiente ejemplo se muestra el modo en que un script ejecutándose en un fotograma secundario puede exponer al fotograma principal un objeto que contiene una función y una propiedad:

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Si este contenido secundario se encuentra en un iframe y se le asignó un `ID` de `"elemento secundario"`, se puede acceder a la interfaz desde el contenido principal leyendo la propiedad `childSandboxBridge` del fotograma:

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

`parentSandboxBridge` — La propiedad `parentSandboxBridge` permite que el fotograma principal exponga una interfaz al contenido en el fotograma secundario. Para exponer una interfaz, se debe establecer la propiedad `parentSandbox` del fotograma secundario a una función u objeto en el fotograma principal. Entonces puede acceder al objeto o función desde el contenido en el fotograma secundario. En el siguiente ejemplo se muestra el modo en que un script ejecutándose en un fotograma principal puede exponer al fotograma secundario un objeto que contiene una función de guardar:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

Usando esta interfaz, el contenido en el fotograma secundario podría guardar texto en un archivo denominado save.txt. Sin embargo, no tendría ningún otro acceso al sistema de archivos. En general, el contenido de aplicación debe exponer una interfaz lo más limitada posible a otros entornos limitados. El contenido secundario puede llamar a la función save de la siguiente manera:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

Si el contenido secundario intenta definir una propiedad del objeto `parentSandboxBridge`, el motor de ejecución emite una excepción SecurityError. Si el contenido principal intenta definir una propiedad del objeto `childSandboxBridge`, el motor de ejecución emite una excepción SecurityError.

Restricciones de código del contenido en entornos limitados diferentes

Adobe AIR 1.0 y posterior

Tal y como se describe en la introducción a este tema, "Seguridad HTML en Adobe AIR" en la página 1127, el motor de ejecución impone reglas y proporciona mecanismos para superar posibles vulnerabilidades de seguridad en HTML y JavaScript. Este tema lista dichas restricciones. Si el código intenta llamar a estas API restringidas, el motor de ejecución emite un error con el mensaje "Adobe AIR runtime security violation for JavaScript code in the application security sandbox" (Infracción de seguridad del motor de ejecución de Adobe AIR para el código JavaScript en el entorno limitado de seguridad de la aplicación).

Para más información, consulte "Cómo evitar errores de JavaScript relacionados con la seguridad" en la página 1021.

Restricciones en el uso de la función eval() JavaScript y técnicas similares

Adobe AIR 1.0 y posterior

Para el contenido HTML en el entorno limitado de seguridad de la aplicación, hay limitaciones en el uso de API que pueden transformar dinámicamente cadenas en código ejecutable después de cargar el código (después de que el evento `onload`del elemento `body` se haya distribuido y la función del controlador `onload` haya terminado de ejecutarse). Esta limitación es para evitar que la aplicación accidentalmente inserte (y ejecute) código desde orígenes que no pertenecen a la aplicación (como dominios de red potencialmente inseguros).

Por ejemplo, si la aplicación usa datos de cadena de un origen remoto para escribir en la propiedad innerHTML de un elemento DOM, la cadena puede incluir código ejecutable (JavaScript) que podría realizar operaciones no seguras. Sin embargo, mientras que el contenido se está cargando, no hay riesgo de que se inserten cadenas remotas en el DOM.

Una restricción es el uso de la función `eval()` JavaScript. Una vez que se carga el código en el entorno limitado de la aplicación y después de procesar el controlador de eventos onload, solo se puede usar la función `eval()` en maneras limitadas. Las siguientes reglas se aplican al uso de la función `eval()` *después* de que se cargue el código del entorno limitado de seguridad de la aplicación:

- Se permiten expresiones relacionadas con literales. Por ejemplo:

```
eval("null");
eval("3 + .14");
eval("'foo'");
```

- Se permiten literales de objetos, como en el siguiente ejemplo:

```
{ prop1: val1, prop2: val2 }
```

- Se *prohíben* los setter/getters literales de objetos, como en el siguiente ejemplo:

```
{ get prop1() { ... }, set prop1(v) { ... } }
```

- Se permiten literales de conjuntos, como en el siguiente ejemplo:

```
[ val1, val2, val3 ]
```

- Se *prohíben* las expresiones relacionadas con las lecturas de propiedades, como en el siguiente ejemplo:

```
a.b.c
```

- La invocación de función está *prohibida.*
- Las definiciones de funciones están *prohibidas.*
- La configuración de cualquier propiedad está *prohibida.*
- Los literales de funciones están *prohibidos.*

Sin embargo, mientras se carga el código, antes del evento `onload` y durante la ejecución de la función del controlador de eventos `onload`, estas restricciones no se aplican al contenido en el entorno limitado de seguridad de la aplicación.

Por ejemplo, después de cargar el código, el código siguiente hace que el motor de ejecución emita una excepción:

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

El código generado dinámicamente, como el que se crea cuando se llama a la función `eval()`, presenta un riego de seguridad si se permite dentro del entorno limitado de la aplicación. Por ejemplo, una aplicación puede accidentalmente ejecutar una cadena cargada de un dominio de red y dicha cadena puede tener código malintencionado. Por ejemplo, puede ser código para eliminar o alterar archivos en el equipo del usuario. O puede ser código que informa a un dominio de red que no es de confianza sobre el contenido de un archivo local.

A continuación se incluyen formas de generar código dinámico:

- Llamar a la función `eval()`.
- Usar las propiedades `innerHTML` o funciones DOM para insertar etiquetas de script que cargan un script fuera del directorio de la aplicación.
- Usar las propiedades `innerHTML` o funciones DOM para insertar etiquetas de script que tienen código insertado (en lugar de cargar un script a través del atributo `src`).
- Configurar el atributo `src` para que las etiquetas `script` carguen un archivo JavaScript que se encuentra fuera del directorio de la aplicación.
- Usar el esquema `javascript` de URL (como en `href="javascript:alert('Test')"`).
- Usar la función `setInterval()` o `setTimout()` donde el primer parámetro (definiendo la función para que se ejecute de forma asíncrona) es una cadena (para evaluar) en vez de un nombre de función (como en `setTimeout('x = 4', 1000)`).
- Llamar a `document.write()` o `document.writeln()`.

El código en el entorno limitado de seguridad de la aplicación solo puede utilizar estos métodos mientras se carga el contenido.

Estas restricciones *no* impiden el uso de `eval()` con los literales de objetos JSON. Esto permite que el contenido de aplicación trabaje con la biblioteca JSON JavaScript. Sin embargo, está restringido en el uso de código JSON sobrecargado (con controladores de eventos).

Para otros marcos de Ajax y bibliotecas de código JavaScript, verifique si el código en el marco o biblioteca funciona dentro de estas restricciones en código generado dinámicamente. Si no funcionan, incluya cualquier contenido que usa el marco o la biblioteca en un entorno limitado de seguridad que no pertenece a la aplicación. Para obtener más información, consulte "Restricciones para JavaScript dentro de AIR" en la página 1090 y "Uso de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación" en la página 1139. Adobe mantiene una lista de marcos de Ajax que admiten el entorno limitado de seguridad de aplicación, en http://www.adobe.com/es/products/air/develop/ajax/features/.

A diferencia del contenido en el entorno limitado de seguridad de la aplicación, el contenido JavaScript en un entorno limitado de seguridad que no pertenece a la aplicación *puede* llamar a la función `eval()` para que ejecute dinámicamente el código generado en cualquier momento.

Restricciones para acceder a las API de AIR (para entornos limitados ajenos a la aplicación)

Adobe AIR 1.0 y posterior

El código JavaScript en un entorno limitado que no pertenece a la aplicación no tiene acceso al objeto `window.runtime` y como tal este código no puede ejecutar las API de AIR. Si el contenido en un entorno limitado de seguridad que no pertenece a la aplicación llama al siguiente código, la aplicación emite una excepción TypeError:

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

El tipo de excepción es TypeError (valor no definido), porque el contenido en el entorno limitado que no pertenece a la aplicación no reconoce el objeto `window.runtime`, por lo tanto se lo considera un valor no definido.

Se puede exponer la funcionalidad del motor de ejecución al contenido en un entorno limitado que no pertenece a la aplicación usando un puente de script. Para más información, consulte "Uso de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación" en la página 1139.

Restricciones para usar llamadas XMLHttpRequest

Adobe AIR 1.0 y posterior

El contenido HTML en el entorno limitado de seguridad de la aplicación no puede utilizar métodos XMLHttpRequest sincrónicos para cargar datos desde fuera del entorno limitado de la aplicación mientras se está cargando el contenido HTML y durante el evento `onLoad`.

De forma predeterminada, el contenido HTML en entornos limitados de seguridad que no pertenecen a la aplicación no pueden usar el objeto JavaScript XMLHttpRequest para cargar datos de dominios que no sea el dominio que llama a la petición. Una etiqueta `frame` o `iframe` puede incluir el atributo `allowcrosscomainxhr`. La configuración de este atributo a cualquier valor que no sea null permite que el contenido en el fotograma o iframe use el objeto XMLHttpRequest JavaScript para cargar datos de dominios que no sea el dominio del código que llama a la petición:

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

Para más información, consulte "Creación de scripts entre contenido en diferentes dominios" en la página 1133.

Restricciones para cargar elementos CSS, frame, iframe e img (para el contenido en entornos limitados que no pertenecen a la aplicación)

Adobe AIR 1.0 y posterior

El contenido HTML en los entornos limitados de seguridad remotos (red) solo puede cargar contenido CSS, `frame`, `iframe` e `img` de entornos limitados remotos (de URL de red).

El contenido HTML en los entornos limitados local con sistema de archivos, local con acceso a la red o local de confianza solo pueden cargar contenido CSS, frame, iframe e `img` de entornos limitados locales (no de aplicaciones o entornos limitados remotos).

Restricciones para llamar al método window.open() JavaScript

Adobe AIR 1.0 y posterior

Si una ventana que se crea a través de una llamada al método `window.open()` JavaScript muestra contenido de un entorno limitado de seguridad que no pertenece a la aplicación, el título de la ventana comienza con el título de la ventana principal (de inicio), seguido por dos puntos (:). No se puede utilizar el código para trasladar esa porción del título de la ventana fuera de la pantalla.

El contenido en entornos limitados de seguridad que no pertenecen a la aplicación solo pueden llamar correctamente al método `window.open()` JavaScript en respuesta a un evento activado por el ratón o interacción del teclado. Esto impide que el contenido que no pertenece a la aplicación cree ventanas que se pueden utilizar maliciosamente (por ejemplo, para ataques de suplantación de identidad (phishing). Asimismo, el controlador de eventos para el evento de ratón o teclado no puede definir el método `window.open()` para ejecutarse después de una demora (por ejemplo llamando a la función `setTimeout()`).

El contenido en entornos limitados remotos (red) solo puede utilizar el método `window.open()` para abrir el contenido en entornos limitados de red remotos. No puede utilizar el método `window.open()` para abrir contenido de la aplicación o entornos limitados locales.

El contenido en los entornos limitados local con sistema de archivos, local con acceso a la red o local de confianza (consulte "Entornos limitados de seguridad" en la página 1087) solo puede usar el método `window.open()` para abrir contenido en entornos limitados locales. No puede utilizar el método `window.open()` para abrir contenido de la aplicación o entornos limitados remotos.

Errores al llamar a código restringido

Adobe AIR 1.0 y posterior

Si se llama al código que no se puede utilizar en un entorno limitado debido a estas restricciones de seguridad, el motor de ejecución distribuye un error de JavaScript: "Adobe AIR runtime security violation for JavaScript code in the application security sandbox" (Infracción de seguridad del motor de ejecución de Adobe AIR para el código JavaScript en el entorno limitado de seguridad de la aplicación).

Para más información, consulte "Cómo evitar errores de JavaScript relacionados con la seguridad" en la página 1021.

Protección del entorno limitado al cargar contenido HTML de una cadena

Adobe AIR 1.0 y posterior

El método `loadString()` de la clase HTMLLoader permite crear contenido HTML en tiempo de ejecución. No obstante, los datos que se utilizan como contenido HTML pueden verse dañados si la información se carga desde un origen de Internet no seguro. Por este motivo, de forma predeterminada, el código HTML creado con el método`loadString()` no se sitúa en el entorno limitado de la aplicación y no tiene acceso a las API de AIR. Sin embargo, puede establecer la propiedad `placeLoadStringContentInApplicationSandbox` de un objeto HTMLLoader en true para que sitúe HTML creado utilizando el método `loadString()` en el entorno limitado de la aplicación. Para obtener más información, consulte "Carga del contenido HTML desde una cadena" en la página 1020.

Creación de scripts entre contenido en diferentes dominios

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR tienen privilegios especiales cuando se instalan. Es fundamental que los mismos privilegios no se filtren a otro contenido, incluyendo archivos remotos y archivos locales que no son parte de la aplicación.

Puente de entorno limitado de AIR

Adobe AIR 1.0 y posterior

Normalmente, el contenido de otros dominios no pueden llamar a scripts en otros dominios. Para proteger las aplicaciones de AIR de filtraciones accidentales de control o información privilegiada, se implementan las siguientes restricciones en el contenido en el entorno limitado de seguridad `aplicación` (contenido instalado con la aplicación).

- El código en el entorno limitado de seguridad de la aplicación no se permite en otros entornos limitados llamando al método `Security.allowDomain()`. Con la llamada a este método desde el entorno limitado de seguridad de la aplicación se emitirá un error.
- Se impide la importación de contenido que no pertenece a la aplicación en el entorno limitado de la aplicación si se define la propiedad `LoaderContext.securityDomain` o `LoaderContext.applicationDomain`.

Aún hay casos donde la aplicación de AIR principal requiere que el contenido de un dominio remoto tenga acceso controlado a los scripts en la aplicación de AIR principal o viceversa. Para lograr esto, el motor de ejecución proporciona un mecanismo de *puente de entorno limitado*, que actúa como una puerta de enlace entre los dos entornos limitados. Un puente de entorno limitado puede proporcionar interacción explícita entre entornos limitados de seguridad de aplicación y remotos.

El puente de entorno limitado expone dos objetos al que pueden acceder ambos scripts cargados y los que se están cargando:

- El objeto `parentSandboxBridge` permite que el contenido que se está cargando exponga las propiedades y funciones a los scripts en el contenido cargado.
- El objeto `childSandboxBridge` permite que el contenido cargado exponga las propiedades y funciones a los scripts en el contenido que se está cargando.

Los objetos expuestos a través del puente de entorno limitado se pasan por valor y no por referencia. Todos los datos se serializan. Esto significa que los objetos expuestos por un lado del puente no pueden ser definidos por el otro lado del puente y que los objetos expuestos no tienen tipo. Asimismo, solo se pueden exponer funciones y objetos simples; no se pueden exponer objetos complejos.

Si el contenido secundario intenta definir una propiedad del objeto parentSandboxBridge, el motor de ejecución emite una excepción SecurityError. De igual manera, si el contenido principal intenta definir una propiedad del objeto childSandboxBridge, el motor de ejecución emite una excepción SecurityError.

Ejemplo de puente de entorno limitado (SWF)

Adobe AIR 1.0 y posterior

Supongamos que una aplicación de almacén de música de AIR quiere permitir que los archivos SWF remotos transmitan el precio de los álbumes, pero no quiere que el archivo SWF remoto divulgue si el precio es un precio de oferta. Para hacer esto, una clase StoreAPI proporciona un método para adquirir el precio, pero oculta el precio de oferta. Una instancia de esta clase StoreAPI se asigna a continuación a la propiedad`parentSandboxBridge` del objeto LoaderInfo del objeto Loader que carga el SWF remoto.

El siguiente es el código para almacén de música de AIR:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
title="Music Store" creationComplete="initApp()">
    <mx:Script>
        import flash.display.Loader;
        import flash.net.URLRequest;

        private var child:Loader;
        private var isSale:Boolean = false;

        private function initApp():void {
            var request:URLRequest =
                    new URLRequest("http://[www.yourdomain.com]/PriceQuoter.swf")

            child = new Loader();
            child.contentLoaderInfo.parentSandboxBridge = new StoreAPI(this);
            child.load(request);
            container.addChild(child);
        }
        public function getRegularAlbumPrice():String {
            return "$11.99";
        }
        public function getSaleAlbumPrice():String {
            return "$9.99";
        }
        public function getAlbumPrice():String {
            if(isSale) {
                return getSaleAlbumPrice();
            }
            else {
                return getRegularAlbumPrice();
            }
        }
    </mx:Script>
    <mx:UIComponent id="container" />
</mx:WindowedApplication>
```

El objeto StoreAPI llama a la aplicación principal para recuperar el precio normal del álbum, pero devuelve “No disponible” cuando se llama al método `getSaleAlbumPrice()`. El siguiente código define la clase StoreAPI:

```
public class StoreAPI
{
    private static var musicStore:Object;

    public function StoreAPI(musicStore:Object)
    {
        this.musicStore = musicStore;
    }

    public function getRegularAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }

    public function getSaleAlbumPrice():String {
        return "Not available";
    }

    public function getAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }
}
```

El siguiente código representa un ejemplo del archivo SWF PriceQuoter que informa sobre el precio de venta pero no puede informar sobre el pecio de oferta:

```
package
{
    import flash.display.Sprite;
    import flash.system.Security;
    import flash.text.*;

    public class PriceQuoter extends Sprite
    {
        private var storeRequester:Object;

        public function PriceQuoter() {
            trace("Initializing child SWF");
            trace("Child sandbox: " + Security.sandboxType);
            storeRequester = loaderInfo.parentSandboxBridge;

            var tf:TextField = new TextField();
            tf.autoSize = TextFieldAutoSize.LEFT;
            addChild(tf);

            tf.appendText("Store price of album is: " + storeRequester.getAlbumPrice());
            tf.appendText("\n");
            tf.appendText("Sale price of album is: " + storeRequester.getSaleAlbumPrice());
        }
    }
}
```

Ejemplo de puente de entorno limitado (HTML)

Adobe AIR 1.0 y posterior

En el contenido HTML, las propiedades `parentSandboxBridge` y `childSandboxBridge` se añaden al objeto window JavaScript de un documento secundario. Para obtener un ejemplo de cómo configurar las funciones del puente en el contenido HTML, consulte "Definición de una interfaz de puente de entorno limitado" en la página 1040.

Limitación de la exposición de API

Adobe AIR 1.0 y posterior

Cuando se exponen puentes de entorno limitado, es importante exponer las API de nivel alto que limitan el grado del uso indebido. Tenga en cuenta que el contenido que está llamando a la implementación del puente puede dejar el sistema vulnerable (por ejemplo, a través de una inyección de código). Entonces, por ejemplo, la exposición de un método `readFile(path:String)` (que lee los contenidos de un archivo arbitrario) a través de un puente es vulnerable a ser utilizado incorrectamente. Sería mejor exponer una API `readApplicationSetting()` que no usa una ruta y lee un archivo específico. El método más semántico limita el daño que puede hacer una aplicación una vez que parte de la misma está en peligro.

Más temas de ayuda

"Uso de scripts entre contenidos en diferentes entornos limitados de seguridad" en la página 1038

"Entorno limitado de la aplicación de AIR" en la página 1089

Escritura en el disco

Adobe AIR 1.0 y posterior

Las aplicaciones que se ejecutan en un navegador web solo tienen una interacción limitada con el sistema de archivos local del usuario. Los navegadores web implementan políticas de seguridad que aseguran que el equipo de un usuario no corre riesgo como resultado de cargar contenido web. Por ejemplo, los archivos SWF que se ejecutan a través de Flash Player en un navegador no pueden interactuar directamente con los archivos que ya se encuentran en el equipo del usuario. Los objetos y cookies compartidos se pueden escribir en el disco de un equipo de usuario para mantener las preferencias de usuario y otros datos, pero este es el límite de la interacción con el sistema de archivos. Dado que las aplicaciones de AIR se instalan de forma nativa, tienen un diferente contrato de seguridad, uno que incluye la capacidad de leer y escribir en todo el sistema de archivos local.

Esta libertad conlleva gran responsabilidad para los desarrolladores. Las inseguridades de aplicación accidentales ponen en peligro no solo la funcionalidad de la aplicación, sino también la integridad del equipo del usuario. Por esta razón, los desarrolladores deben leer "Prácticas recomendadas de seguridad para desarrolladores" en la página 1140.

Los desarrolladores de AIR puede acceder y escribir archivos en el sistema de archivos local usando varias convenciones de esquemas de URL:

Esquema de URL	Descripción
app:/	Un alias al directorio de la aplicación. Los archivos que se acceden desde esta ruta se les asignan un entorno limitado de la aplicación y tienen privilegios completos concedidos por el motor de ejecución.
app-storage:/	Un alias al directorio de almacenamiento local, estandarizado por el motor de ejecución. Los archivos que se acceden desde esta ruta se les asigna un entorno limitado que no pertenece a la aplicación.
file:///	Un alias que representa la raíz del disco duro del usuario. Un archivo que se accede desde esta ruta se le asigna un entorno limitado de la aplicación si el archivo existe en el directorio de la aplicación, de lo contrario se le asigna un entorno limitado que no pertenece a la aplicación.

Nota: *las aplicaciones de AIR no pueden modificar el contenido usando el esquema de URL app:. Asimismo, el directorio de la aplicación puede leer solo debido a la configuración del administrador.*

A menos que haya restricciones de administrador en el equipo del usuario, las aplicaciones de AIR tienen el privilegio de escribir en cualquier ubicación en el disco duro del usuario. Se recomienda que los desarrolladores usen la ruta `app-storage:/` para el almacenamiento local relacionado con su aplicación. Los archivos que se escriben en `app-storage:/` desde una aplicación se colocan en una ubicación estándar:

- En Mac OS: el directorio de almacenamiento de una aplicación es `<appData>/<appId>/Local Store/` donde `<appData>` es la carpeta de preferencias del usuario. Normalmente es `/Usuarios/<usuario>/Biblioteca/Preferencias`
- En Windows: el directorio de almacenamiento de una aplicación es `<appData>\<appId>\Local Store\` donde `<appData>` es la carpeta especial del usuario CSIDL_APPDATA. Normalmente es `C:\Documents and Settings\<Nombre de usuario>\Application Data`
- En Linux: `<appData>/<appID>/Local Store/`, donde `<appData>` es `/home/<user>/.appdata`

Si una aplicación está diseñada para interactuar con archivos existentes en el sistema de archivos del usuario, asegúrese de leer "Prácticas recomendadas de seguridad para desarrolladores" en la página 1140.

Cómo trabajar de forma segura con contenido que no es de confianza

Adobe AIR 1.0 y posterior

El contenido que no se asigna al entorno limitado de la aplicación puede proporcionar una funcionalidad adicional de uso de scripts para la aplicación pero solo si cumple con los criterios de seguridad del motor de ejecución. Este tema explica el contrato de seguridad de AIR con el contenido que no pertenece a la aplicación.

Security.allowDomain()

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR restringen el acceso al uso de scripts para el contenido que no pertenece a la aplicación más severamente que el plug-in de navegador de Flash Player restringe el acceso al uso de scripts para el contenido que no es de confianza. Por ejemplo, en Flash Player en el navegador, cuando un archivo SWF que se asigna al entorno limitado `local-trusted` llama al método `System.allowDomain()`, el acceso a la secuencia de comandos se concede a cualquier archivo SWF cargado desde el dominio especificado. El método equivalente no se permite por parte del contenido de `aplicación` en las aplicaciones de AIR, ya que concedería acceso peligroso al archivo que no pertenece a la aplicación en el sistema de archivos del usuario. Los archivos remotos no pueden acceder de forma directa al entorno limitado de la aplicación, independientemente de las llamadas al método `Security.allowDomain()`.

Uso de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación

Adobe AIR 1.0 y posterior

Las aplicaciones de AIR que usan script entre contenido de la aplicación y contenido que no pertenece a la aplicación tienen condiciones más complejas de seguridad. Los archivos que no están en el entorno limitado de la aplicación solo tienen permitido acceder a las propiedades y métodos de los archivos en el entorno limitado de la aplicación a través del uso de un puente de entorno limitado. Un puente de entorno limitado actúa como una puerta de enlace entre el contenido de aplicación y el contenido que no pertenece a la aplicación, proporcionando una interacción explícita entre los dos archivos. Cuando se usan correctamente, los puentes de entorno limitado brindan una capa adicional de seguridad, restringiendo el contenido que no pertenece a la aplicación de acceder a las referencias de objetos que forman parte del contenido de aplicación.

La ventaja de los puentes de entorno limitado se describe mejor en este ejemplo. Supongamos que una aplicación de almacén de música de AIR desea proporcionar una API a los anunciantes que quieren crear sus propios archivos SWF, con los que la aplicación de almacén puede comunicarse. El almacén desea proporcionar a los anunciantes métodos para buscar artistas y CD en el almacén, pero también quieren aislar algunos métodos y propiedades del archivo SWF de terceros por razones de seguridad.

Un puente de entorno limitado puede brindar esta funcionalidad. De forma predeterminada, el contenido cargado externamente en una aplicación de AIR durante el tiempo de ejecución no tiene acceso a ningún método ni propiedad en la aplicación principal. Con una implementación personalizada del puente de entorno limitado, un desarrollador puede proporcionar servicios al contenido remoto sin exponer estos métodos o propiedades. Considere al puente de entorno limitado como una vía entre el contenido de confianza y el contenido que no es de confianza, brindando comunicación entre el contenido cargado y el cargador sin exponer las referencias de objetos.

Para más información sobre cómo utilizar de forma segura los puentes de entorno limitado, consulte "Creación de scripts entre contenido en diferentes dominios" en la página 1133.

Protección contra la generación dinámica de contenido SWF no seguro

Adobe AIR 1.0 y posterior

El método `Loader.loadBytes()` proporciona un modo para que una aplicación genere contenido SWF desde un conjunto de bytes. Sin embargo, los ataques a los datos cargados desde orígenes remotos podrían causar un daño serio cuando se carga el contenido. Esto es cierto cuando se cargan datos en el entorno limitado de la aplicación, donde el contenido SWF generado puede acceder al conjunto entero de las API de AIR.

Existen usos legítimos para la utilización del método `loadBytes()` sin generar código SWF ejecutable. Por ejemplo, se puede usar el método `loadBytes()` para generar datos de imagen para controlar la coordinación de la visualización de la imagen. Asimismo, existen usos legítimos que *si* utilizan la ejecución del código, como la creación dinámica de contenido SWF para reproducción de audio. En AIR, de forma predeterminada, el método `loadBytes()` *no* permite cargar el contenido SWF; solo permite cargar contenido de imágenes. En AIR, la propiedad `loaderContext` del método `loadBytes()` tiene una propiedad `allowLoadBytesCodeExecution`, que se puede definir en `true` para permitir explícitamente que la aplicación use `loadBytes()` para cargar contenido SWF ejecutable. El siguiente código muestra la manera de usar esta función:

```
var loader:Loader = new Loader();
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.allowLoadBytesCodeExecution = true;
loader.loadBytes(bytes, loaderContext);
```

Si se llama a `loadBytes()` para cargar el contenido SWF y la propiedad `allowLoadBytesCodeExecution` del objeto LoaderContext está definida en `false` (el valor predeterminado), el objeto Loader emite una excepción SecurityError.

Nota: *es posible que esta API cambie en una futura versión de Adobe AIR. Cuando ocurra, tal vez tenga que recompilar el contenido que usa la propiedad* `allowLoadBytesCodeExecution` *de la clase LoaderContext.*

Prácticas recomendadas de seguridad para desarrolladores

Adobe AIR 1.0 y posterior

Aunque las aplicaciones de AIR se crean usando tecnologías web, es importante que los desarrolladores observen que no están trabajando dentro del entorno limitado de seguridad del navegador. Esto significa que es posible crear aplicaciones de AIR que pueden causar daño al sistema local, ya sea accidental o malintencionadamente. AIR intenta minimizar este riesgo, pero aún hay maneras donde se pueden introducir vulnerabilidades. Este tema abarca importantes situaciones potenciales no seguras.

Riesgos de importar archivos en un entorno limitado de seguridad de la aplicación

Adobe AIR 1.0 y posterior

Los archivos que existen en el directorio de la aplicación se asignan al entorno limitado de la aplicación y tienen privilegios completos del motor de ejecución. Se recomienda que las aplicaciones que escriben en el sistema de archivos local escriban en `app-storage:/`. Este directorio existe de forma separada de los archivos de la aplicación en el equipo del usuario, por ende los archivos no están asignados al entorno limitado de la aplicación y presentan un riesgo reducido de seguridad. Se recomienda que los desarrolladores consideren los siguientes puntos:

- Incluir un archivo en un archivo de AIR (en la aplicación instalada) solo si es necesario.
- Incluir un archivo de script en un archivo de AIR (en la aplicación instalada) solo si el comportamiento se comprende cabalmente y es de confianza.
- No escribir ni modificar el contenido en el directorio de la aplicación. El motor de ejecución impide que las aplicaciones escriban o modifiquen los archivos y directorios usando el esquema de URL `app:/` emitiendo una a excepción SecurityError.
- No utilizar datos de un origen de red como parámetros para los métodos de la API de AIR que pueden llevar a la ejecución de código. Esto incluye el uso del método `Loader.loadBytes()` y la función `eval()` JavaScript.

Riesgos de usar un origen externo para determinar rutas

Adobe AIR 1.0 y posterior

Una aplicación de AIR puede correr riesgos cuando se usan datos o contenido externo. Por esta razón, tenga cuidado cuando utiliza datos de la red o del sistema de archivos. La responsabilidad de confianza depende del desarrollador y las conexiones de red que realizan, pero la carga de datos externos es intrínsicamente peligrosa y no se debería usar para operaciones confidenciales. No se recomienda que los desarrolladores:

- Usen datos de un origen de red para determinar un nombre de archivo.
- Utilicen datos de un origen de red para construir una URL que usa la aplicación para enviar información privada.

Riesgos de usar, almacenar o transmitir credenciales no seguras

Adobe AIR 1.0 y posterior

El almacenamiento de credenciales de usuario en el sistema de archivos local del usuario intrínsicamente introduce el riesgo de que estas credenciales sean vulnerables. Se recomienda que los desarrolladores consideren los siguientes puntos:

- Si se deben almacenar las credenciales de forma local, se deben cifrar las credenciales cuando se escribe en el sistema de archivos local. El motor de ejecución proporciona un almacenamiento cifrado exclusivo para cada aplicación instalada, a través de la clase EncryptedLocalStore. Para obtener información, consulte "Almacenamiento local cifrado" en la página 740.
- No transmita credenciales de usuario no cifradas a un origen de red a no ser que dicho origen sea de confianza y que la transmisión utilice el protocolo HTTPS: o TLS.
- Nunca se debe especificar una contraseña predeterminada en la creación de una credencial, los usuarios deben crear las propias. Los usuarios que dejan una contraseña predeterminada exponen sus credenciales a un atacante que ya conoce la contraseña predeterminada.

Riesgos de un ataque de desactualización

Adobe AIR 1.0 y posterior

Durante la instalación de la aplicación, el motor de ejecución verifica que la versión de la aplicación no está actualmente instalada. Si una aplicación ya está instalada, el motor de ejecución compara la cadena de la versión con la versión que se está instalando. Si esta cadena es diferente, el usuario puede elegir actualizar la instalación. El motor de ejecución no garantiza que la versión recién instalada sea más nueva que la versión más antigua, solo que es diferente. Un atacante puede distribuir una versión anterior al usuario para evitar una debilidad de seguridad. Por esta razón, se recomienda que el desarrollador verifique las versiones cuando se ejecuta la aplicación. Es una buena idea que las aplicaciones verifiquen en la red las actualizaciones requeridas. De ese modo, aun si un atacante engaña al usuario para que ejecute una versión antigua, dicha versión antigua reconocerá que se debe actualizar. Asimismo, el uso de un esquema claro de versiones para la aplicación hace más difícil engañar a los usuarios para que instalen una versión desactualizada.

Firma de código

Adobe AIR 1.0 y posterior

Se requiere que todos los archivos de instalación de AIR tengan el código firmado. La firma de código es un proceso criptográfico de confirmar que el origen especificado del software es veraz. Las aplicaciones de AIR se pueden firmar utilizando un certificado emitido por una entidad de emisora de certificados externa (CA) o un certificado con firma automática de creación propia. Se recomienda un certificado comercial de una CA conocida ya que proporciona la garantía a los usuarios de que están instalando la aplicación real y no una falsificación. Sin embargo, los certificados con firma automática se pueden crear usando `adt` del SDK u usando Flash, Flash Builder u otra aplicación que utilice `adt` para la generación de certificados. Los certificados con firma automática no garantizan que la aplicación que se está instalando es original y se deben utilizar únicamente para probar una aplicación antes de su lanzamiento público.

Seguridad en dispositivos de Android

Adobe AIR 2.5 y posterior

En Android, al igual que sucede en todos los dispositivos informáticos, AIR se ajusta al modelo de seguridad nativo. Al mismo tiempo, AIR mantiene sus propias reglas de seguridad, que pretenden facilitar a los desarrolladores la escritura de aplicaciones seguras y conectadas a Internet.

Las aplicaciones de AIR en Android utilizan el formato de empaquetado de Android, por lo que la instalación falla en el modelo de seguridad de Android. El archivo de instalación de la aplicación de AIR no se utiliza.

El modelo de seguridad de Android presenta tres aspectos principales:

- Permisos
- Firmas de la aplicación
- IDs de usuario de la aplicación

Permisos de Android

Muchas funciones de Android están protegidas por el mecanismo de permisos del sistema operativo. Para poder utilizar una función protegida, el descriptor de la aplicación de AIR debe declarar que la aplicación requiere el permiso necesario. Cuando un usuario intenta instalar la aplicación, el sistema operativo Android muestra todos los permisos solicitados al usuario antes de que continúe la instalación.

La mayor parte de las aplicaciones de AIR necesitarán especificar los permisos de Android en el descriptor de la aplicación. De forma predeterminada, no se incluye ningún permiso. Los siguientes permisos son necesarios para las funciones protegidas de Android expuestas mediante el motor de ejecución de AIR:

ACCESS_COARSE_LOCATION Permite que la aplicación acceda a datos de ubicación de red WIFI y móvil a través de la clase Geolocation.

ACCESS_FINE_LOCATION Permite que la aplicación acceda a datos GPS a través de la clase Geolocation.

ACCESS_NETWORK_STATE y ACCESS_WIFI_STATE Permite que la aplicación acceda a la información de red a través de la clase NetworkInfo.

CAMERA Permite que la aplicación acceda a la cámara.

INTERNET Permite que la aplicación realice solicitudes de red y permite la depuración remota.

READ_PHONE_STATE Permite que el motor de ejecución de AIR silencie el audio cuando se produzca una llamada entrante.

RECORD_AUDIO Permite que la aplicación acceda al micrófono.

WAKE_LOCK y DISABLE_KEYGUARD Permite que la aplicación impida que el dispositivo entre en reposo con la configuración de la clase SystemIdleMode.

WRITE_EXTERNAL_STORAGE Permite que la aplicación escriba en la tarjeta de memoria externa del dispositivo.

Firmas de la aplicación

Todos los paquetes de la aplicación creados para la plataforma Android se deben firmar. Debido a que las aplicaciones de AIR en Android se empaquetan en el formato APK de Android nativo, se firman de acuerdo con las convenciones de Android en lugar de conforme a las convenciones de AIR. Aunque Android y AIR utilizan la firma de código de forma similar, existen diferencias importantes:

- En Android, la firma verifica que la clave privada esté en posesión del desarrollador, pero no se utiliza para comprobar la identidad del mismo.

- Para las aplicaciones enviadas al mercado de Android, el certificado debe ser válido durante 25 años como mínimo.
- Android no admite la migración de la firma del paquete a otro certificado. Si una actualización se firma mediante un certificado diferente, el usuario debe desintalar la aplicación original antes de que pueda instalar la aplicación actualizada.
- Dos aplicaciones firmadas con el mismo certificado pueden especificar un ID compartido que les permita acceder a los archivos de datos y a la caché entre sí. (Este uso compartido no lo facilita AIR.)

IDs de usuario de la aplicación

Android utiliza un núcleo Linux. A todas las aplicaciones instaladas se les asigna un ID de usuario de tipo Linux que determina sus permisos para las operaciones como el acceso a archivos. Los archivos de los directorios temporales, de almacenamiento de la aplicación y de la propia aplicación se protegen frente al acceso mediante los permisos del sistema de archivos. Los archivos escritos en el almacenamiento externo (es decir, la tarjeta SD) se pueden leer, modificar y eliminar por otras aplicaciones, o por el usuario, cuando la tarjeta SD se instala como dispositivo de almacenamiento masivo en un equipo.

Las cookies recibidas con solicitudes de Internet no se comparten entre aplicaciones de AIR.

Privacidad de la imagen de fondo

Cuando un usuario pone la aplicación en segundo plano, algunas versiones de Android hacen una captura de pantalla que utiliza una miniatura de la lista de aplicaciones recientes. Esta captura de pantalla se almacena en la memoria del dispositivo y a ella puede acceder un atacante con control físico del dispositivo.

Si la aplicación muestra información confidencial, estos datos se deben proteger para que no se capturen mediante la captura de pantalla de fondo. El evento `deactivate` distribuido mediante el objeto NativeApplication señala que una aplicación está a punto de cambiar a segundo plano. Utilice este evento para borrar u ocultar la información confidencial.

Más temas de ayuda

Android: Seguridad y permisos

Datos cifrados en Android

Las aplicaciones de AIR en Android pueden utilizar las opciones de cifrado disponibles en la base de datos SQL incorporada para guardar datos cifrados. Para lograr la mejor seguridad, base la clave de cifrado en una contraseña introducida por el usuario que deba facilitarse cada vez que se ejecute la aplicación. Es muy difícil (casi imposible) “ocultar” una clave de cifrado o contraseña almacenada localmente para que ningún atacante pueda acceder a los archivos de la aplicación. Si el atacante puede recuperar la clave, el cifrado de los datos no garantizará protección adicional, más allá de la seguridad del sistema de archivos (basada en el ID de usuario) que proporciona el sistema Android.

La clase EncryptedLocalStore se puede utilizar para guardar datos, pero estos datos no estarán cifrados en dispositivos Android. En su lugar, el modelo de seguridad de Android se basa en el ID de usuario de la aplicación para proteger los datos de otras aplicaciones. Las aplicaciones que utilizan un ID de usuario compartido y que comparten el mismo certificador de firma del código, usan el mismo almacenamiento local de cifrado.

Importante: *en un teléfono raíz, cualquier aplicación que se ejecute con privilegios de raíz puede acceder a los archivos de cualquier otra aplicación. Así, los datos guardados en el almacenamiento local de cifrado no están seguros en un dispositivo raíz.*

Seguridad en dispositivos iOS

En iOS, AIR se ajusta al modelo de seguridad nativo. Al mismo tiempo, AIR mantiene sus propias reglas de seguridad, que pretenden facilitar a los desarrolladores la escritura de aplicaciones seguras y conectadas a Internet.

Las aplicaciones de AIR en iOS utilizan el formato de empaquetado de iOS, por lo que la instalación falla en el modelo de seguridad de iOS. El archivo de instalación de la aplicación de AIR no se utiliza. Además, en los dispositivos iOS no se utiliza un motor de ejecución de AIR independiente. Cada aplicación de AIR contiene todo el código necesario para dicha función.

Firmas de la aplicación

Todos los paquetes de la aplicación creados para la plataforma iOS se deben firmar. Dado que las aplicaciones de AIR en iOS son paquetes en formato IPA de iOS, se firman de acuerdo con los requisitos de iOS y no con los de AIR. Aunque Android y iOS utilizan la firma de código de forma similar, existen diferencias importantes:

- EN iOS, el certificado que se usa para firmar una aplicación debe estar emitido por Apple; no se podrán usar certificados de otras entidades emisoras.
- En iOS, los certificados de distribución emitidos por Apple suelen ser válidos durante un año.

Privacidad de la imagen de fondo

Cuando un usuario cambia a una aplicación en el fondo en el iOS, este captura una captura de pantalla que utiliza para animar la transición. Esta captura de pantalla se almacena en la memoria del dispositivo y a ella puede acceder un atacante con control físico del dispositivo.

Si la aplicación muestra información confidencial, estos datos se deben proteger para que no se capturen mediante la captura de pantalla de fondo. El evento `deactivate` distribuido mediante el objeto NativeApplication señala que una aplicación está a punto de cambiar a segundo plano. Utilice este evento para borrar u ocultar la información confidencial.

Capítulo 65: Cómo utilizar ejemplos de ActionScript

Ejecutar un ejemplo de código de ActionScript 3.0 es la mejor forma de aprender cómo funcionan determinadas clases y métodos. Puede utilizar los ejemplos de muchas maneras, en función del dispositivo que esté utilizando o de la finalidad.

Equipos con Flash Professional o Flash Builder Consulte "Ejecución de ejemplos de ActionScript 3.0 en Flash Professional" en la página 1147 o "Ejecución de ejemplos de ActionScript 3.0 en Flash Builder" en la página 1148 para obtener más información sobre cómo utilizar estos entornos de desarrollo para ejecutar ejemplos de ActionScript 3.0. Utilice sentencias trace y otras herramientas de depuración para mejorar sus conocimientos sobre el modo en que funciona un ejemplo de código.

Dispositivos móviles Puede ejecutar los ejemplos de código de ActionScript 3.0 en dispositivos móviles que admitan Flash Player 10.1 y versiones posteriores. Consulte "Ejecución de ejemplos de ActionScript 3.0 en dispositivos móviles" en la página 1150. También puede ejecutar estos ejemplos en su equipo si tiene instalado Flash Professional o Flash Builder.

Dispositivos de TV Aunque no es posible ejecutar estos ejemplos en un TV, sí puede aprender de los ejemplos si los ejecuta en un ordenador. Consulte Plataforma de Flash para TV en el sitio web Adobe Developer Connection para obtener más información sobre el desarrollo de aplicaciones para dispositivos de TV.

Tipos de ejemplos

Entre los tipos de ejemplos de código de ActionScript 3.0 se encuentran:

- Ejemplos de fragmento de código (los encontrará en toda la documentación de ActionScript 3.0)
- Ejemplos basados en clases (los encontrará principalmente en la Referencia del lenguaje ActionScript 3.0)
- Ejemplos prácticos con varios archivos de origen (puede descargar los archivos ZIP de origen en www.adobe.com/go/learn_programmingAS3samples_flash_es)

Ejemplos de fragmento de código

Un ejemplo de fragmento de código presenta el siguiente aspecto:

```
var x:int = 5;
trace(x); // 5
```

Los fragmentos de código solo contienen código suficiente para mostrar una sola idea. Normalmente no incluyen un paquete ni sentencias de clase.

Ejemplos basados en clases

Muchos ejemplos muestran el código fuente de una clase completa de ActionScript. Un ejemplo basado en clases presenta el siguiente aspecto:

```
package {
    public class Example1 {
        public function Example1():void {
            var x:int = 5;
            trace(x); //5
        }
    }
}
```

El código para un ejemplo basado en clase incluye una sentencia package, una declaración de clase y una función constructora.

Ejemplos prácticos que contienen varios archivos de origen

Muchos de los temas de la Guía del desarrollador de ActionScript 3.0 concluyen con ejemplos prácticos que muestran cómo utilizar determinadas funciones de ActionScript en un contexto práctico y real. Estos ejemplos suelen contener varios archivos, entre los que se incluyen:

- Uno o varios archivos de origen de ActionScript.
- Un archivo .FLA para su uso con Flash Professional.
- Uno o varios archivos MXML para su uso con Flash Builder.
- Archivos de datos, archivos de imagen, archivos de sonido u otros recursos utilizados por la aplicación de ejemplo (opcional).

Los ejemplos prácticos se suelen distribuir como archivos ZIP.

Lista de ejemplos de la Guía del desarrollador en el archivo ZIP

El archivo ZIP de Flash Professional CS5 y Flex 4 (puede descargarlo en www.adobe.com/go/learn_programmingAS3samples_flash_es) contiene los siguientes ejemplos:

- AlarmClock (“Ejemplo de gestión de eventos: Reloj con alarma” en la página 143)
- AlgorithmicVisualGenerator (“Ejemplo de API de dibujo: Generador visual algorítmico” en la página 240)
- ASCIIArt (“Ejemplo de cadenas: Arte ASCII” en la página 19)
- CapabilitiesExplorer (“Ejemplo de Capabilities: Detección de las características del sistema” en la página 907)
- CustomErrors (“Ejemplo de gestión de errores: Aplicación CustomErrors” en la página 71)
- DisplayObjectTransformer (“Ejemplo de geometría: Aplicación de una transformación de matriz a un objeto de visualización” en la página 225)
- FilterWorkbench (“Ejemplo de filtrado de objetos de visualización: Filter Workbench” en la página 305)
- GlobalStockTicker (“Ejemplo: Internacionalización de una aplicación de bolsa” en la página 991)
- IntrovertIM_HTML (“Ejemplo de API externa: Comunicación entre ActionScript y JavaScript en un navegador web” en la página 883)
- NewsLayout (“Ejemplo de TextField: Formato de texto con estilo periodístico” en la página 407)
- PlayList (“Ejemplo de conjuntos: PlayList” en la página 48)
- PodcastPlayer (“Ejemplo de sonido: Podcast Player” en la página 487)
- ProjectionDragger (“Ejemplo: Proyección en perspectiva” en la página 375)
- ReorderByZ (“Uso de objetos Matrix3D para reordenar la visualización” en la página 380)
- RSSViewer (“Ejemplo de XML en ActionScript: Carga de datos RSS desde Internet” en la página 115)

- RuntimeAssetsExplorer (“Ejemplo de clip de película: RuntimeAssetsExplorer” en la página 346)
- SimpleClock (“Ejemplo de fecha y hora: Un sencillo reloj analógico” en la página 6)
- SpinningMoon (“Ejemplo de mapa de bits: Luna giratoria animada” en la página 264)
- SpriteArranger (“Ejemplo de objeto de visualización: SpriteArranger” en la página 210)
- TelnetSocket (“Ejemplo de socket TCP: Creación de un cliente Telnet” en la página 831)
- VideoJukebox (“Ejemplo de vídeo: Video Jukebox” en la página 528)
- WikiEditor (“Ejemplo de expresiones regulares: Analizador Wiki” en la página 92)
- WordSearch (“Ejemplo de entrada de ratón: WordSearch” en la página 602)

Los ejemplos prácticos también se encuentran en muchos de los artículos de inicio rápido de los centros para desarrolladores de Flash y Flex.

Ejecución de ejemplos de ActionScript 3.0 en Flash Professional

Utilice uno de los siguientes procedimientos (en función del tipo de ejemplo) para ejecutar un ejemplo con Flash Professional.

Ejecución de un ejemplo de fragmento de código en Flash Professional

Para ejecutar un ejemplo de fragmento de código en Flash Professional:

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

   Aparece una nueva ventana de Flash.
3. Haga clic en el primer fotograma de la primera capa en el panel Línea de tiempo.
4. En el panel Acciones, escriba o copie el ejemplo de fragmento de código.
5. Seleccione Archivo > Guardar. Asigne un nombre al archivo y haga clic en Aceptar.
6. Para probar el ejemplo, seleccione Control > Probar película.

Ejecución de un ejemplo basado en clases en Flash Professional

Para ejecutar un ejemplo basado en clases en Flash Professional:

1. Seleccione Archivo > Nuevo.
2. En el cuadro de diálogo Nuevo documento, seleccione Archivo ActionScript y haga clic en Aceptar. Aparece una nueva ventana del editor.
3. Copie el código de ejemplo y péguelo en la ventana del editor.

   Si la clase es la clase del documento principal para el programa, debe ampliar la clase MovieClip:

```
import flash.display.MovieClip;
public class Example1 extends MovieClip{
//...
}
```

También se debe asegurar que todas las clases con referencia en el ejemplo se declaran utilizando sentencias `import`.

4 Seleccione Archivo > Guardar. Asigne al archivo el mismo nombre que la clase en el ejemplo (p. ej. ContextMenuExample.as).

Nota: *algunos de los ejemplos basados en clases, como el ejemplo de la clase flashx.textLayout.container.ContainerController, incluyen varios niveles en la declaración del paquete (`paquete flashx.textLayout.container.examples {`). En estos ejemplos, guarde el archivo en una subcarpeta que coincida con la declaración del paquete (flashx/textLayout/container/examples), o elimine el nombre del paquete (para que ActionScript comience con `package {` solamente) y podrá probar el archivo desde cualquier ubicación.*

5 Seleccione Archivo > Nuevo.

6 En el cuadro de diálogo Nuevo documento, seleccione Documento de Flash (ActionScript 3.0) y haga clic en Aceptar. Aparece una nueva ventana de Flash.

7 En el panel Propiedades, en el campo Clase de documento, indique el nombre de la clase de ejemplo, que debe coincidir con el nombre del archivo de origen de ActionScript que acaba de guardar (p. ej. ContextMenuExample).

8 Seleccione Archivo > Guardar. Asigne al archivo FLA el mismo nombre que la clase en el ejemplo (p. ej. ContextMenuExample.as).

9 Para probar el ejemplo, seleccione Control > Probar película.

Ejecución de un ejemplo práctico en Flash Professional

Los ejemplos prácticos se suelen distribuir como archivos ZIP. Para ejecutar un ejemplo práctico en Flash Professional:

1 Descomprima el archivo de almacenamiento en una carpeta que seleccione.

2 En Flash Professional, seleccione Archivo > Abrir.

3 Desplácese a la carpeta donde descomprimió el archivo de almacenamiento. Seleccione el archivo FLA en esa carpeta y haga clic en Abrir.

4 Para probar el ejemplo, seleccione Control > Probar película.

Ejecución de ejemplos de ActionScript 3.0 en Flash Builder

Utilice uno de los siguientes procedimientos (en función del tipo de ejemplo) para ejecutar un ejemplo en Flash Builder.

Ejecución de un ejemplo de fragmento de código en Flash Builder

Para ejecutar un ejemplo de fragmento de código en Flash Builder:

1 Cree un nuevo proyecto de Flex (seleccione Archivo > Nuevo > Proyecto de Flex), o bien, en un proyecto de Flex existente, cree una nueva aplicación MXML (elija Archivo > Nuevo > Aplicación de MXML). Asigne al proyecto o a la aplicación un nombre descriptivo (p. ej. ContextMenuExample).

2 Dentro del archivo MXML generado, añada una etiqueta `<mx:Script>`.

3 Pegue el contenido del ejemplo de fragmento de código entre las etiquetas `<mx:Script>` y `</mx:Script>`. Guarde el archivo MXML.

4 Para ejecutar el ejemplo, seleccione la opción de menú Ejecutar > Ejecutar para el archivo MXML principal (por ejemplo, Ejecutar > Ejecutar ContextMenuExample).

Ejecución de un ejemplo basado en clases en Flash Builder

Para ejecutar un ejemplo basado en clases en Flash Builder:

1 Seleccione File (Archivo) > New (Nuevo)> ActionScript Project (Proyecto de ActionScript).

2 Indique el nombre de la clase principal (p. ej. ContextMenuExample) en el campo Project Name (Nombre de proyecto). Utilice los valores predeterminados para otros campos (o cámbielos según su entorno específico). Haga clic en Finish (Finalizar) para crear el proyecto y el archivo principal de ActionScript.

3 Borre todo el contenido generado del archivo de ActionScript. Pegue código de ejemplo, incluyendo las sentencias de importación y empaquetado, en el archivo ActionScript y guarde el archivo.

   Nota: *algunos de los ejemplos basados en clases, como el ejemplo de la clase flashx.textLayout.container.ContainerController, incluyen varios niveles en la declaración del paquete (`paquete flashx.textLayout.container.examples {`). En estos ejemplos, guarde el archivo en una subcarpeta que coincida con la declaración del paquete (flashx/textLayout/container/examples), o elimine el nombre del paquete (para que ActionScript comience con `package {` solamente) y podrá probar el archivo desde cualquier ubicación.*

4 Para ejecutar el ejemplo, seleccione la opción del menú Run (Ejecutar) > Run (Ejecutar) para el nombre de la clase de ActionScript principal (p. ej. Run (Ejecutar) > Run ContextMenuExample (Ejecutar ContextMenuExample).

Ejecución de un ejemplo práctico en Flash Builder

Los ejemplos prácticos se suelen distribuir como archivos ZIP. Para ejecutar un ejemplo práctico utilizando Flash Builder:

1 Descomprima el archivo de almacenamiento en una carpeta que seleccione. Asigne a la carpeta un nombre descriptivo (por ejemplo, ContextMenuExample).

2 En Flash Builder, seleccione File (Archivo) > New (Nuevo)> Flex Project (Proyecto de Flex).. En la sección Project Location (Ubicación del proyecto), haga clic en Browse (Examinar) y seleccione la carpeta que contiene los archivos de ejemplo. En el campo Project Name (Nombre de proyecto), indique el nombre de la carpeta (p. ej. ContextMenuExample). Utilice los valores predeterminados para otros campos (o cámbielos según su entorno específico). Haga clic en Next (Siguiente) para continuar.

3 En el panel Output (Salida), haga clic en Next (Siguiente) para aceptar el valor predeterminado.

4 En el panel Source Paths (Rutas de origen), haga clic en el botón Browse (Examinar) situado junto al campo Main Application File (Archivo de aplicación principal). Seleccione el archivo de ejemplo MXML principal en la carpeta de ejemplo. Haga clic en Finish (Finalizar) para crear los archivos del proyecto.

5 Para ejecutar el ejemplo, seleccione la opción de menú Ejecutar > Ejecutar para el archivo MXML principal (por ejemplo, Ejecutar > Ejecutar ContextMenuExample).

Ejecución de ejemplos de ActionScript 3.0 en dispositivos móviles

Los ejemplos de código de ActionScript 3.0 se pueden ejecutar en dispositivos móviles que admiten Flash Player 10.1. No obstante, un ejemplo de código se suele ejecutar para aprender el funcionamiento de clases y métodos concretos. En este caso, ejecute el ejemplo en un dispositivo que no sea móvil como, por ejemplo, un equipo de escritorio. En el equipo de escritorio, se pueden utilizar sentencias trace y otras herramientas de depuración en Flash Professional o Flash Builder para ampliar el conocimiento de un ejemplo de código.

Si desea ejecutar el ejemplo en un dispositivo móvil, puede copiar los archivos en el dispositivo o en un servidor web. Para copiar archivos en el dispositivo y ejecutar el ejemplo en el navegador, realice lo siguiente:

1. Cree el archivo SWF siguiendo las instrucciones indicadas en "Ejecución de ejemplos de ActionScript 3.0 en Flash Professional" en la página 1147 o en "Ejecución de ejemplos de ActionScript 3.0 en Flash Builder" en la página 1148. En Flash Professional, el archivo SWF se crea cuando se selecciona Control > Probar película. En Flash Builder, el archivo SWF se crea cuando se ejecuta, se depura o se crea el proyecto de Flash Builder.
2. Copie el archivo SWF en un directorio en el dispositivo móvil. Utilice el software que acompaña al dispositivo para copiar el archivo.
3. En la barra de direcciones del navegador del dispositivo móvil, indique la URL file:// para el archivo SWF. Por ejemplo, indique `file:://applications/myExample.swf`.

Para copiar archivos en un servidor web y ejecutar el ejemplo en el navegador del dispositivo, realice lo siguiente:

1. Cree un archivo SWF y otro HTML. En primer lugar, siga las instrucciones incluidas en "Ejecución de ejemplos de ActionScript 3.0 en Flash Professional" en la página 1147 o en "Ejecución de ejemplos de ActionScript 3.0 en Flash Builder" en la página 1148. En Flash Professional, al seleccionar Control (Control) > Test Movie (Probar película) se crea únicamente el archivo SWF. Para crear ambos archivos, en primer lugar seleccione tanto Flash como HTML en la ficha Formats (Formatos) del cuadro de diálogo Publish Settings (Configuración de publicación). A continuación, seleccione Archivo > Publicar para crear los archivos HTML y SWF. En Flash Builder, los archivos SWF y HTML se crean cuando se ejecuta, depura o se crea el proyecto de Flash Builder.
2. Copie los archivos SWF y HTML en un directorio del servidor web.
3. En la barra de direcciones del navegador del dispositivo móvil, introduzca la dirección HTTP del archivo HTML. Por ejemplo, indique `http://www.myWebServer/examples/myExample.html`.

Antes de ejecutar un ejemplo en un dispositivo móvil, tenga en cuenta los siguientes puntos.

Tamaño del escenario

El tamaño del escenario que se utiliza cuando se ejecuta un ejemplo en un dispositivo móvil es mucho más pequeño que cuando se emplea cualquier otro tipo de dispositivo. Muchos ejemplos no requieren ningún tamaño de escenario concreto. Al crear el archivo SWF, especifique un tamaño de escenario apropiado para el dispositivo. Por ejemplo, especifique 176 x 208 píxeles.

El objetivo de los ejemplos prácticos de la Guía del desarrollador de ActionScript 3.0 consiste en ilustrar distintos conceptos y clases de ActionScript 3.0. Sus interfaces de usuario están diseñadas para ofrecer un buen aspecto y funcionamiento en un equipo portátil y de escritorio. Aunque los ejemplos funcionan en dispositivos móviles, el tamaño del escenario y el diseño de la interfaz de usuario no son adecuados para la pantalla pequeña. Adobe recomienda la ejecución de los ejemplos prácticos en un equipo para aprender ActionScript y posteriormente utilizar los fragmentos de código necesarios en las aplicaciones móviles.

Campos de texto en lugar de sentencias trace

Al ejecutar un ejemplo en un dispositivo móvil, no es posible ver el resultado en sus sentencias trace. Para ver el resultado, cree una instancia de la clase TextField. A continuación, añada el texto de las sentencias trace a la propiedad `text` del campo de texto.

Se puede utilizar la siguiente función para configurar un campo de texto para seguimiento:

```
function createTracingTextField(x:Number, y:Number,
                                width:Number, height:Number):TextField {

    var tracingTF:TextField = new TextField();
    tracingTF.x = x;
    tracingTF.y = y;
    tracingTF.width = width;
    tracingTF.height = height;

    // A border lets you more easily see the area the text field covers.
    tracingTF.border = true;
    // Left justifying means that the right side of the text field is automatically
    // resized if a line of text is wider than the width of the text field.
    // The bottom is also automatically resized if the number of lines of text
    // exceed the length of the text field.
    tracingTF.autoSize = TextFieldAutoSize.LEFT;

    // Use a text size that works well on the device.
    var myFormat:TextFormat = new TextFormat();
    myFormat.size = 18;
    tracingTF.defaultTextFormat = myFormat;

    addChild(tracingTF);
    return tracingTF;
}
```

Por ejemplo, añada esta función a la clase de documento como función privada. Después, en otros métodos de la clase de documento, realice un seguimiento de los datos con código tal y como se muestra a continuación:

```
var traceField:TextField = createTracingTextField(10, 10, 150, 150);
// Use the newline character "\n" to force the text to the next line.
traceField.appendText("data to trace\n");
traceField.appendText("more data to trace\n");
// Use the following line to clear the text field.
traceField.appendText("");
```

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

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

Tamaño del texto

Muchos ejemplos utilizan campos de texto para ayudar a ilustrar un concepto. En ocasiones, el ajuste del tamaño del texto en el campo de texto proporciona mejor legibilidad en un dispositivo móvil. Por ejemplo, si un ejemplo utiliza una instancia del campo de texto denominada `myTextField`, cambie el tamaño de su texto con el siguiente código:

```
// Use a text size that works well on the device.
var myFormat:TextFormat = new TextFormat();
myFormat.size = 18;
myTextField.defaultTextFormat = myFormat
```

Captura de entradas del usuario

El navegador y el sistema operativo móvil capturan algunos eventos de entrada de usuario que no recibe el contenido SWF. El comportamiento específico depende del navegador y el sistema operativo, pero se puede producir un comportamiento inesperado cuando los ejemplos se ejecutan en un dispositivo móvil. Para obtener más información, consulte "Precedencia de KeyboardEvent" en la página 586.

Asimismo, las interfaces de usuario de muchos ejemplos están diseñadas para un equipo portátil o de escritorio. Por ejemplo, la mayor parte de los ejemplos prácticos de la Guía del desarrollador de ActionScript 3.0 también son adecuados para la visualización de escritorio. Por lo tanto, en ocasiones no se puede ver todo el escenario en la pantalla del dispositivo móvil. La capacidad para desplazarse por el contenido del navegador depende del propio navegador. Asimismo, los ejemplos no están diseñados para capturar y administrar eventos de desplazamiento lateral y vertical. Por esta razón, algunas interfaces de usuario de los ejemplos no resultan adecuadas para su ejecución en la pequeña pantalla. Adobe recomienda que los ejemplos se ejecuten en un equipo para aprender ActionScript y posteriormente utilizar los fragmentos de código necesarios en las aplicaciones móviles.

Para obtener más información, consulte "Desplazamiento lateral y vertical de objetos de visualización" en la página 183.

Administración de la selección

Algunos ejemplos requieren que se proporcione la selección a un campo. Con esta operación se puede, por ejemplo, introducir texto o seleccionar un botón. Para seleccionar un campo, utilice el dispositivo de señalización del dispositivo móvil como, por ejemplo, una pluma o el dedo. O bien, utilice las teclas de navegación del dispositivo móvil para seleccionar un campo. Para elegir un botón que tenga la selección, utilice la tecla para seleccionar del dispositivo móvil tal y como se usa Intro en un ordenador. En algunos dispositivos, al puntear dos veces en un botón, este se selecciona.

Para obtener más información sobre la selección, consulte "Administración de la selección" en la página 581.

Administración de eventos de ratón

En muchos ejemplos se detectan eventos de ratón. En un equipo, estos eventos puede suceder, por ejemplo, cuando el usuario se desplaza sobre un objeto de visualización con el ratón o hace clic con el botón del ratón en un objeto. En los dispositivos móviles, los eventos del uso de los dispositivos de señalización, como una pluma o el dedo, se denominan eventos de toque. Flash Player 10.1 asigna eventos de toque a eventos de ratón. Con esta asignación se garantiza que el contenido SWF desarrollado antes de Flash Player 10.1 continúe en funcionamiento. Por lo tanto, los ejemplos funcionan cuando se utiliza un dispositivo de señalización para seleccionar o arrastrar un objeto de visualización.

Rendimiento

Los dispositivos móviles cuentan con menos capacidad de procesamiento que los de escritorio. Posiblemente algunos ejemplos que consumen más CPU funcionan más lentamente en dispositivos móviles. En el ejemplo de la sección "Ejemplo de API de dibujo: Generador visual algorítmico" en la página 240 se realizan dibujos y procesamientos extensos al entrar en cada fotograma. Al ejecutar este ejemplo en un equipo se muestran distintas APIs de dibujo. Sin embargo, el ejemplo no resulta adecuado en algunos dispositivos móviles debido a sus limitaciones de rendimiento.

Para obtener más información sobre el rendimiento en los dispositivos móviles, consulte *Optimización del rendimiento para la plataforma Flash*.

Prácticas recomendadas

En los ejemplos no se consideran las mejores prácticas en el desarrollo de aplicaciones para dispositivos móviles. Las limitaciones de memoria y capacidad de procesamiento de los dispositivos móviles requieren una atención especial. Del mismo modo, la interfaz de usuario para la pequeña pantalla presenta distintas necesidades que una pantalla de escritorio. Para obtener más información sobre el desarrollo de aplicaciones para dispositivos móviles, *consulte Optimización del rendimiento para la plataforma Flash*.

Capítulo 66: Compatibilidad de SQL en bases de datos locales

Adobe AIR incluye un motor de base de datos SQL que admite bases de datos SQL locales con muchas funciones SQL estándar mediante el sistema de código abierto de bases de datos SQLite. El motor de ejecución no especifica cómo ni dónde se almacenan los datos de la base de datos en el sistema de archivos; sino, cada base de datos se almacena completamente dentro de un único archivo. Un desarrollador puede especificar la ubicación de grabación de la base de datos en el sistema de archivos y hacer que una sola aplicación de AIR acceda a una o varias bases de datos independientes (es decir, archivos de base de datos independientes). En este documento se describe la sintaxis SQL y la compatibilidad de tipos de datos de bases de datos SQL locales con Adobe AIR. Este documento no pretende ser una referencia exhaustiva de SQL. Más bien, describe detalles concretos del lenguaje SQL admitidos por Adobe AIR. El motor de ejecución admite la mayor parte del lenguaje SQL de la norma SQL-92. Dada la enorme cantidad de referencias, sitios web, manuales y material de aprendizaje de SQL, este documento no pretende ser una referencia detallada ni un tutorial sobre SQL. Este documento se centra especialmente en la sintaxis SQL admitida por AIR, y en las diferencias entre el lenguaje SQL-92 y el SQL admitido.

Convenciones de definición de declaraciones SQL

En las definiciones de declaraciones de este documento se han utilizando las siguientes convenciones:

- Mayúsculas/Minúsculas
  - UPPER CASE: las palabras clave SQL de literal se escriben todas en mayúsculas.
  - lower case: los marcadores de posición o los nombres de las cláusulas se escriben todos en minúsculas.
- Caracteres de definición
  - ::= Indica una definición de cláusula o de declaración.
- Caracteres de agrupación y alternancia
  - | El carácter de barra se utiliza entre opciones alternativas y se puede leer como “or”.
  - [] Los elementos entre corchetes son opcionales; los corchetes pueden contener un solo elemento o varios elementos.
  - () Los paréntesis que encierran un conjunto de alternativas (conjunto de elementos separados por el carácter barra) designan un grupo requerido de elementos, es decir, un conjunto de elementos que son los valores posibles de un solo elemento requerido.
- Cuantificadores
  - + Un signo "más" después de un elemento entre paréntesis indica que el elemento precedente puede existir una o varias veces.
  - * Un asterisco después de un elemento entre corchetes indica que el elemento precedente (entre corchetes) puede existir 0 o más veces.
- Caracteres literales
  - * Un asterisco en un nombre de columna o entre paréntesis después de un nombre de función indica equivale al carácter de asterisco literal más que al cuantificador "0 oo más".
  - . Un punto representa un punto literal.
  - , Una coma representa una coma literal

- () Un par de paréntesis encerrando una sola cláusula o elemento indica que es necesario el uso de paréntesis literales.
- Otros caracteres: a no ser que se indique lo contrario, el resto de caracteres representan sus equivalentes literales.

Sintaxis SQL admitida

Las siguientes listas de sintaxis SQL se admiten en el motor de la base de datos SQL de Adobe AIR. El contenido se divide en explicaciones de las distintas declaraciones y tipos de cláusulas, funciones incorporadas y operadores. Se trata lo siguiente:

- Sintaxis SQL general
- Declaraciones de manipulación de datos (SELECT, INSERT, UPDATE y DELETE)
- Declaraciones de definición de datos (declaraciones CREATE, ALTER y DROP para tablas, índices, vistas y activadores)
- Declaraciones y cláusulas especiales
- Funciones incorporadas (funciones de agregación, escalares y de formato de fecha/hora)
- Operadores
- Parámetros
- Funciones SQL no admitidas
- Funciones SQL adicionales

Sintaxis SQL general

Además de la sintaxis específica de las distintas declaraciones y expresiones, a continuación enumeran algunas reglas generales de SQL:

Distinción entre mayúsculas y minúsculas Las declaraciones SQL, incluidos los nombres de objetos, no distinguen entre mayúsculas y minúsculas. No obstante, las declaraciones SQL se suelen escribir con palabras clave SQL en mayúscula, por lo que seguiremos dicha convención en este documento. Aunque la sintaxis SQL no distingue entre mayúsculas y minúsculas, los valores de texto literales sí lo hacen en SQL. Las operaciones de comparación y ordenación también distinguen entre mayúsculas y minúsculas, según se haya especificado en la secuencia de intercalación definida para una columna o una operación. Para más información, consulte COLLATE.

Espacio en blanco Debe usarse un carácter de espacio en blanco (por ejemplo, un espacio, una tabulación, un salto de línea, etc.) para separar palabras independientes en una declaración SQL. Sin embargo, el espacio en blanco es optativo entre palabras y símbolos El tipo y la cantidad de caracteres de espacio en blanco en una declaración SQL no es relevante. Puede utilizar espacios en blanco (como la sangría o saltos de línea) para dar un formato más claro a las declaraciones SQL sin que afecte a su funcionalidad.

Declaraciones de manipulación de datos

Las declaraciones de manipulación de datos son declaraciones SQL más utilizadas. Estas declaraciones se utilizan para recuperar, añadir y eliminar datos de las tablas de la base de datos. Se admiten las siguientes declaraciones de manipulación de datos: SELECT, INSERT, UPDATE, y DELETE.

SELECT

La declaración SELECT se utiliza para consultar la base de datos. El resultado de una declaración SELECT es cero o más filas de datos, donde cada fila tiene un número fijo de columnas. El número de columnas del resultado se especifica mediante el nombre de la columna result o la lista de expresiones entre las palabras clave SELECT y FROM (opcional).

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

Cualquier expresión arbitraria se puede utilizar como resultado. Si una expresión del resultado es *, todas las columnas de todas las tablas se reemplazan por dicha expresión. Si la expresión es el nombre de una tabla seguido de .*,el resultado es todas las columnas de dicha tabla.

La palabra clave DISTINCT provoca la devolución de un subconjunto de filas de resultados (donde cada fila es distinta). Se asume que todos los valores de NULL son iguales. El comportamiento predeterminado es la devolución de todas las filas de resultados, que se pueden explicitar con la palabra clave ALL.

La consulta se ejecuta en una o varias tablas especificadas con la palabra clave FROM. Si los distintos nombres de las tablas están separados por comas, la consulta utiliza la unión cruzada de las distintas tablas. La sintaxis JOIN también se puede utilizar para especificar el modo de unión de las tablas. El único tipo de unión externa admitida es LEFT OUTER JOIN. Las expresiones de la cláusula ON deben producir como resultado un valor booleano. Se puede utilizar una subconsulta entre paréntesis como una tabla de la cláusula FROM. Se puede omitir toda la cláusula FROM, en cuyo caso, el resultado será una única fila con los valores de la lista de expresiones del resultado.

La cláusula WHERE se utiliza para limitar el número de filas recuperadas por la consulta. Las expresiones de la cláusula WHERE deben producir como resultado un valor booleano. El filtrado de la cláusula WHERE se lleva a cabo antes de cualquier agrupación, por lo que las expresiones de la cláusula WHERE no pueden incluir funciones de agregación.

La cláusula GROUP BY combina una o varias filas del resultado en una sola fila de resultados. La cláusula GROUP BY es especialmente útil cuando el resultado contiene funciones de agregación. Las expresiones de la cláusula GROUP BY no tienen por qué ser expresiones que aparezcan en la lista de expresiones SELECT.

La cláusula HAVING se parece a WHERE, en el sentido que limita las filas devueltas por la declaración. Sin embargo, la cláusula HAVING se aplica tras producirse cualquier agrupación especificada por una cláusula GROUP BY. En consecuencia, la expresión HAVING puede hacer referencia a valores que incluyan funciones de agregación. No es necesario que aparezca ninguna expresión de la cláusula HAVING en la lista SELECT. Al igual que ocurre con la expresión WHERE, la expresión HAVING debe producir como resultado un valor booleano.

La cláusula ORDER BY provoca la ordenación de las filas de resultados. El argumento sort-expr-list de la cláusula ORDER BY es una lista de expresiones utilizadas como clave para la ordenación. Las expresiones no tienen por qué formar parte del resultado de una declaración SELECT, pero en una declaración SELECT compuesta (una declaración SELECT que utilice uno de los operadores compound-op), cada expresión de ordenación debe coincidir exactamente con una de las columnas de resultados. Cada expresión de ordenación puede ir acompañada, de forma opcional, de una cláusula sort-order formada por la palabra clave COLLATE y el nombre de la función de intercalación utilizada para ordenar el texto, o bien la palabra clave ASC o DESC para especificar el tipo de ordenación (ascendente o descendente). sort-order se puede omitir y utilizar el valor predeterminado (orden ascendente). Para obtener una definición de la cláusula COLLATE y de las funciones de intercalación, consulte COLLATE.

La cláusula LIMIT pone un límite superior al número de filas devueltas en el resultado. Un valor negativo de LIMIT indica la ausencia de límite superior. El OFFSET opcional que sigue a LIMIT especifica el número de filas que se deben saltar al principio del conjunto de resultados. En una consulta SELECT compuesta, la cláusula LIMIT tal vez aparezca solo tras la declaración SELECT final. El límite se aplica a toda la consulta. Tenga en cuenta que si utiliza la palabra clave OFFSET en la cláusula LIMIT, el límite es el primer entero y el desfase es el segundo entero. Si se utiliza una coma en vez de la palabra clave OFFSET, el desfase es primer número y el límite es el segundo. Esta aparente contradicción es intencionada: maximiza la compatibilidad con sistemas de bases de datos SQL heredados.

Un SELECT compuesto está formado por do o más declaraciones SELECT simples conectadas por uno de los operadores UNION, UNION ALL, INTERSECT o EXCEPT. En un SELECT, compuesto, todas las declaraciones SELECT que lo forman deben especificar el mismo número de columnas de resultados. Solo puede haber una cláusula ORDER BY única tras la declaración SELECT final (y antes de la cláusula LIMIT única, si se ha especificado). Los operadores UNION y UNION ALL combinan los resultados de las declaraciones SELECT siguiente y anterior en una sola tabla. La diferencia reside en que en UNION, todas las filas de resultados son distintas, mientras que en UNION ALL puede haber duplicados. El operador INTERSECT toma la intersección de los resultados de las declaraciones SELECT anterior y siguiente. EXCEPT toma el resultado del SELECT anterior tras eliminar los resultados del SELECT siguiente. Si se conectan tres o más declaraciones SELECT en una compuesta, se agrupan de la primera a la última.

Para obtener una definición de las expresiones permitidas, consulte Expresiones.

Desde AIR 2.5, el operador CAST de SQL es compatible al leer para convertir los datos BLOB a objetos ByteArray de ActionScript. Por ejemplo, el siguiente código lee los datos sin procesar que no se almacenen en formato AMF y los almacena en un objeto ByteArray:

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

INSERT

La declaración INSERT se expresa de dos formas básicas y se utiliza para llenar tablas con datos.

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] VALUES (value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] select-statement
               REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                    REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

La primera expresión (con la palabra clave VALUES) crea una única nueva fila en una tabla existente. Si no se especifica un column-list, el número de valores debe coincidir con el número de columnas de la tabla. Si se especifica un column-list, el número de valores debe coincidir con el número de columnas especificadas. Las columnas de la tabla que no aparecen en la lista de columnas se llenan con valores predeterminados en el momento de creación de la tabla, o con NULL si no hay ningún valore predeterminado definido.

La segunda expresión de la declaración INSERT toma los datos de una declaración SELECT. El número de columnas del resultado de SELECT debe coincidir exactamente con el número de columnas de la tabla si no se especifica column-list, o debe coincidir con el número de columnas especificadas en el column-list. Se realiza una nueva entrada en la tabla por cada fila del resultado de SELECT. El SELECT puede ser simple o compuesto. Para obtener una definición de las declaraciones SELECT admitidas, consulte SELECT.

El conflict-algorithm opcional permite especificar un algoritmo de resolución de conflictos de restricciones alternativo y utilizarlo durante este comando. Para ver una explicación y definición de algoritmos de conflicto, consulte "Declaraciones y cláusulas especiales" en la página 1165.

Las dos formas de REPLACE INTO de la declaración equivalen a utilizar la expresión INSERT [OR conflict-algorithm] estándar con el algoritmo de conflictos REPLACE (p.ej. la forma INSERT OR REPLACE...).

Las dos formas de REPLACE INTO de la declaración equivalen a utilizar la expresión INSERT [OR conflict-algorithm] estándar con el algoritmo de conflictos REPLACE (p.ej. la forma INSERT OR REPLACE...).

UPDATE

El comando de actualización cambia los registros existentes en una tabla.

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,...
[WHERE expr]
```

El comando consta de una palabra clave UPDATE seguida del nombre de la tabla en la que se desean actualizar los registros. Tras la palabra clave SET, indique el nombre de la columna y el valor con el que la columna cambiará como lista separada por comas. La expresión de la cláusula WHERE proporciona la fila o filas en las que se actualizan los registros.

DELETE

El comando DELETE se utiliza para eliminar registros de una tabla.

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

El comando está formado por las palabras clave DELETE FROM seguidas del nombre de la tabla de la que se eliminarán los registros.

Sin una cláusula WHERE, se eliminan todas las filas de la tabla. Si se facilita una cláusula WHERE, solo se eliminan las filas que coincidan con la expresión. La expresión de la cláusula WHERE debe producir como resultado un valor booleano. Para obtener una definición de las expresiones permitidas, consulte Expresiones.

Declaraciones de definición de datos

Las declaraciones de definición de datos se utilizan para crear, modificar y eliminar objetos de bases de datos, como tablas, vistas, índices y activadores. Se admiten las siguientes declaraciones de definición de datos:

- Tablas:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- Índices:
  - CREATE INDEX
  - DROP INDEX

- Vistas:
  - CREATE VIEWS
  - DROP VIEWS
- Activadores:
  - CREATE TRIGGERS
  - DROP TRIGGERS

CREATE TABLE

Una declaración CREATE TABLE está formada por las palabras clave CREATE TABLE, seguidas del nombre de la nueva tabla y (entre paréntesis) una lista de definiciones y restricciones de columnas. El nombre de la tabla puede ser un identificador o una cadena.

```
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-
name
                        ( column-def [, column-def]* [, constraint]* )
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-
statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value     ::=  NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

Cada definición de columna es el nombre de la columna seguido de su tipo de datos correspondiente y, por último, una o varias restricciones de columna opcionales. El tipo de datos de la columna limita los datos que pueden guardarse en dicha columna. Si se intenta guardar un valor en una columna con tipos de datos distintos, el motor de ejecución convertirá el valor en el tipo adecuado si es posible, o lanzará un error. Consulte la sección Compatibilidad de tipos de datos para obtener más información.

La restricción de la columna NOT NULL indica que la columna no puede contener valores NULL.

Una restricción UNIQUE provoca la creación de un índice en la columna o columnas especificadas. Este índice puede contener claves únicas: dos filas no pueden contener valores duplicados o combinaciones de valores para la columna o columnas especificadas. Una declaración CREATE TABLE puede tener varias restricciones UNIQUE, incluidas varias columnas con una restricción UNIQUE en la definición de la columna o múltiples restricciones UNIQUE en el nivel de las tablas.

Una restricción CHECK define una expresión que se calcula y debe ser true para poder insertar o actualizar los datos de una fila. La expresión CHECK debe producir como resultado un valor booleano.

Una cláusula COLLATE en una definición de columna especifica qué función de intercalación de texto debe usarse al comparar entradas de texto en la columna. La función de intercalación BINARY se utiliza de forma predeterminada. Para obtener más información sobre la cláusula COLLATE y sus funciones de intercalación, consulte COLLATE.

La restricción DEFAULT especifica un valor predeterminado para utilizarlo con un INSERT. El valor puede ser NULL, una constante de cadena o un número. El valor predeterminado también puede ser una de las palabras clave especiales (no distinguen entre mayúsculas y minúsculas) CURRENT_TIME, CURRENT_DATE o CURRENT_TIMESTAMP. Si el valor es NULL, se inserta (literalmente) una constante de cadena o un número en la columna cada vez que una declaración INSERT no especifique un valor para la columna. Si el valor es CURRENT_TIME, CURRENT_DATE o CURRENT_TIMESTAMP, se inserta en la columna la fecha/hora UTC actual. Para CURRENT_TIME, el formato es HH:MM:SS. Para CURRENT_DATE, el formato es YYYY-MM-DD. El formato para CURRENT_TIMESTAMP es YYYY-MM-DD HH:MM:SS.

Al especificar un PRIMARY KEY, normalmente solo se crea un índice de UNIQUE en la columna o columnas correspondientes. Sin embargo, si la restricción PRIMARY KEY se encuentra en una sola columna con el tipo de datos INTEGER (o a uno de sus equivalentes como int), entonces la base de datos utiliza dicha columna como clave principal actual para la tabla. Esto significa que la columna solo puede contener valores enteros únicos. (Tenga en cuenta que en muchas implementaciones de SQLite solo el tipo de columna INTEGER hace que la columna sirva de clave principal interna, pero en Adobe AIR, los equivalentes de INTEGER como int también proporcionan este comportamiento.)

Si una tabla no contiene una columna INTEGER PRIMARY KEY, se generará automáticamente una clave de enteros al insertar una fila. Siempre se puede acceder a la clave principal de una fila mediante uno de los nombres especiales ROWID, OID o _ROWID_. Puede utilizar estos nombres tanto si son un INTEGER PRIMARY KEY declarado explícitamente o un valor interno generado. No obstante, si la tabla tiene un INTEGER PRIMARY KEY explícito, el nombre de la columna de los datos del resultado es el nombre real de la columna, no el nombre especial.

Una cINTEGER PRIMARY KEY también puede contener la palabra clave AUTOINCREMENT. Cuando se utiliza la palabra clave AUTOINCREMENT , la base de datos genera automáticamente una clave entera incrementada y la inserta en la columna INTEGER PRIMARY KEY cuando ejecuta una declaración INSERT que no especifica ningún valor explícito para la columna.

Solo puede haber una restricción PRIMARY KEY en una declaración CREATE TABLE. Puede ser parte de una definición de columna o tratarse de una restricción PRIMARY KEY única en el nivel de tablas. Una columna de clave principal es NOT NULL implícitamente.

La cláusula conflict-clause opcional con restricciones permite especificar un algoritmo de resolución de conflictos de restricciones alternativo predeterminado para dicha restricción. El valor predeterminado es ABORT. Las restricciones distintas de la misma tabla pueden tener distintos algoritmos de resolución de conflictos predeterminados. Si una declaración INSERT o UPDATE especifica un algoritmo de resolución de conflictos de restricciones distinto, dicho algoritmo se utilizará en vez del especificado en la declaración CREATE TABLE. Consulte la sección ON CONFLICT de “Declaraciones y cláusulas especiales” en la página 1165 para obtener más información.

Las restricciones adicionales, como las FOREIGN KEY, no producen errores porque el motor de ejecución las omite.

Si las palabras clave TEMP o TEMPORARY se encuentran entre CREATE y TABLE, la tabla creada solo será visible dentro de la misma conexión de base de datos (instancia de SQLConnection). Se elimina automáticamente cuando se cierra la conexión de base de datos. Todos los índices creados en una tabla temporal también son temporales. Las tablas y los índices temporales se guardan en un archivo independiente distinto del archivo principal de la base de datos.

Si se especifica el prefijo opcional database-name, la tabla se crea en una base de datos especificada (base de datos conectada a la instancia de SQLConnection mediante la llamada al método attach() con el nombre de base de datos especificado). Es un error especificar un prefijo database-name y la palabra clave TEMP, a no ser que el prefijo database-name sea temp. Si no se especifica ningún nombre de base de datos y la palabra clave TEMP no está presente, la tabla se crea en la base de datos principal (la base de datos conectada a la instancia de SQLConnection mediante el método open() o openAsync()).

No hay límites arbitrarios en el número de columnas o en el número de restricciones de una tabla. Tampoco hay límites arbitrarios en la cantidad de datos de una fila.

La expresión CREATE TABLE AS define la tabla como el conjunto de resultados de una consulta. Los nombres de las columnas de la tabla son los nombres de las columnas del resultado.

Si la cláusula IF NOT EXISTS opcional está presente y ya existe otra tabla con el mismo nombre, la base de datos ignora el comando CREATE TABLE.

Se puede eliminar una tabla con la declaración DROP TABLE, y se pueden realizar cambios limitados con la declaración ALTER TABLE.

ALTER TABLE

El comando ALTER TABLE permite al usuario renombrar o agregar una nueva columna a una tabla existente. No es posible eliminar una columna de una tabla.

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

Se utiliza la sintaxis RENAME TO para renombrar la tabla identificada por [database-name.] como new-table-name. Este comando no se puede utilizar para mover una tabla entre bases de datos asociadas; solo sirve para renombrar una tabla dentro de la misma base de datos.

Si la tabla que se va a renombrar contiene activadores o índices, estos permanecen asociados a la tabla una vez renombrada. No obstante, si hay definiciones o declaraciones de vistas ejecutadas por activadores que hacen referencia a la tabla que se está renombrando, no se actualizan automáticamente con el nuevo nombre de la tabla. Si la tabla renombrada tiene vistas o activadores asociados, debe descartarlos manualmente y recrear las definiciones de los activadores o las vistas con el nuevo nombre de la tabla.

Se utiliza la sintaxis ADD [COLUMN] para agregar una nueva columna a una tabla existente. La nueva columna siempre se añade al final de la lista de las columnas existentes. La cláusula column-def puede expresarse de todas las formas permitidas en una declaración CREATE TABLE, aunque con las siguientes restricciones:

- La columna no puede tener una restricción PRIMARY KEY o UNIQUE.
- La columna no puede tener un valor predeterminado CURRENT_TIME, CURRENT_DATE o CURRENT_TIMESTAMP.
- Si se especifica una restricción NOT NULL, la columna debe tener un valor predeterminado distinto de NULL.

El tiempo de ejecución de la declaración ALTER TABLE no se ve afectado por la cantidad de datos de la tabla.

DROP TABLE

La declaración DROP TABLE elimina una tabla agregada con una declaración CREATE TABLE. La tabla con el table-name especificado es la que se descarta. Se elimina completamente de la base de datos y del archivo del disco. No es posible recuperar la tabla. Todos los índices asociados a la tabla se eliminan también.

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

De forma predeterminada, la declaración DROP TABLE no reduce el tamaño del archivo de base de datos. El espacio libre de la base de datos se conserva y se utiliza en operaciones INSERT posteriores. Para eliminar espacio libre de la base de datos, utilice el método SQLConnection.clean(). Si el parámetro autoClean se establece en true al crear la base de datos por primera vez, el espacio se libera automáticamente.

La cláusula IF EXISTS opcional desactiva el error que se produciría si la tabla no existiese.

CREATE INDEX

El comando CREATE INDEX está formado por las palabras clave CREATE INDEX seguidas del nombre del nuevo índice, la palabra clave ON, el nombre de una tabla creada previamente para indexar y una lista entre paréntesis de los nombres de las columnas cuyos valores se utilizan para la clave del índice.

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

Cada columna puede ir seguida de las palabras clave ASC o DESC para indicar la ordenación, pero la designación del orden se omite en tiempo de ejecución. La ordenación siempre se establece en orden ascendente.

La cláusula COLLATE que sigue a cada nombre de columna define una secuencia de intercalación utilizada para los valores de texto de dicha columna. La secuencia de intercalación predeterminada es la definida para dicha columna en la declaración CREATE TABLE. Si no se especifica ninguna secuencia de intercalación, se utiliza BINARY. Para obtener una definición de la cláusula COLLATE y de las funciones de intercalación, consulte COLLATE.

No hay límites arbitrarios en el número de índices que se pueden asociar a una sola tabla. Tampoco hay límites en el número de columnas de cada índice.

DROP INDEX

La declaración DROP INDEX elimina un índice agregado con la declaración CREATE INDEX. El índice especificado se elimina completamente del archivo de base de datos. La única forma de poder recuperar el índice es volver a escribir el comando CREATE INDEX correspondiente.

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

De forma predeterminada, la declaración DROP INDEX no reduce el tamaño del archivo de base de datos. El espacio libre de la base de datos se conserva y se utiliza en operaciones INSERT posteriores. Para eliminar espacio libre de la base de datos, utilice el método SQLConnection.clean(). Si el parámetro autoClean se establece en true al crear la base de datos por primera vez, el espacio se libera automáticamente.

CREATE VIEW

El comando CREATE VIEW asigna un nombre a una declaración SELECT predefinida. Este nuevo nombre se puede utilizar posteriormente en una cláusula FROM de otra declaración SELECT en vez de un nombre de tabla. Las vistas se suelen utilizar para simplificar las consultas combinando un complejo conjunto de datos (de uso frecuente) en una estructura que se puede utilizar en otras operaciones.

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS
select-statement
```

Si la palabra clave TEMP o TEMPORARY se encuentra entre CREATE y VIEW, la vista creada solo es visible para la instancia de SQLConnection que abrió la base de datos y se elimina automáticamente al cerrarla.

Si se especifica un [database-name], la vista se crea en la base de datos especificada (una base de datos conectada a la instancia de SQLConnection mediante el método attach(), con el argumento name especificado. Es un error especificar un [database-name] y la palabra clave TEMP a no ser que [database-name] sea temp. Si no se especifica ningún nombre de base de datos y la palabra clave TEMP no está presente, la vistase crea en la base de datos principal (la base de datos conectada a la instancia de SQLConnection mediante el método open() o openAsync()).

Las vistas son de solo lectura. Una declaración DELETE, INSERT o UPDATE no se puede utilizar en una vista, a no ser que se haya definido al menos un activador del tipo asociado (INSTEAD OF DELETE, INSTEAD OF INSERT, INSTEAD OF UPDATE). Para obtener más información sobre la creación de un activador para una vista, consulte CREATE TRIGGER.

La declaración DROP VIEW elimina una vista de la base de datos.

DROP VIEW

La declaración DROP VIEW elimina una vista creada con la declaración CREATE VIEW.

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

El view-name especificado es el nombre de la vista que se va a descartar. Se elimina de la base de datos, pero no se modifica ningún dato de las tablas subyacentes.

CREATE TRIGGER

La declaración CREATE TRIGGER se utiliza para agregar activadores al esquema de base de datos. Un activador es una operación de base de datos (trigger-action) que se lleva a cabo automáticamente cuando se produce un evento concreto de base de datos (database-event).

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      [BEFORE | AFTER] database-event
                      ON table-name
                      trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      INSTEAD OF database-event
                      ON view-name
                      trigger-action
database-event  ::=  DELETE |
                      INSERT |
                      UPDATE |
                      UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                      BEGIN
                        trigger-step ;
                        [ trigger-step ; ]*
                      END
trigger-step    ::=  update-statement |
                      insert-statement |
                      delete-statement |
                      select-statement
column-list     ::=  column-name [, column-name]*
```

Un activador se activa cada vez que se produce un DELETE, INSERT o UPDATE en una tabla de base de datos particular, o cuando se actualiza el UPDATE de una o varias columnas de una tabla. Los activadores son permanente, a no ser que se utilice la palabra clave TEMP o TEMPORARY. En ese caso, el activador se elimina cuando se cierra la conexión de base de datos principal de la instancia de SQLConnection. Si no se especifica ningún tiempo, (BEFORE o AFTER) el valor predeterminado del activador es BEFORE.

Solo se admiten activadores FOR EACH ROW, por lo que el texto FOR EACH ROW es opcional. Con un activador FOR EACH ROW, se ejecutan las declaraciones trigger-step en cada fila de la base de datos que se inserte, actualice o elimine. Para ello, la declaración activa el activador si la expresión de cláusula WHEN produce true como resultado.

Si se facilita una cláusula WHEN, las declaraciones SQL especificadas como pasos de activación solo se ejecutan en las filas en las que la cláusula WHEN es true. Si no se facilita ninguna cláusula WHEN, las declaraciones SQL se ejecutan en todas las filas.

Dentro del cuerpo de un activador (la cláusula trigger-action), los valores previos y siguientes de cambios de la tabla afectada están disponibles con los nombres especiales de tabla OLD y NEW. La estructura de las tablas OLD y NEW coincide con la estructura de la tabla en la que se crea el activador. La tabla OLD contiene todas las filas modificadas o eliminadas con la declaración del activador en el estado que tenían antes de iniciarse las operaciones de la declaración. La tabla NEW contiene todas las filas modificadas o creadas con la declaración del activador en el estado que tenían después de iniciarse las operaciones de la declaración. Tanto la cláusula WHEN como las declaraciones trigger-step pueden acceder a valores de la fila insertada, eliminada o actualizada mediante referencias expresadas como NEW.column-name y OLD.column-name, donde column-name es el nombre de la columna de la tabla con la que se asocia el activador. La disponibilidad de las referencias a la tabla OLD y NEW depende del tipo de database-event gestionado por el activador:

- INSERT – las referencias NEW son válidas
- UPDATE – las referencias NEW y OLD son válidas
- DELETE – las referencias OLD son válidas

El tiempo especificado (BEFORE, AFTER o INSTEAD OF) determina cuándo se ejecutan las declaraciones trigger-step en relación a la inserción, modificación o eliminación de la fila asociada. Una cláusula ON CONFLICT puede especificarse como parte de una declaración UPDATE o INSERT en un trigger-step. Sin embargo, si se especifica una cláusula ON CONFLICT como parte de la declaración que activa el activador, se utilizará la política de gestión de conflictos.

Además de activadores de tablas, se puede crear un activador INSTEAD OF en una vista. Si se definen uno o varios INSTEAD OF INSERT, INSTEAD OF DELETE o INSTEAD OF UPDATE en una vista, no se considerará un error ejecutar el tipo asociado de la instancia (INSERT, DELETE o UPDATE) en la vista. En ese caso, al ejecutar un INSERT, DELETE o UPDATE en la vista, activa los activadores asociados. Como el activador es un activador INSTEAD OF, las tablas subyacentes de la vista no se modifican con la declaración que activa el activador. Sin embargo, los activadores se pueden utilizar para realizar operaciones de modificación en las tablas subyacentes.

Debe tener algo importante en cuenta cuando cree un activador en una tabla con una columna INTEGER PRIMARY KEY. Si un activador BEFORE modifica la columna INTEGER PRIMARY KEY de una fila que se deba actualizar con la declaración que activa el activador, no se realiza ninguna actualización. Una alternativa es crear la tabla con una columna PRIMARY KEY en vez de con una columna INTEGER PRIMARY KEY.

Un activador se puede eliminar con la declaración DROP TRIGGER. Cuando se visualiza o se descarta una tabla, todos los activadores asociados a la tabla o a la vista también se descartan automáticamente.

Función RAISE ()

Se puede utilizar una función SQL RAISE() especial en la declaración trigger-step de un activador. Esta función tiene la sintaxis siguiente:

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

Cuando se llama a una de las primeras tres expresiones durante la ejecución del activador, la acción de procesamiento ON CONFLICT especificada (ABORT, FAIL o ROLLBACK) se lleva a cabo y finaliza la ejecución de la declaración actual. ROLLBACK se considera un error de ejecución de declaración y la instancia de SQLStatement cuyo método execute() se estaba ejecutando distribuye un evento (SQLErrorEvent.ERROR). El objeto SQLError de la propiedad error del objeto de evento distribuido tiene su propiedad details establecida como el error-message especificado en la función RAISE().

Cuando se llama a RAISE(IGNORE), se abandona el resto del activador actual, la declaración que provocó la ejecución del activador y los activadores posteriores que podrían ejecutarse. No se invierte ningún cambio en la base de datos. Si la declaración que provocó la ejecución del activador es en sí misma un activador, su programa reanuda la ejecución al principio del siguiente paso. Para obtener más información sobre los algoritmos de resolución de conflictos, consulte la sección ON CONFLICT (algoritmos de conflictos).

DROP TRIGGER

La declaración DROP TRIGGER elimina un activador creado por la declaración CREATE TRIGGER.

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

El activador se elimina de la base de datos. Tenga en cuenta que los activadores se descartan automáticamente cuando se descarta su tabla asociada.

Declaraciones y cláusulas especiales

En esta sección se describen las distintas cláusulas que amplían el SQL proporcionado por el motor de ejecución, así como dos elementos del lenguaje que se pueden utilizar en muchas declaraciones, comentarios y expresiones.

COLLATE

La cláusula COLLATE se utiliza en las declaraciones SELECT, CREATE TABLE y CREATE INDEX para especificar el algoritmo de comparación empleado al comparar u ordenar valores.

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

El tipo de intercalación predeterminada de las columnas es BINARY. Cuando se utiliza la intercalación BINARY con valores de la clase de almacenamiento TEXT, se lleva a cabo una intercalación binaria comparando los bytes de la memoria que representan el valor, independientemente de la codificación del texto.

La secuencia de intercalación NOCASE solo se aplica a valores de la clase de almacenamiento TEXT. Cuando se utiliza, la intercalación NOCASE lleva a cabo una comparación sin distinguir entre mayúsculas y minúsculas.

No se utiliza ninguna secuencia de intercalación en las clases de almacenamiento de tipo NULL, BLOB, INTEGER o REAL.

Para utilizar un tipo de intercalación que no sea BINARY con una columna, debe especificar una cláusula COLLATE como parte de la definición de la columna en la declaración CREATE TABLE. Cada vez que se comparan dos valores TEXT, se utiliza una secuencia de intercalación para determinar los resultados de la comparación a partir de las siguientes reglas:

- En los operadores de comparación binarios, si uno de los operandos es una columna, el tipo de intercalación predeterminada de la columna determina la secuencia de intercalación que se utiliza para la comparación. Si ambos operandos son columnas, el tipo de intercalación del operando izquierdo determina la secuencia de intercalación utilizada. Si ninguno de los operandos es una columna, se utiliza la secuencia de intercalación BINARY.
- El operador BETWEEN...AND equivale a utilizar dos expresiones con los operadores >= y <=. Por ejemplo, la expresión x BETWEEN y AND z es equivalente a x >= y AND x <= z. En consecuencia, el operador BETWEEN...AND sigue la regla anterior para determinar la secuencia de intercalación.
- El operador IN se comporta como el operador = en lo que respecta a la determinación de la secuencia de intercalación que se debe utilizar. Por ejemplo, la secuencia de intercalación utilizada para la expresión x IN (y, z) es el tipo de intercalación predeterminado de x si x es una columna. En caso contrario, se utiliza intercalación BINARY.
- Una cláusula ORDER BY que forme parte de una declaración SELECT puede asignarse explícitamente a una secuencia de intercalación para utilizarla en la operación de ordenación. En este caso, siempre se utiliza la secuencia de intercalación explícita. En cualquier otro caso, si la expresión ordenada por una cláusula ORDER BY es una columna, el tipo de intercalación predeterminado de la columna se utilizará para determinar la ordenación. Si la expresión no es una columna, se utiliza la secuencia de intercalación BINARY.

EXPLAIN

El modificador de comandos EXPLAIN es una ampliación no estándar de SQL.

```
sql-statement  ::=  EXPLAIN sql-statement
```

Si la palabra clave EXPLAIN aparece antes de cualquier otra declaración SQL, en vez de ejecutar el comando, en realidad el resultado informa sobre la secuencia de instrucciones de la máquina virtual que se habría utilizado para ejecutar el comando, de no haber estado presente la palabra clave EXPLAIN. La función EXPLAIN es una función avanzada y permite a los desarrolladores cambiar el texto de declaraciones SQL para intentar mejorar el rendimiento o depurar una declaración que parezca no funcionar correctamente.

ON CONFLICT (algoritmos de conflictos)

La cláusula ON CONFLICT no es un comando SQL independiente. Se trata de una cláusula no estándar que puede aparecer en muchos otros comandos SQL.

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

La primera expresión de la cláusula ON CONFLICT utiliza las palabras clave ON CONFLICT y se utiliza en una declaración CREATE TABLE. En el caso de una declaración INSERT o UPDATE, se utiliza la segunda expresión ON CONFLICT sustituida por OR para que la sintaxis sea más natural. Por ejemplo, en vez de INSERT ON CONFLICT IGNORE, la declaración se convierte en INSERT OR IGNORE. Aunque las palabras clave son distintas, el significado de la cláusula es el mismo en ambas expresiones.

La cláusula ON CONFLICT especifica el algoritmo empleado para resolver conflictos de restricciones. Los cinco algoritmos son ROLLBACK, ABORT, FAIL, IGNORE y REPLACE. El algoritmo predeterminado es ABORT. A continuación se explican los cinco algoritmos de conflictos:

ROLLBACK Cuando se produce una violación de restricción, se activa un ROLLBACK inmediatamente y finaliza la transacción actual. El comando se cancela y la instancia de SQLStatement distribuye un evento error. Si no hay ninguna transacción activa (que no sea la transacción implícita creada en cada comando), este algoritmo funciona igual que ABORT.

ABORT Cuando se produce una violación de restricción, el comando descarta todos los cambios anteriores realizados y la instancia de SQLStatement distribuye un evento error. No se ejecuta ROLLBACK, por lo que se conservan los comandos anteriores de una transacción. ABORT es el comportamiento predeterminado.

FAIL Cuando se produce una violación de restricción, el comando se cancela y la instancia de SQLStatement distribuye un evento error. No obstante, todos los cambios realizados por la declaración en la base de datos antes de producirse la violación se conservan y no se descartan. Por ejemplo, si una declaración UPDATE encuentra una violación de restricción en la fila 100 que intenta actualizar, los cambios realizados en las 99 filas anteriores se conservan, pero no así los de la fila 100 y sucesivas.

IGNORE Cuando se produce una violación de restricción, la fila que contiene la violación no se inserta ni se cambia. Salvo el caso omiso que se hace a esta fila, el comando continúa ejecutándose con normalidad. Las filas anteriores y posteriores a la que contiene la violación de restricción se insertan y se actualizan también con normalidad. No se devuelve ningún error.

REPLACE Cuando se produce una violación de restricción UNIQUE, las filas previas existentes que generan la violación se eliminan antes de insertar o actualizar la fila actual. En consecuencia, siempre se produce la inserción y la actualización de filas y el comando sigue ejecutándose con normalidad. No se devuelve ningún error. Si se produce una violación de restricción NOT NULL, el valor NULL se reemplaza por el valor predeterminado de dicha columna. Si la columna no tiene ningún valor predeterminado, se utiliza el algoritmo ABORT. Si se produce una violación de restricción CHECK, se utiliza el algoritmo IGNORE. Cuando esta estrategia de resolución de conflictos elimina filas para poder satisfacer una restricción, no se invocan activadores de eliminación en otras filas.

El algoritmo especificado en la cláusula OR de una declaración INSERT o UPDATE anula cualquier algoritmo especificado en una declaración CREATE TABLE. Si no se especifica ningún algoritmo en la declaración CREATE TABLE o en la declaración INSERT o UPDATE en ejecución, se utiliza el algoritmo ABORT.

REINDEX

El comando REINDEX se utiliza para eliminar y recrear uno o varios índices. El comando resulta útil si ha cambiado la definición de una secuencia de intercalación.

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

En la primera expresión, se recrean todos los índices de las bases de datos asociadas que utilicen la secuencia de intercalación especificada. En la segunda expresión, cuando se especifica un table-name, se reconstruyen todos los índices asociados a la tabla. Si se especifica un index-name, solo se elimina y se recrea el índice especificado.

COMENTARIOS

Los comentarios no son comandos SQL, pero pueden estar presentes en las consultas SQL. El motor de ejecución los trata como espacios en blanco. Pueden empezar en cualquier ubicación de un espacio en blanco, incluidas expresiones al margen que abarquen varias líneas.

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

Un comentario de una sola línea se indica con dos guiones. Un comentario de una sola línea solo llega hasta el final de la línea actual.

Los comentarios en bloque pueden abarcar varias líneas o incorporarse en una sola línea. Si no hay ningún delimitador, un comentario en bloque llega hasta el final de la entrada. Esta situación no se trata como un error. Una nueva declaración SQL puede empezar en una nueva línea detrás del final del comentario en bloque. Los comentarios en bloque pueden incluirse en cualquier espacio en blanco, incluidas expresiones al margen, y en medio de otras declaraciones SQL. Los comentarios en bloque no se pueden anidar. Los comentarios de una sola línea dentro de un comentario en bloque no se tienen en cuenta.

EXPRESIONES

Las expresiones son subcomandos dentro de otros bloques SQL. A continuación se describe la sintaxis válida para una expresión dentro de una declaración SQL:

```
expr            ::=  expr binary-op expr |
                     expr [NOT] like-op expr [ESCAPE expr] |
                     unary-op expr |
                     ( expr ) |
                     column-name |
                     table-name.column-name |
                     database-name.table-name.column-name |
                     literal-value |
                     parameter |
                     function-name( expr-list | * ) |
                     expr ISNULL |
                     expr NOTNULL |
                     expr [NOT] BETWEEN expr AND expr |
                     expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name |
                     [EXISTS] ( select-statement ) |
                     CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                     CAST ( expr AS type ) |
                     expr COLLATE collation-name
like-op         ::=  LIKE | GLOB
binary-op       ::=  see Operators
unary-op        ::=  see Operators
parameter       ::=  :param-name | @param-name | ?
value-list      ::=  literal-value [, literal-value]*
literal-value   ::=  literal-string | literal-number | literal-boolean | literal-blob |
literal-null
literal-string  ::=  'string value'
literal-number  ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

Una expresión es una combinación de valores y operadores que pueden producir como resultado un valor. Las expresiones se pueden dividir en dos tipos generales, en función de si producen un valor booleano (true o false) o un valor no booleano.

En múltiples situaciones habituales, incluidas las cláusulas WHERE y HAVING, la expresión ON en una cláusula JOIN, y una expresión CHECK, la expresión debe producir como resultado un valor booleano. Los siguientes tipos de expresiones cumplen esta condición:

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()
- LIKE
- GLOB
- Ciertas funciones
- Ciertos operadores (concretamente, operadores de comparación)

Valores literales

Un valor numérico literal se escribe como un número entero o un número de coma flotante. Se admite notación científica. El carácter . (punto) siempre se utiliza como punto decimal.

Un literal de cadena se indica encerrando la cadena en comillas simples '. Para incluir una comilla dentro de una cadena, ponga dos comillas sencillas seguidas como en el siguiente ejemplo: ''.

Un literal booleano se indica con el valor true o false. Los valores de literal booleano se utilizan con el tipo de datos de columna Boolean.

Un literal BLOB es un literal de cadena que contiene datos hexadecimales y está precedido de un solo carácter x o X, como X'53514697465'.

Un valor de literal también puede ser el token NULL.

Nombre de columna

Un nombre de columna puede ser cualquiera de los nombres definidos en la declaración CREATE TABLE o uno de los siguientes identificadores especiales: ROWID, OID o _ROWID_. Todos estos identificadores especiales describen la única clave de enteros aleatorios (la "clave de fila") asociada a cada fila de la tabla. Los identificadores especiales solo hacen referencia a la clave de la fila si la declaración CREATE TABLE no define una columna real con el mismo nombre. Las claves de fila se comportan como columnas de solo lectura. Una clave de fila se puede utilizar en cualquier situación en que utilizaría una columna normal, salgo para cambiar el valor de una clave de fila en una declaración UPDATE o INSERT. La declaración SELECT * FROM no incluye la clave de fila en su conjunto de resultados.

Declaración SELECT

Una declaración SELECT puede aparecer en una expresión bien como el operando a la derecha del operador IN, como una cantidad escalar (un valor de resultado único) o como el operando de un operador EXISTS. Cuando se utiliza como una cantidad escalar o como el operando de un operador IN, el SELECT solo puede tener una columna en su resultado. Se permite una declaración SELECT compuesta (conectada con palabras clave como UNION o EXCEPT). Con el operador EXISTS, las columnas del conjunto de resultados de SELECT se ignoran y la expresión devuelve TRUE si existen una o varias filas y devuelve FALSE si el conjunto de resultados está vacío. Si ningún término de la expresión SELECT hace referencia al valor de la consulta que lo contiene, la expresión se calcula una vez antes de cualquier otro procesamiento y, si es necesario, se reutiliza el resultado. Si la expresión SELECT contiene variables de la consulta exterior (denominada subconsulta correlacionada), SELECT se vuelve a calcular cada vez que sea necesario.

Cuando SELECT es el operando a la derecha del operador IN, el operador IN devuielve TRUE si el resultado del operando a la izquierda es igual que cualquiera de los valores del conjunto de resultados de la declaración SELECT. El operador IN puede estar precedido de la palabra clave NOT para invertir el sentido de la prueba.

Cuando aparece un SELECT en una expresión, pero no es un operando a la derecha del operador IN, la primera fila del resultado de SELCT pasa a ser el valor utilizado en la expresión. Si SELECT arroja más de una fila de resultados, se ignoran todas las filas siguientes. Si SELECT no arroja ninguna fila, el valor de SELECT es NULL.

Expresión CAST

Una expresión CAST cambia el tipo de datos del valor especificado al proporcionado. El tipo especificado puede ser cualquier nombre de tipo no vacío válido para el tipo en una definición de columna de la declaración CREATE TABLE. Consulte la sección Compatibilidad de tipos de datos para obtener más información.

Elementos de expresiones adicionales

Los siguientes elementos SQL también se pueden utilizar en expresiones:

- Funciones incorporadas: Funciones de agregación, funciones escalares y funciones de formato de fecha y hora
- Operadores
- Parámetros

Funciones incorporadas

Las funciones incorporadas se pueden dividir en tres categorías principales:

- Funciones de agregación
- Funciones escalares
- Funciones de fecha y hora

Además de estas funciones, existe una función especial RAISE() que se utiliza para notificar sobre el error de ejecución de un activador. Esta función solo se puede utilizar dentro del cuerpo de una declaración CREATE TRIGGER. Para obtener información sobre la función RAISE(), consulte CREATE TRIGGER > RAISE().

Al igual que ocurre con todas las palabras clave de SQL, los nombres de funciones no distinguen entre mayúsculas y minúsculas.

Funciones de agregación

Las funciones de agregación realizan operaciones en valores de varias filas. Estas funciones se utilizan fundamentalmente en declaraciones SELECT en combinación con la cláusula GROUP BY.

AVG(X)	Devuelve el valor promedio de todos los valores X no NULL X dentro de un grupo. Los valores de cadena y de BLOB que no parezcan números se interpretan como 0. El resultado de AVG() siempre es un valor de coma flotante, incluso si todos los datos introducidos son enteros.
COUNT(X) COUNT(*)	La primera expresión devuelve un recuento del número de veces que X es no NULL en un grupo. La segunda expresión (con el argumento *) devuelve el número total de filas del grupo.
MAX(X)	Devuelve el valor máximo de todos los valores del grupo. Se utiliza la ordenación habitual para determinar el máximo.
MIN(X)	Devuelve el valor no NULL mínimo de todos los valores del grupo. Se utiliza la ordenación habitual para determinar el mínimo. Si todos los valores del grupo son NULL, se devuelve NULL.
SUM(X) TOTAL(X)	Devuelve la suma numérica de todos los valores no NULL del grupo. Si todos los valores son NULL, SUM() devuelve NULL, y TOTAL() devuelve 0.0. El resultado de TOTAL() siempre es un valor de coma flotante. El resultado de SUM() es un valor entero si todos los datos no NULL introducidos son enteros. Si cualquier entrada de SUM() no es un entero y no NULL, SUM() devuelve un valor de coma flotante. Este valor puede ser una aproximación de la suma real.

En cualquiera de las funciones de agregación anteriores que toman un solo argumento, dicho argumento puede ir precedido de la palabra clave DISTINCT. En este caso, los elementos duplicados se filtran antes de transferirse a la función de agregación. Por ejemplo, la llamada a la función COUNT(DISTINCT x) devuelve el número de valores distintos de la columna X y no el número total de valores no NULL de la columna x.

Funciones escalares

Las funciones escalares operan con valores, de fila en fila.

ABS(X)	Devuelve el valor absoluto del argumento X.
COALESCE(X, Y, ...)	Devuelve una copia del primer argumento no NULL. Si todos los argumentos son NULL se devuelve NULL. Debe haber al menos dos argumentos.
GLOB(X, Y)	Esta función se utiliza para implementar la sintaxis de X GLOB Y.
IFNULL(X, Y)	Devuelve una copia del primer argumento no NULL. Si ambos argumentos son NULL se devuelve NULL. Esta función se comporta como COALESCE().
HEX(X)	El argumento se interpreta como un valor de tipo de almacenamiento BLOB. El resultado es una representación hexadecimal del contenido de dicho valor.
LAST_INSERT_ROWID( )	Devuelve el identificador de fila (clave principal generada) de la última fila insertada en la base de datos mediante la instancia de SQLConnection actual. Este valor es el mismo que el valor devuelto por la propiedad SQLConnection.lastInsertRowID.
LENGTH(X)	Devuelve la longitud de cadena de X en caracteres.
LIKE(X, Y [, Z])	Esta función se utiliza para implementar la sintaxis de X LIKE Y [ESCAPE Z] de SQL. Si la cláusula ESCAPE opcional está presente, se invoca la función con tres argumentos. En caso contrario, se invoca solo con dos argumentos.
LOWER(X)	Devuelve una copia de la cadena X con todos los caracteres convertidos a minúscula.
LTRIM(X) LTRIM(X, Y)	Devuelve una cadena formada al eliminar espacios desde la izquierda de X. Si se especifica un argumento Y, la función elimina todos los caracteres de Y desde la izquierda de X.
MAX(X, Y, ...)	Devuelve el argumento con el valor máximo. Los argumentos pueden ser cadenas, además de números. El valor máximo se determina con la ordenación definida. Tenga en cuenta que MAX() es una función simple cuando tiene 2 o más argumentos, pero es una función de agregación cuando tiene un solo argumento.
MIN(X, Y, ...)	Devuelve el argumento con el valor mínimo. Los argumentos pueden ser cadenas, además de números. El valor mínimo se determina con la ordenación definida. Tenga en cuenta que MIN() es una función simple cuando tiene 2 o más argumentos, pero es una función de agregación cuando tiene un solo argumento.
NULLIF(X, Y)	Devuelve el primer argumento si los argumentos son distintos; en caso contrario, devuelve NULL.
QUOTE(X)	Esta rutina devuelve una cadena que es el valor de su argumento indicado para su inclusión en otra declaración SQL. Las cadenas se encierran con comillas simples y tantos saltos en las comillas interiores como sea necesario. Las clases de almacenamiento BLOB se codifican como literales hexadecimales. Esta función resulta útil cuando se escriben activadores que implementan funciones de deshacer y rehacer.
RANDOM(*)	Devuelve un entero seudoaleatorio entre -9223372036854775808 y 9223372036854775807. Este valor aleatorio no tiene protección cifrada.
RANDOMBLOB(N)	Devuelve un BLOB de N bytes con bytes seudoaleatorios. N debe ser un entero positivo. Este valor aleatorio no tiene protección cifrada. Si el valor de N es negativo, se devuelve un solo byte.
ROUND(X) ROUND(X, Y)	Redondea el número X a Y dígitos a la derecha del punto decimal. Si se omite el argumento Y, se utiliza 0.
RTRIM(X) RTRIM(X, Y)	Devuelve una cadena formada al eliminar espacios desde la derecha de X. Si se especifica un argumento Y, la función elimina todos los caracteres de Y desde la derecha de X.
SUBSTR(X, Y, Z)	Devuelve una subcadena de la cadena de entrada X que comienza con el carácter número Y de Z caracteres de longitud. El carácter situado más a la izquierda de X es la posición 1 del índice. Si Y es negativo, el primer carácter de la subcadena se busca contando desde la derecha y no desde la izquierda.
TRIM(X) TRIM(X, Y)	Devuelve una cadena formada al eliminar espacios desde la derecha de X. Si se especifica un argumento Y, la función elimina todos los caracteres de Y desde la derecha de X.
TYPEOF(X)	Devuelve el tipo de expresión X. Los posibles valores devueltos son 'null', 'integer', 'real', 'text' y 'blob'. Para obtener más información sobre los tipos de datos, consulte la sección Compatibilidad de tipos de datos.
UPPER(X)	Devuelve una copia de la cadena de entrada X convertida a mayúscula.

ZEROBLOB(N)	Devuelve un BLOB con N bytes de 0x00.

Funciones de formato de fecha y hora

Las funciones de formato de fecha y hora son un grupo de funciones escalares que se utilizan para crear datos de fecha y hora con formato. Tenga en cuenta que estas funciones operan con cadenas y valores numéricos y devuelven este tipo de valores. Estas funciones no están pensadas para utilizarse con el tipo de datos DATE. Si utiliza estas funciones en datos de una columna cuyo tipo de datos declarado sea DATE, no tendrán el comportamiento predeterminado.

DATE(T, ...)	La función DATE() devuelve una cadena con la fecha en este formato: YYYY-MM-DD. El primer parámetro (T) especifica una cadena de tiempo con el formato encontrado en formato Time. Se puede especificar cualquier número de modificadores después de la cadena de tiempo. Puede encontrar los modificadores disponibles en Modificadores.
TIME(T, ...)	La función TIME() devuelve una cadena que contiene la hora en formato HH:MM:SS. El primer parámetro (T) especifica una cadena de tiempo con el formato encontrado en formato Time. Se puede especificar cualquier número de modificadores después de la cadena de tiempo. Puede encontrar los modificadores disponibles en Modificadores.
DATETIME(T, ...)	La función DATETIME() devuelve una cadena que contiene la fecha y la hora en formato AAAA-MM-DD HH:MM:SS. El primer parámetro (T) especifica una cadena de tiempo con el formato encontrado en formato Time. Se puede especificar cualquier número de modificadores después de la cadena de tiempo. Puede encontrar los modificadores disponibles en Modificadores.
JULIANDAY(T, ...)	La función JULIANDAY() devuelve un número que indica el número de días transcurridos entre el mediodía (Greenwich) del 24 de noviembre de 4714 a.C. y la fecha suministrada. El primer parámetro (T) especifica una cadena de tiempo con el formato encontrado en formato Time. Se puede especificar cualquier número de modificadores después de la cadena de tiempo. Puede encontrar los modificadores disponibles en Modificadores.
STRFTIME(F, T, ...)	La rutina STRFTIME() devuelve la fecha según la cadena de formato especificada como el primer argumento F. La cadena de formato admite las siguientes sustituciones: %d: día del mes %f: fracciones de segundo SS.SSS %H: hora 00-24 %j: día del año 001-366 %J: número de día de calendario juliano %m: mes 01-12 %M: minuto 00-59 %s: segundos desde 1970-01-01 %S: segundos 00-59 %w: día de la semana 0-6 (domingo = 0) %W: semana del año 00-53 %Y: año 0000-9999 %% - % El segundo parámetro (T) especifica una cadena de tiempo con el formato encontrado en formato Time. Se puede especificar cualquier número de modificadores después de la cadena de tiempo. Puede encontrar los modificadores disponibles en Modificadores.

Formatos de tiempo

Una cadena de tiempo puede estar en cualquiera de los siguientes formatos:

YYYY-MM-DD	2007-06-15
YYYY-MM-DD HH:MM	2007-06-15 07:30
YYYY-MM-DD HH:MM:SS	2007-06-15 07:30:59
YYYY-MM-DD HH:MM:SS.SSS	2007-06-15 07:30:59.152

YYYY-MM-DDTHH:MM	2007-06-15T07:30
YYYY-MM-DDTHH:MM:SS	2007-06-15T07:30:59
YYYY-MM-DDTHH:MM:SS.SSS	2007-06-15T07:30:59.152
HH:MM	07:30 (la fecha es 2000-01-01)
HH:MM:SS	07:30:59 (la fecha es 2000-01-01)
HH:MM:SS.SSS	07:30:59:152 (la fecha es 2000-01-01)
now	Fecha y hora actual en sistema de coordenadas universal (UCT).
DDDD.DDDD	Número de día de calendario juliano como número de coma flotante.

El carácter T en estos formatos es un carácter literal "T" que separa la fecha y la hora. Los formatos que solo incluyen una hora asumen que la fecha es 2001-01-01.

Modificadores

La cadena de tiempo puede no ir seguida de ningún modificador o de varios modificadores que alteran la fecha o su interpretación. Los modificadores disponibles son:

NNN days	Número de días que se agregan a la hora.
NNN hours	Número de horas que se agregan a la hora.
NNN minutes	Número de minutos que se agregan a la hora.
NNN.NNNN seconds	Número de segundos y milisegundos que se agregan a la hora.
NNN months	Número de meses que se agregan a la hora.
NNN years	Número de años que se agregan a la hora.
start of month	Retrocede al principio del mes.
start of year	Retrocede al principio del año.
start of day	Retrocede al principio del día.
weekday N	Avanza la hora al día de la semana especificado. (0 = domingo, 1 = lunes, etc.).
localtime	Convierte la fecha en hora local.
utc	Convierte la fecha en hora del sistema de coordenadas universal (UTC).

Operadores

SQL admite una amplia selección de operadores, incluidos los operadores habituales que existen en la mayoría de los lenguajes de programación, además de otros operadores exclusivos de SQL.

Operadores habituales

Los siguientes operadores binarios se permiten en un bloque SQL y se muestran ordenados de mayor a menor precedencia:

```
*     /     %
+     -
<<  >>  &  |
<  >=    >  >=
=     ==    !=    <>  IN
AND
OR
```

Los operadores de prefijo unarios admitidos son:

```
 !     ~     NOT
```

El operador COLLATE se puede entender como un operador de postfijo unario. El operador COLLATE tiene la precedencia mayor. Siempre se vincula con mayor fuerza que cualquier operador de prefijo unario u operador binario.

Tenga en cuenta que existen dos variaciones de los operadores de igualdad y de desigualdad. El operador de igualdad puede ser = o ==. El operador de desigualdad puede ser != o <>.

El operador || es el operador de concatenación de cadenas: une las dos cadenas y sus operandos.

El operador % genera el resto de su módulo de entre el operando izquierdo y el derecho.

El resultado de cualquier operador binario es un valor numérico, excepto el operador de concatenación ||, que produce un resultado de cadena.

Operadores SQL

LIKE

El operador LIKE lleva a cabo una comparación de coincidencia de patrones.

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

El operando a la derecha del operador LIKE contiene el patrón y el operando de la izquierda contiene la cadena que debe comparar con el patrón. Un símbolo de porcentaje (%) en el patrón es un carácter comodín: busca cada secuencia de cero o más caracteres de la cadena y la compara. Un guion bajo (_) en el patrón busca coincidencias de cualquier carácter de la cadena. Cualquier otro carácter se compara consigo mismo o con su equivalente en minúsculas/mayúsculas, es decir, se comparan las búsquedas sin distinción entre mayúsculas y minúsculas. (Nota: el motor de base de datos solo puede leer caracteres en mayúscula/minúscula Latin de 7 bits. En consecuencia, el operador LIKE distingue entre mayúsculas y minúsculas en caracteres iso8859 de 8 bits o en caracteres UTF-8. Por ejemplo, la expresión 'a' LIKE 'A' es TRUE pero 'æ' LIKE 'Æ' es FALSE). La distinción entre mayúsculas y minúsculas en caracteres Latin se puede cambiar con la propiedad SQLConnection.caseSensitiveLike.

Si la cláusula ESCAPE opcional está presente, la expresión que sigue a la palabra clave ESCAPE debe producir como resultado una cadena con un solo carácter. Este carácter se puede utilizar en el patrón LIKE para coincidir con el porcentaje literal o con los caracteres de guion bajo. El carácter de escape seguido de un símbolo de porcentaje, un guion bajo o el mismo carácter coincide con un símbolo de porcentaje literal, con un guion bajo o con un carácter de escape de la cadena, respectivamente.

GLOB

El operador GLOB es parecido a LIKE pero utiliza la sintaxis de glob de archivos Unix para los comodines. Al contrario que LIKE, GLOB sí distingue entre mayúsculas y minúsculas.

IN

El operador IN calcula si su operando izquierdo es igual a los valores de su operando derecho (un conjunto de valores entre paréntesis).

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

El operando derecho puede ser un conjunto de valores literales separados por comas, o el resultado de una declaración SELECT. Consulte las declaraciones SELECT de las expresiones para ver una explicación y las limitaciones de una declaración SELECT como operando derecho del operador IN.

BETWEEN...AND

El operador BETWEEN...AND equivale a utilizar dos expresiones con los operadores >= y <=. Por ejemplo, la expresión x BETWEEN y AND z es equivalente a x >= y AND x <= z.

NOT

El operador NOT es un operador de negación. Los operadores GLOB, LIKEe IN pueden ir precedidos de la palabra clave NOT para invertir el sentido de la prueba (dicho de otro modo, para comprobar que un valor no coincide con el patrón indicado).

Parámetros

Un parámetro especifica un marcador de posición en la expresión para un valor literal que se llena en tiempo de ejecución asignándole un valor para el conjunto asociativo SQLStatement.parameters. Los parámetros pueden expresarse de tres formas:

?	Un signo de interrogación indica un parámetro indexado. A los parámetros se les asignan valores de índice numéricos (basados en cero) según su orden en la declaración.
:AAAA	Dos puntos seguidos de un nombre de identificador indica un parámetro con el nombre AAAA. Los parámetros especificados también se numeran según su orden en la declaración SQL. Para evitar confusiones, lo mejor es evitar mezclar parámetros con nombre y con número.
@AAAA	Un "símbolo de arroba" equivale a dos puntos.

Funciones SQL no admitidas

A continuación se incluye una lista con los elementos SQL estándar no admitidos en Adobe AIR:

Restricciones de FOREIGN KEY Las restricciones de FOREIGN KEY se analizan pero no se fuerzan.

Activadores Los activadores FOR EACH STATEMENT no se admiten (todos los activadores deben ser FOR EACH ROW). Los activadores INSTEAD OF no se admiten en tablas (los activadores INSTEAD OF solo se permiten en vistas). Activadores recurrentes: los activadores que se autoactivan no se admiten.

ALTER TABLE Solo se admiten las variantes RENAME TABLE y ADD COLUMN del comando ALTER TABLE. Otros tipos de operaciones ALTER TABLE como DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT, etc., se omiten.

Transacciones anidadas Solo se permite una sola transacción activa.

RIGHT y FULL OUTER JOIN RIGHT OUTER JOIN o FULL OUTER JOIN no se admiten.

VIEW actualizable Una vista es de solo lectura. No se puede ejecutar una declaración DELETE, INSERT o UPDATE en una vista. Un activador INSTEAD OF que se activa para intentar aplicar un DELETE, INSERT o UPDATE en una vista sí se admite y se puede utilizar para actualizar las tablas admitidas en el cuerpo del activador.

GRANT y REVOKE Una base de datos es un archivo de disco normal; los únicos permisos de acceso que se pueden aplicar son los habituales de acceso de archivos del sistema operativo instalado. Los comandos GRANT y REVOKE que suelen estar en RDBMS de cliente/servidor no se implementan.

Los siguientes elementos SQL y funciones SQLite se admiten en algunas implementaciones de SQLite, pero no en Adobe AIR. La mayor parte de su funcionalidad está disponible mediante métodos de la clase SQLConnection:

Elementos SQL relativos a la transacción (BEGIN, END, COMMIT, ROLLBACK) Esta funcionalidad está disponible mediante métodos de la clase SQLConnection, relativos a la transacción: SQLConnection.begin(), SQLConnection.commit() y SQLConnection.rollback().

ANALYZE Esta funcionalidad está disponible mediante el método SQLConnection.analyze().

ATTACH Esta funcionalidad está disponible mediante el método SQLConnection.attach().

COPY Esta declaración no se admite.

CREATE VIRTUAL TABLE Esta declaración no se admite.

DETACH Esta funcionalidad está disponible mediante el método SQLConnection.detach().

PRAGMA Esta declaración no se admite.

VACUUM Esta funcionalidad está disponible mediante el método SQLConnection.compact().

El acceso a la tabla del sistema no está disponible Las tablas del sistema, incluida sqlite_master y otras tablas con el prefijo "sqlite_" no están disponibles en las declaraciones SQL. El motor de ejecución contiene una API de esquema que proporciona un método orientado a objetos para acceder a los datos del esquema. Para obtener más información, consulte el método SQLConnection.loadSchema().

Funciones de expresiones regulares (MATCH() y REGEX()) Estas funciones no están disponibles en declaraciones SQL.

La siguiente funcionalidad no coincide en muchas implementaciones de SQLite y Adobe AIR:

Parámetros de declaración indexados En muchas implementaciones, los parámetros indexados están basados en uno. Sin embargo, en Adobe AIR, los parámetros de declaración indexados están basados en cero (es decir, el primer parámetro tiene el índice 0, el segundo parámetro tiene el índice 1, etc.

Definiciones de columnas INTEGER PRIMARY KEY En muchas implementaciones, solo las columnas definidas exactamente como INTEGER PRIMARY KEY se utilizan como columna de clave principal real para una tabla. En dichas implementaciones, si se utiliza otro tipo de datos (normalmente un equivalente para INTEGER (como int), la columna no se utiliza como clave principal interna. Sin embargo, en Adobe AIR, el tipo de datos int (y otros sinónimos de INTEGER) se consideran exactamente equivalentes a INTEGER. En consecuencia, se utiliza una columna definida como int PRIMARY KEY como clave interna para la tabla. Para obtener más información, consulte las secciones CREATE TABLE y Afinidad de columnas.

Funciones SQL adicionales

Los siguientes tipos de afinidad de columnas no se admiten de forma predeterminada en SQLite, pero sí en Adobe AIR (Tenga en cuenta que, como ocurre con todas las palabras clave de SQL, estos nombres de tipos de datos no distinguen entre mayúsculas y minúsculas):

Boolean correspondiente a la clase Boolean.

Date correspondiente a la clase Date.

int correspondiente a la clase int (equivalente a la afinidad de columna INTEGER).

Number correspondiente a la clase Number (equivalente a la afinidad de columna REAL).

Object correspondiente a la clase Object o a cualquier subclase que pueda serializarse y deserializarse con AMF3. (Esto incluye la mayoría de las clases con clases personalizadas, pero excluye algunas clases con objetos de visualización que contengan propiedades como objetos de visualización.)

String correspondiente a la clase String (equivalente a la afinidad de columna TEXT).

XML correspondiente ala clase XML de ActionScript (E4X).

XMLList correspondiente a la clase XMLList de ActionScript (E4X).

Los siguientes valores literales no se admiten de forma predeterminada en SQLite, pero sí en Adobe AIR:

true se utiliza para representar el valor booleano literal true, para trabajar con columnas BOOLEAN.

false se utiliza para representar el valor booleano literal false, para trabajar con columnas BOOLEAN.

Compatibilidad de tipos de datos

Al contrario de lo que ocurre con la mayoría de bases de datos SQL, el motor de base de datos SQL de Adobe AIR no exige que las columnas de las tablas contengan valores de tipos concretos. El motor de ejecución utiliza dos conceptos, las clases de almacenamiento y la afinidad de columnas, para controlar los tipos de datos. En esta sección se describen las clases de almacenamiento y la afinidad de columnas, así como el modo en que se resuelven distintos tipos de datos en diversas condiciones:

- "Clases de almacenamiento" en la página 1177
- "Afinidad de columnas" en la página 1178
- "Tipos de datos y operadores de comparación" en la página 1180
- "Tipos de datos y operadores matemáticos" en la página 1181
- "Tipos de datos y ordenación" en la página 1181
- "Tipos de datos y agrupación" en la página 1181
- "Tipos de datos y declaraciones SELECT compuestas" en la página 1181

Clases de almacenamiento

Las clases de almacenamiento representan los tipos de datos reales que se utilizan para guardar valores en una base de datos. Se utilizan las siguientes clases de almacenamiento en la base de datos:

NULL El valor es un valor NULL.

INTEGER El valor es un entero con signo.

REAL El valor es un valor numérico de coma flotante.

TEXT El valor es una cadena de texto (limitada a 256 MB).

BLOB El valor es un objeto grande binario (BLOB); dicho de otro modo, datos binarios sin procesar (limitados a 256 MB).

Todos los valores suministrados a la base de datos como literales incorporados en una declaración SQL o valores ligados al uso de parámetros en una declaración SQL preparada se asignan a una clase de almacenamiento antes de que se ejecute la declaración SQL.

A los literales que forman parte de una declaración SQL se les asigna una clase de almacenamiento TEXT si están encerrados con comillas simples o dobles, una clase INTEGER si el literal está especificado como un número sin comillas, sin punto decimal ni exponente, una clase REAL si el literal es un número sin comillas, con punto decimal o exponente y una clase de almacenamiento NULL si el valor es NULL. Los literales con clase de almacenamiento BLOB se especifican con la notación X'ABCD'. Para obtener más información, consulte Valores literales en expresiones.

Los valores suministrados como parámetros con el conjunto asociativo SQLStatement.parameters se asignan a la clase de almacenamiento que mejor coincidencia tenga con el tipo de datos. Por ejemplo, los valores int se vinculan a una clase de almacenamiento INTEGER, los valores Number a la clase REAL, los valores String a la clase de almacenamiento TEXT y los objetos ByteArray a la clase de almacenamiento BLOB.

Afinidad de columnas

La *afinidad* de una columna es el tipo recomendado para los datos guardados en dicha columna. Cuando se guarda un valor en una columna (mediante una declaración INSERT o UPDATE), el motor de ejecución intenta convertir el valor de su tipo de datos en la afinidad especificada. Por ejemplo, si se inserta un valor Date (instancia de Date en ActionScript o JavaScript) en una columna cuya afinidad es TEXT, el valor Date se convertirá a la representación de String (equivale a llamar al método toString() del objeto) antes de guardarlo en la base de datos. Si el valor no se puede convertir a la afinidad especificada, se produce un error y la operación no se lleva a cabo. Cuando se recupera un valor de la base de datos mediante una declaración SELECT, se devuelve como una instancia de la clase correspondiente a la afinidad, independientemente de si se convirtió a un tipo de datos distinto al guardarse.

Si una columna acepta valores NULL, el valor null de ActionScript o JavaScript se puede utilizar como valor de parámetros para almacenar NULL en la columna. Cuando se recupera una clase de almacenamiento NULL de una declaración SELECT, siempre se devuelve como el valor null de ActionScript o JavaScript, independientemente de la afinidad de la columna. Si una columna acepta valores NULL, siempre comprueba los valores recuperados de dicha columna para determinar si son null antes de intentar convertir los valores a un tipo que no pueda ser NULL (como Number o Boolean).

A cada columna de la base de datos se le asigna uno de los siguientes tipos de afinidad:

- TEXT (o String)
- NUMERIC
- INTEGER (o int)
- REAL (o Number)
- Boolean
- Date
- XML
- XMLLIST
- Object
- NONE

TEXT (o String)

Una columna con afinidad TEXT o String almacena todas las clases de almacenamiento NULL, TEXT o BLOB. Si se insertan datos numéricos en una columna con afinidad TEXT, se convierten a texto antes de guardarlos.

NUMERIC

Una columna con afinidad NUMERIC contiene valores que utilizan clases de almacenamiento NULL, REAL o INTEGER. Cuando se insertan datos de texto en una columna NUMERIC, se intenta convertir en un número entero o real antes de guardarlo. Si la conversión se realiza correctamente, el valor se guarda con la clase de almacenamiento INTEGER o REAL (por ejemplo, un valor de '10.05' se convierte a una clase de almacenamiento REAL antes de ordenarlo). Si no se puede realizar la conversión, se produce un error. No se intenta convertir a un valor NULL. Se devuelve un valor recuperado de una columna NUMERIC como una instancia del tipo numérico más específico con el que encaje el valor. Dicho de otro modo, si el valor es un entero positivo o 0, se devuelve como una instancia de uint. Si es un entero negativo, se devuelve como instancia de int. Finalmente, si tiene un componente de coma flotante (no es un entero), se devuelve como instancia de Number.

INTEGER (o int)

Una columna que utiliza afinidad INTEGER se comporta del mismo modo que una columna con afinidad NUMERIC, con una excepción. Si el valor se va a guardar como un valor real (instancia de Number) sin componente de coma flotante o si el valor es un valor de texto que se puede convertir a un valor real sin componente de coma flotante, se convierte a un entero y se guarda utilizando la clase de almacenamiento INTEGER. Si se intenta guardar un valor real con un componente de coma flotante se produce un error.

REAL (o Number)

Una columna con afinidad REAL o NUMBER se comporta como una columna con afinidad NUMERIC con la diferencia de que fuerza los valores enteros a una representación de coma flotante. Siempre se devuelve un valor de una columna REAL de la base de datos como una instancia de Number.

Boolean

Una columna con afinidad Boolean almacena valores true o false. Una columna Boolean acepta un valor que es una instancia de Boolean en ActionScript o JavaScript. Si el código intenta guardar un valor String, un elemento String con longitud mayor que cero se considerará true y un elemento vacío String se considerará false. Si el código intenta guardar datos numéricos, se guardará cualquier valor distinto de cero como true y 0 se guardará como false. Cuando se recupera un valor Boolean con una declaración SELECT, se devuelve como instancia de Boolean. Los valores no NULL se guardan con la clase de almacenamiento INTEGER (0 para false; 1 para true) y se convierten en objetos Boolean cuando se recuperan los datos.

Date

Una columna con afinidad Date almacena valores de fecha y hora. Una columna Date está diseñada para aceptar valores que sean instancias de Date de ActionScript o JavaScript. Si se intenta guardar un valor String en una columna Date, el motor de ejecución intentará convertirlo en una fecha de calendario juliano. Si falla la conversión, se produce un error. Si el código intenta guardar un valor Number, int o uint, no se realiza ningún intento para validar los datos y se asume que se trata de un valor de fecha de calendario juliano correcto. Un valor Date recuperado con una declaración SELECT se convierte automáticamente en una instancia de Date. Los valores de DATE se guardan como valores de fecha de calendario juliano con la clase de almacenamiento REAL. Por ello, las operaciones de ordenación y comparación funcionan del modo esperado.

XML o XMLList

Una columna que utiliza afinidad XML o XMLList almacena estructuras XML. Cuando el código intenta guardar datos en una columna XML con un parámetro SQLStatement, el motor de ejecución intenta convertir y validar el valor con la función XML() o XMLList() de ActionScript. Si el valor no se puede convertir a XML válido, se produce un error. Si se intenta guardar los datos con un valor de texto SQL literal (por ejemplo, INSERT INTO (col1) VALUES ('Invalid XML (no closing tag)'), el valor no se analiza ni se valida: se asume que está bien expresado. Si se guarda un valor no válido, se devuelve un objeto vacío XML al recuperarlo. Los datos XML y XMLList se almacenan con la clase de almacenamiento TEXT o NULL.

Object

Una columna con afinidad Object almacena objetos complejos de ActionScript o JavaScript, incluidas instancias de la clase Object, así como instancias de subclases de Object como instancias de Array e incluso instancias de clases personalizadas. Los datos de la columna OBJECT están serializados en formato AMF3 y se guardan con la clase de almacenamiento BLOB. Cuando se recupera una valor, se deserializa de AMF3 y se devuelve como una instancia de la clase tal como se guardó. Tenga en cuenta que algunas clases de ActionScript, especialmente los objetos de visualización, no se pueden deserializar como instancias de su tipo de datos original. Antes de guardar una instancia de clase personalizada, debe registrar un apodo para la clase con el método flash.net.registerClassAlias() (o, en Flex,

agregando metadatos [RemoteObject] a la declaración de la clase). Asimismo, antes de recuperar los datos debe registrar el mismo apodo para la clase. Cualquier dato que no pueda deserializarse correctamente, bien porque la clase no se puede deserializar inherentemente o porque falta o no coincide un apodo de la clase, se devuelve como un objeto anónimo (una instancia de la clase Object con propiedades y valores que se corresponden con la instancia original tal como se guardó.

NONE

Una columna con afinidad NONE no tiene preferencia sobre una clase de almacenamiento frente a otra. No intenta convertir los datos antes de insertarlos.

Determinación de la afinidad

El tipo de afinidad de una columna se determina mediante el tipo declarado de la columna de la declaración CREATE TABLE. Al determinar el tipo, se aplican las siguientes reglas (no distinguen entre mayúsculas y minúsculas):

- Si el tipo de datos de la columna contiene alguna de las cadenas "CHAR", "CLOB", "STRI" o "TEXT", dicha columna tiene afinidad TEXT/String. Tenga en cuenta que el tipo VARCHAR contiene la cadena "CHAR" y, por lo tanto, se le asigna afinidad TEXT.
- Si el tipo de datos de la columna contiene la cadena "BLOB" o si no hay ningún tipo de datos especificado, la columna tiene afinidad NONE.
- Si el tipo de datos de la columna contiene la cadena "XMLL" la columna tiene afinidad XMLList.
- Si el tipo de datos es la cadena "XML", la columna tiene afinidad XML.
- Si el tipo de datos contiene la cadena "OBJE", la columna tiene afinidad Object.
- Si el tipo de datos contiene la cadena "BOOL", la columna tiene afinidad Boolean.
- Si el tipo de datos contiene la cadena "DATE", la columna tiene afinidad de datos Date.
- Si el tipo de datos contiene la cadena "INT" (incluida "UINT"), se le asigna afinidad INTEGER/int.
- Si el tipo de datos contiene cualquiera de las cadenas "REAL", "NUMB", "FLOA" o "DOUB", la columna tiene afinidad REAL/Number.
- En caso contrario, la afinidad es NUMERIC.
- Si se crea una tabla con una declaración CREATE TABLE t AS SELECT... ninguna columna tendrá un tipo de datos especificado y se le dará la afinidad NONE.

Tipos de datos y operadores de comparación

Se admiten los siguientes operadores de comparación binarios =, <, <=, >= y != además de una operación para probar la pertenencia, IN y el operador de comparación ternario BETWEEN. Para obtener más información sobre estos operadores, consulte Operadores.

Los resultados de una comparación dependen de las clases de almacenamiento de los dos valores que se comparan. Cuando se comparan dos valores, se aplican las siguientes reglas:

- Un valor con clase de almacenamiento NULL sed considera menor que cualquier otro valor (incluidos valores con clase de almacenamiento NULL).
- Un valor INTEGER o REAL es menor que cualquier otro valor TEXT o BLOB. Cuando se compara un INTEGER o un REAL con otro INTEGER o REAL, se lleva a cabo una comparación numérica.
- Un valor TEXT es menor que un valor BLOB. Cuando se comparan dos valores TEXT, se lleva a cabo una comparación binaria.
- Cuando se comparan dos valores BLOB, el resultado siempre se determina mediante una comparación binaria.

El operador ternario BETWEEN siempre se recrea como la expresión binaria equivalente. Por ejemplo, BETWEEN b AND c se recrea como >= b AND a <= c, incluso si esto significa que se deben aplicar distintas afinidades a IN en cada una de las comparaciones necesarias para producir la expresión.

Las expresiones de tipo a IN (SELECT b ....) se gestionan mediante tres reglas enumeradas previamente para comparaciones binarias, es decir, de un modo similar a a = b. Por ejemplo, si b es un valor de columna y a es una expresión, la afinidad de b se aplica a a antes de realizar ninguna comparación. La expresión a IN (x, y, z) se recrea como a = +x OR a = +y OR a = +z. Los valores a la derecha del operador IN (los valores x, y y z en este ejemplo) se consideran expresiones incluso si se encuentran en los valores de columna. Si el valor a la izquierda del operador IN es una columna, se utiliza la afinidad de dicha columna. Si el valor es una expresión, no se produce ninguna conversión.

El modo en que se realizan las comparaciones también se puede ver afectado al utilizar una cláusula COLLATE. Para obtener más información, consulte COLLATE.

Tipos de datos y operadores matemáticos

Para cada uno de los operadores matemáticos admitidos, *, /, %, + and -, se aplica la afinidad numérica a todos los operandos antes de evaluar la expresión. Si algún operando no se puede convertir a la clase de almacenamiento NUMERIC correctamente, la expresión produce como resultado NULL.

Cuando se utiliza el operador de concatenación ||, cada operando se convierte a la clase de almacenamiento TEXT antes de evaluar la expresión. Si algún operando no se puede convertir a la clase de almacenamiento TEXT, el resultado de la expresión es NULL. Esta imposibilidad de convertir el valor puede aparecer en dos situaciones: si el valor del operando es NULL o si se trata de un BLOB que contiene una clase de almacenamiento no TEXT.

Tipos de datos y ordenación

Cuando se ordenan valores mediante una cláusula ORDER BY los valores con clase de almacenamiento NULL tienen preferencia. Luego van los valores INTEGER y REAL, en orden numérico, seguidos de los valores TEXT en orden binario o basados en la intercalación especificada (BINARY o NOCASE). Por último, van los valores BLOB en orden binario. No se producen conversiones de clase de almacenamiento antes de la ordenación.

Tipos de datos y agrupación

Cuando se agrupan valores con la cláusula GROUP BY, los valores con distintas clases de almacenamiento se consideran diferentes. Una excepción son los valores INTEGER y REAL, que se consideran iguales aunque sean equivalentes numéricamente. No se aplica ninguna afinidad a ningún valor como resultado de una cláusula GROUP BY.

Tipos de datos y declaraciones SELECT compuestas

Los operadores SELECT compuestosUNION, INTERSECT y EXCEPT llevan a cabo comparaciones implícitas entre valores. Antes de realizar estas comparaciones, es posible aplicar una afinidad a cada valor. Esa misma afinidad (si existe) se aplica a todos los valores que se puedan devolver en una sola columna del conjunto de resultados del SELECT compuesto. La afinidad aplicada es la de la columna devuelta por la declaración SELECT del primer componente que tenga un valor de columna (y no cualquier otra expresión) en dicha posición. Si, en una columna SELECT compuesta dada, ninguna de las declaraciones SELECT del componente devuelven un valor de columna, no se aplica ninguna afinidad a los valores de dicha columna antes de la comparación.

Capítulo 67: Mensajes detallados de errores SQL, identificadores y argumentos

La clase SQLError representa diversos errores que pueden producirse al trabajar con una base de datos SQL local de Adobe AIR. Para cada excepción, la instancia de SQLError tiene una propiedad `details` que contiene un mensaje de error en español. Además, cada mensaje de error tiene asociado un identificador exclusivo disponible en la propiedad `detailID` del objeto SQLError. Si se utiliza la propiedad `detailID`, una aplicación puede identificar el mensaje de error de details específico. La aplicación puede proporcionar texto alternativo al usuario final en el idioma de su configuración regional. Los valores de los argumentos del conjunto `detailArguments` se pueden sustituir en la posición adecuada de la cadena del mensaje de error. Esto resulta útil en aplicaciones que quieran mostrar el mensaje de error de la propiedad `details` para este error directamente a los usuarios finales en una configuración regional concreta.

La siguiente tabla contiene una lista con los valores de `detailID` y el texto del mensaje de error asociado en español. Los marcadores de posición de los mensajes indican dónde se sustituirán los valores de `detailArguments` en tiempo de ejecución. Esta lista se puede utilizar como referencia para localizar los mensajes de error que puedan producirse durante las operaciones de la base de datos SQL.

ID detallado de error SQL	Parámetros y detalles del mensaje de error en inglés y traducción al español entre paréntesis
1001	Connection closed. (Conexión cerrada.)
1102	Database must be open to perform this operation. (Debe estar abierta una base de datos para poder realizar esta operación.)
1003	%s [,|and %s] parameter name(s) found in parameters property but not in the SQL specified. (Nombre(s) de parámetro %s [,|y %s] encontrado(s) en la propiedad parameters pero no en el SQL especificado.)
1004	Mismatch in parameter count. (No coincide el recuento de parámetros.) Found %d in SQL specified and %d value(s) set in parameters property. (%d encontrado en el SQL especificado y %d valor(es) definidos en la propiedad parameters.) Expecting values for %s [,|and %s]. (Esperando valores para %s [,|y %s].)
1005	Auto compact could not be turned on. (No se pudo activar la compactación automática.)
1006	The pageSize value could not be set. (No se pudo definir el valor de pageSize.)
1007	The schema object with name '%s' of type '%s' in database '%s' was not found. (No se encontró el objeto de esquema con nombre '%s' de tipo '%s' en la base de datos '%s'.)
1008	The schema object with name '%s' in database '%s' was not found. (No se encontró el objeto de esquema con nombre '%s' en la base de datos '%s'.)
1009	No schema objects with type '%s' in database '%s' were found. (No se encontró ningún objeto de esquema con tipo '%s' en la base de datos '%s'.)
1010	No schema objects in database '%s' were found. (No se encontró ningún objeto de esquema en la base de datos '%s'.)
2001	Parser stack overflow. (Desbordamiento de pila de analizador.)
2002	Too many arguments on function '%s'. (Demasiados argumentos en la función '%s'.)
2003	near '%s': syntax error (cerca de '%s': error de sintaxis)
2004	there is already another table or index with this name: '%s'. (Ya hay otra tabla o índice con este nombre: '%s'.)
2005	PRAGMA is not allowed in SQL. (No se permite PRAGMA en SQL.)
2006	Not a writable directory. (No es un directorio de escritura.)
2007	Unknown or unsupported join type: '%s %s %s'. (Tipo de unión desconocida o no admitida: '%s %s %s'.)
2008	RIGHT and FULL OUTER JOINs are not currently supported. (RIGHT y FULL OUTER JOIN no se admiten actualmente.)
2009	A NATURAL join may not have an ON or USING clause. (Una unión NATURAL no puede tener una cláusula ON o USING.)

2010	Cannot have both ON and USING clauses in the same join. (No se pueden tener las cláusulas ON y USING en la misma unión.)
2011	Cannot join using column '%s' - column not present in both tables. (No se puede unir con la columna '%s'; la columna no está presente en ambas tablas.)
2012	Only a single result allowed for a SELECT that is part of an expression. (Solo un único resultado permitido para un SELECT que forme parte de una expresión.)
2013	No such table: '[%s.]%s'. (No existe dicha tabla: '[%s.]%s'.)
2014	No tables specified. (No se ha especificado ninguna tabla.)
2015	Too many columns in result set|too many columns on '%s'. (Demasiadas columnas en el conjunto de resultados|demasiadas columnas en '%s'.)
2016	%s ORDER|GROUP BY term out of range - should be between 1 and %d. (Término %s ORDER|GROUP BY fuera de rango; debe estar entre 1 y %d.)
2017	Too many terms in ORDER BY clause. (Demasiados términos en la cláusula ORDER BY.)
2018	%s ORDER BY term out of range - should be between 1 and %d. (Término %s ORDER BY fuera de rango; debe estar entre 1 y %d.)
2019	%r ORDER BY term does not match any column in the result set. (Término %r ORDER BY no coincide con ninguna columna del conjunto de resultados.)
2020	ORDER BY clause should come after '%s' not before. (La cláusula ORDER BY debe ir detrás de '%s', no delante.)
2021	LIMIT clause should come after '%s' not before. (La cláusula LIMIT debe ir detrás de '%s', no delante.)
2022	SELECTs to the left and right of '%s' do not have the same number of result columns. (Los SELECT a la izquierda y a la derecha de '%s' do no tienen el mismo número de columnas de resultados.)
2023	A GROUP BY clause is required before HAVING. (Se requiere una cláusula GROUP BY antes de HAVING.)
2024	Aggregate functions are not allowed in the GROUP BY clause. (La cláusula GROUP BY no admite funciones de agregación.)
2025	DISTINCT in aggregate must be followed by an expression. (El DISTINCT de la agregación debe ir seguido de una expresión.)
2026	Too many terms in compound SELECT. (Demasiados términos en el SELECT compuesto.)
2027	Too many terms in ORDER|GROUP BY clause. (Demasiados términos en la cláusula ORDER|GROUP BY.)
2028	Temporary trigger may not have qualified name. (El activador temporal podría no tener un nombre cualificado.)
2030	Trigger '%s' already exists. (El activador '%s' ya existe.)
2032	Cannot create BEFORE|AFTER trigger on view: '%s'. (No se puede crear activador BEFORE|AFTER en la vista: '%s'.)
2033	Cannot create INSTEAD OF trigger on table: '%s'. (No se puede crear activador INSTEAD OF en la tabla: '%s'.)
2034	No such trigger: '%s'. (No existe dicho activador: '%s'.)
2035	Recursive triggers not supported ('%s'). (No se admiten activadores recurrentes ('%s').)
2036	No such column: %s[. (No existe dicha columna: %s[..)%s[.%s]]
2037	VACUUM is not allowed from SQL. (No se permite VACUUM desde SQL.)
2043	Table '%s': indexing function returned an invalid plan. (Tabla '%s': la función de indexado devolvió un plan no válido.)
2044	At most %d tables in a join. (En la mayoría de las tablas %d de una unión.)
2046	Cannot add a PRIMARY KEY column. (No se puede añadir una columna PRIMARY KEY.)
2047	Cannot add a UNIQUE column. (No se puede añadir una columna UNIQUE.)
2048	Cannot add a NOT NULL column with default value NULL. (No se puede añadir una columna NOT NULL con valore predeterminado NULL.)
2049	Cannot add a column with non-constant default. (No se puede añadir una columna con valor predeterminado no constante.)
2050	Cannot add a column to a view. (No se puede añadir una columna a una vista.)
2051	ANALYZE is not allowed in SQL. (No se permite ANALYZE en SQL.)
2052	Invalid name: '%s'. (Nombre no válido: '%s'.)
2053	ATTACH is not allowed from SQL. (No se permite ATTACH desde SQL.)

2054	%s '%s' cannot reference objects in database '%s'. (%s '%s' no puede hacer referencia a objetos de la base de datos '%s'.)
2055	Access to '[%s.]%s.%s' is prohibited. (El acceso a '[%s.]%s.%s' está prohibido.)
2056	Not authorized. (No autorizado.)
2058	No such view: '[%s.]%s'. (No existe dicha vista: '[%s.]%s'.)
2060	Temporary table name must be unqualified. (El nombre de la tabla temporal debe ser incompleto.)
2061	Table '%s' already exists. (La tabla '%s' ya existe.)
2062	There is already an index named: '%s'. (Ya hay un índice llamado: '%s'.)
2064	Duplicate column name: '%s'. (Nombre duplicado de columna: '%s'.)
2065	Table '%s' has more than one primary key. (La tabla '%s' tiene más de una clave principal.)
2066	AUTOINCREMENT is only allowed on an INTEGER PRIMARY KEY. (AUTOINCREMENT solo se permite en un INTEGER PRIMARY KEY.)
2067	No such collation sequence: '%s'. (No existe dicha secuencia de intercalación: '%s'.)
2068	Parameters are not allowed in views. (No se permiten parámetros en las vistas.)
2069	View '%s' is circularly defined. (La vista '%s' está definida circularmente.)
2070	Table '%s' may not be dropped. (La tabla '%s' no se puede descartar.)
2071	Use DROP VIEW to delete view '%s'. (Utilice DROP VIEW para eliminar la vista '%s'.)
2072	Use DROP TABLE to delete table '%s'. (Utilice DROP TABLE para eliminar la tabla '%s'.)
2073	Foreign key on '%s' should reference only one column of table '%s'. (La clave externa de '%s' debería hacer referencia a una sola columna de la tabla '%s'.)
2074	Number of columns in foreign key does not match the number of columns in the referenced table. (El número de columnas de la clave externa no coincide con el número de columnas de la tabla de referencia.)
2075	Unknown column '%s' in foreign key definition. (Columna desconocida '%s' en definición de clave externa.)
2076	Table '%s' may not be indexed. (La tabla '%s' no se puede indexar.)
2077	Views may not be indexed. (Las vistas no se pueden indexar.)
2080	Conflicting ON CONFLICT clauses specified. (Cláusulas ON CONFLICT conflictivas especificadas.)
2081	No such index: '%s'. (No existe dicho índice: '%s'.)
2082	Index associated with UNIQUE or PRIMARY KEY constraint cannot be dropped. (El índice asociado a la restricción UNIQUE o PRIMARY KEY no se puede descartar.)
2083	BEGIN is not allowed in SQL. (No se permite BEGIN en SQL.)
2084	COMMIT is not allowed in SQL. (No se permite COMMIT en SQL.)
2085	ROLLBACK is not allowed in SQL. (No se permite ROLLBACK en SQL.)
2086	Unable to open a temporary database file for storing temporary tables. (No se puede abrir un archivo de base de datos temporal para guardar las tablas temporales.)
2087	Unable to identify the object to be reindexed. (No se puede identificar el objeto que se va a reindexar.)
2088	Table '%s' may not be modified. (La tabla '%s' no se puede modificar.)
2089	Cannot modify '%s' because it is a view. (No se puede modificar '%s' porque es una vista.)
2090	Variable number must be between ?0 and ?%d<. (El número de variables debe estar entre ?0 y ?%d<.)
2092	Misuse of aliased aggregate '%s'. (Uso incorrecto de agregación dentada '%s'.)
2093	Ambiguous column name: '[%s.[%s.]]%s'. (Nombre de columna ambiguo: '[%s.[%s.]]%s'.)
2094	No such function: '%s'. (No existe dicha función: '%s'.)
2095	Wrong number of arguments to function '%s'. (Número incorrecto de argumentos para la función '%s'.)
2096	Subqueries prohibited in CHECK constraints. (Subconsultas prohibidas en restricciones de CHECK.)
2097	Parameters prohibited in CHECK constraints. (Parámetros prohibidos en restricciones de CHECK.)
2098	Expression tree is too large (maximum depth %d). (El árbol de expresiones es demasiado grande (profundidad máxima: %d).)

2099	RAISE() may only be used within a trigger-program. (RAISE() solo se puede utilizar en un programa de activador.)
2100	Table '%s' has %d columns but %d values were supplied. (La tabla '%s' tiene %d columnas, pero se han facilitado %d valores.)
2101	Database schema is locked: '%s'. (El esquema de base de datos está bloqueado: '%s'.)
2102	Statement too long. (Declaración demasiado larga.)
2103	Unable to delete/modify collation sequence due to active statements. (No se puede eliminar/modificar secuencia de intercalación porque hay declaraciones activas.)
2104	Too many attached databases - max %d. (Demasiadas bases de datos asociadas; máx.: %d.)
2105	Cannot ATTACH database within transaction. (No se puede realizar una operación ATTACH con la base de datos durante la transacción.)
2106	Database '%s' is already in use. (La base de datos '%s' ya se está utilizando.)
2108	Attached databases must use the same text encoding as main database. (Las bases de datos asociadas deben utilizar la misma codificación de texto que la base de datos principal.)
2200	Out of memory. (Memoria insuficiente.)
2201	Unable to open database. (No se puede abrir la base de datos.)
2202	Cannot DETACH database within transaction. (No se puede realizar una operación DETACH con la base de datos durante la transacción.)
2203	Cannot detach database: '%s'. (No se puede desasociar la base de datos: '%s'.)
2204	Database '%s' is locked. (La base de datos '%s' está bloqueada.)
2205	Unable to acquire a read lock on the database. (No se puede adquirir un bloqueo de lectura en la base de datos.)
2206	[column|columns] '%s'[,'%s'] are not [unique|is] not unique. (La(s) [columna|columnas] '%s'[,'%s'] no son [exclusivas|no] es exclusiva.)
2207	Malformed database schema. (Esquema de base de datos formado incorrectamente.)
2208	Unsupported file format. (Formato de archivo no admitido.)
2209	Unrecognized token: '%s'. (Símbolo no reconocido: '%s'.)
2300	Could not convert text value to numeric value. (No se pudo convertir el valor de texto en valor numérico.)
2301	Could not convert string value to date. (No se pudo convertir el valor de cadena en una fecha.)
2302	Could not convert floating point value to integer without loss of data. (No se pudo convertir el valor de coma flotante en entero sin pérdida de datos.)
2303	Cannot rollback transaction - SQL statements in progress. (No se puede invertir la transacción: declaraciones SQL en curso.)
2304	Cannot commit transaction - SQL statements in progress. (No se puede confirmar la transacción: declaraciones SQL en curso.)
2305	Database table is locked: '%s'. (La base de datos está bloqueada: '%s'.)
2306	Read-only table. (Tabla de solo lectura.)
2307	String or blob too big. (Cadena o blob demasiado grande.)
2309	Cannot open indexed column for writing. (No se puede abrir la columna indexada para su escritura.)
2400	Cannot open value of type %s. (No se puede abrir un valor de tipo %s.)
2401	No such rowid: %s<. (No existe dicho identificador de fila: %s<.)
2402	Object name reserved for internal use: '%s'. (Nombre de objeto reservado para uso interno: '%s'.)
2403	View '%s' may not be altered. (La vista '%s' no se puede alterar.)
2404	Default value of column '%s' is not constant. (El valor predeterminado de la columna '%s' no es constante.)
2405	Not authorized to use function '%s'. (No dispone de autorización para utilizar la función '%s'.)
2406	Misuse of aggregate function '%s'. (Uso incorrecto de función de agregación '%s'.)
2407	Misuse of aggregate: '%s'. (Uso incorrecto de agregación: '%s'.)
2408	No such database: '%s'. (No existe dicha base de datos: '%s'.)
2409	Table '%s' has no column named '%s'. (La tabla '%s' no contiene ninguna columna llamada '%s'.)

2501	No such module: '%s'. (No existe dicho módulo: '%s'.)
2508	No such savepoint: '%s'. (No existe dicho punto guardado: '%s'.)
2510	Cannot rollback - no transaction is active. (No se puede restaurar: no hay ninguna transacción activa.)
2511	Cannot commit - no transaction is active. (No se puede confirmar: no hay ninguna transacción activa.)

Capítulo 68: AGAL (Adobe Graphics Assembly Language, Lenguaje de ensamblaje de gráficos de Adobe)

El Lenguaje de ensamblaje de gráficos de Adobe (AGAL) es un lenguaje sombreador para definir programas de procesamiento de vértices y fragmentos. Los programas AGAL deben estar cargados en el contexto de procesamiento en formato de código de bytes binarios que se describe en este documento.

Formato de código de bytes de AGAL

El código de bytes de AGAL debe utilizar formato Endian.LITTLE_ENDIAN.

Encabezado del código de bytes

El código de bytes de AGAL debe comenzar con un encabezado de 7 bytes:

```
A0 01000000 A1 00 -- for a vertex program
A0 01000000 A1 01 -- for a fragment program
```

Desplazamiento (bytes)	Tamaño (bytes)	Nombre	Descripción
0	1	magic	debe ser 0xa0
1	4	version	debe ser 1
5	1	ID de tipo de sombreador	debe ser 0xa1
6	1	tipo de sombreador	0 para un programa de vértices; 1 para un programa de fragmentos

Símbolos

El encabezado va inmediatamente seguido de cualquier número de símbolos. Cada símbolo tiene un tamaño de 192 bits (24 bytes) y siempre tiene el formato:

```
[opcode][destination][source1][source2 o sampler]
```

No todos los códigos de operación utilizan todos estos campos. Los campos no utilizados deben establecerse en 0.

Códigos de operación

El campo [opcode] tiene un tamaño de 32 bits y puede tomar uno de los valores siguientes:

Nombre	Código de operación	Operación	Descripción
mov	0x00	mover	mueve datos de source1 a destination, en el sentido de los componentes
add	0x01	sumar	destination = source1 + source2, en el sentido de los componentes
sub	0x02	restar	destination = source1 - source2, en el sentido de los componentes
mul	0x03	multiplicar	destination = source1 * source2, en el sentido de los componentes
div	0x04	dividir	destination = source1 / source2, en el sentido de los componentes
rcp	0x05	recíproco	destination = 1/source1, en el sentido de los componentes
min	0x06	mínimo	destination = minimum(source1,source2), en el sentido de los componentes
max	0x07	máximo	destination = maximum(source1,source2), en el sentido de los componentes
frc	0x08	fraccional	destination = source1 - (float)floor(source1), en el sentido de los componentes
sqt	0x09	raíz cuadrada	destination = sqrt(source1), en el sentido de los componentes
rsq	0x0a	raíz recíproca	destination = 1/sqrt(source1), en el sentido de los componentes
pow	0x0b	potencia	destination = pow(source1,source2), en el sentido de los componentes
log	0x0c	logaritmo	destination = log_2(source1), en el sentido de los componentes
exp	0x0d	exponencial	destination = 2^source1, en el sentido de los componentes
nrm	0x0e	normalizar	destination = normalize(source1), en el sentido de los componentes (produce solo un resultado de 3 componentes, destination debe estar enmascarado en .xyz o valor inferior)
sin	0x0f	seno	destination = sin(source1), en el sentido de los componentes
cos	0x10	coseno	destination = cos(source1), en el sentido de los componentes
crs	0x11	producto cruzado	destination.x = source1.y * source2.z - source1.z * source2.y destination.y = source1.z * source2.x - source1.x * source2.z destination.z = source1.x * source2.y - source1.y * source2.x (produce solo un resultado de 3 componentes, destination debe estar enmascarado en .xyz o valor inferior)
dp3	0x12	producto de puntos	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z
dp4	0x13	producto de puntos	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z + source1.w*source2.w
abs	0x14	absoluto	destination = abs(source1), en el sentido de los componentes
neg	0x15	negativo	destination = -source1, en el sentido de los componentes
sat	0x16	saturar	destination = maximum(minimum(source1,1),0), en el sentido de los componentes

Nombre	Código de operación	Operación	Descripción
m33	0x17	multiplicar por matriz 3x3	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) (produce solo un resultado de 3 componentes, destination debe estar enmascarado en .xyz o valor inferior)
m44	0x18	multiplicar por matriz 4x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) destination.w = (source1.x * source2[3].x) + (source1.y * source2[3].y) + (source1.z * source2[3].z) + (source1.w * source2[3].w)
m34	0x19	multiplicar por matriz 3x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) (produce solo un resultado de 3 componentes, destination debe estar enmascarado en .xyz o valor inferior)
kil	0x27	eliminar/descartar (solamente sombreado de fragmentos)	Si un único componente de origen escalar es menor que cero, el fragmento se descarta y no se lleva al búfer de fotogramas. (El registro del destino debe establecerse todo en 0)
tex	0x28	muestra de textura (solamente sombreado de fragmentos)	destination equals load from texture source2 at coordinates source1. En este caso, source2 debe estar en formato de muestreador.
sge	0x29	establecido si es mayor o igual que	destination = source1 >= source2 ? 1 : 0, en el sentido de los componentes
slt	0x2a	establecido si es menor que	destination = source1 < source2 ? 1 : 0, en el sentido de los componentes
seq	0x2c	establecido si es igual que	destination = source1 == source2 ? 1 : 0, en el sentido de los componentes
sne	0x2d	establecido si no es igual que	destination = source1 != source2 ? 1 : 0, en el sentido de los componentes

Formato de campo de destino

El campo [destination] tiene 32 bits de tamaño:

```
31.............................0
----TTTT----MMMMNNNNNNNNNNNNNNNN
```

T = Tipo de registro (4 bits)

M = Máscara de escritura (4 bits)

N = Número de registro (16 bits)

- = sin definir, debe ser 0

Formato de campo de origen

El campo [source] tiene 64 bits de tamaño:

```
63.............................................................0
D-------------QQ----IIII----TTTTSSSSSSSSOOOOOOOONNNNNNNNNNNNNNNN
```

D = Directo=0/Indirecto=1 para directo se omiten Q e I, 1 bit

Q = Selección de componentes de registro de índices (2 bits)

I = Tipo de registro de índices (4 bits)

T = Tipo de registro (4 bits)

S = Reordenación (8 bits, 2 bits por componente)

O = Desplazamiento indirecto (8 bits)

N = Número de registro (16 bits)

- = sin definir, debe ser 0

Formato de campo de muestreador

El segundo campo de origen para el código de operación tex debe estar en formato [sampler], con tamaño de 64 bits:

```
63.............................................................0
FFFFMMMMWWWWSSSSDDDD--------TTTT--------BBBBBBBBNNNNNNNNNNNNNNNN
```

N = Número de registro del muestreador (16 bits)

B = Nivel de detalle de textura (NdD) parcial, entero con signo, escala por 8. El valor de coma flotante utilizado es b/8.0 (8 bits)

T = Tipo de registro, debe ser 5, muestreador (4 bits)

F = Filtro (0=más cercano,1=lineal) (4 bits)

M = Mipmap (0=desactivado,1=más cercano, 2=lineal)

W = Envoltura (0=fijación,1=repetición)

S = Bits especiales de indicación (debe ser 0)

D = Dimensión (0=2D, 1=Cubo)

Registros de programa

Se definen los siguientes tipos de registros: Utilice el valor mostrado para especificar un tipo de registro en los campos de un símbolo:

Nombre	Valor	Número por programa de fragmentos	Número por programa de vértices	Uso
Atributo	0	No disponible	8	Entrada de sombreador de vértices; se lee desde un búfer de vértices especificado con Context3D.setVertexBufferAt().
Constante	1	28	128	Entrada de sombreador; se establece con la familia de funciones Context3D.setProgramConstants().
Temporal	2	8	8	Registro temporal para computación, no accesible desde fuera del programa.
Salida	3	1	1	Salida de sombreador: en un programa de vértices, el resultado es la posición del espacio del clip; en un programa de fragmentos, el resultado es un color.
Varying	4	8	8	Transfiera datos interpolados entre sombreadores de vértices y fragmentos. Los registros variantes desde el programa de vértices se aplican como entrada para el programa de fragmentos. Los valores se interpolan según la distancia desde los vértices del triángulo.
Sampler	5	8	No disponible	Entrada de sombreador de fragmentos; lee desde una textura especificada mediante Context3D.setTextureAt().