

Desarrollo de aplicaciones de ADOBE® AIR™ 1.1 con ADOBE® FLASH® CS4 PROFESSIONAL

© 2008 Adobe Systems Incorporated. Todos los derechos reservados.

Desarrollo de aplicaciones de Adobe® AIR™ 1.1 con Adobe® Flash® CS4 Professional

Si esta guía se distribuye con software que incluye un contrato de licencia de usuario final, la guía, así como el software descrito en ella, se proporciona con una licencia y sólo puede usarse o copiarse en conformidad con los términos de dicha licencia. Con la excepción de lo permitido por la licencia, ninguna parte de esta guía puede ser reproducida, almacenada en un sistema de recuperación de datos ni transmitida de ninguna forma ni por ningún medio, ya sea electrónico, mecánico, de grabación o de otro tipo, sin el consentimiento previo por escrito de Adobe Systems Incorporated. Tenga en cuenta que el contenido de esta guía está protegido por las leyes de derechos de autor aunque no se distribuya con software que incluya un contrato de licencia de usuario final.

El contenido de esta guía se proporciona exclusivamente con fines informativos, está sujeto a cambios sin previo aviso y no debe interpretarse como un compromiso de Adobe Systems Incorporated. Adobe Systems Incorporated no asume ninguna responsabilidad por los errores o imprecisiones que puedan existir en el contenido informativo de esta guía.

Recuerde que las ilustraciones o imágenes existentes que desee incluir en su proyecto pueden estar protegidas por las leyes de derechos de autor. La incorporación no autorizada de este material en sus trabajos puede infringir los derechos del propietario de los derechos de autor. Asegúrese de obtener los permisos necesarios del propietario de los derechos de autor.

Las referencias a nombres de empresas que aparecen en las plantillas de ejemplo sólo tienen fines ilustrativos y no pretenden hacer referencia a ninguna organización real ni a personas concretas.

Adobe, the Adobe logo, Acrobat, ActionScript, Adobe AIR, ColdFusion, Dreamweaver, Flash, Flex, Flex Builder, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Macintosh, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)

MPEG Layer-3 audio compression technology licensed by Fraunhofer IIS and Thomson Multimedia (<http://www.mp3licensing.com>).

Speech compression and decompression technology licensed from Nellymoser, Inc. (www.nellymoser.com)

Video compression and decompression is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

This product includes software developed by the OpenSymphony Group (<http://www.opensymphony.com/>)

This product contains either BSAFE and/or TIPEM software by RSA Security, Inc.

**Sorenson
Spark.**

Sorenson Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

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

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contenido

Capítulo 1: Instalación de Adobe AIR

Requisitos del sistema para Adobe AIR	1
Instalación de Adobe AIR	2
Desinstalación de Adobe AIR	2
Instalación y ejecución de las aplicaciones de AIR de muestra	2

Capítulo 2: Configuración de la Actualización de Adobe AIR para Flash CS3

Requisitos del sistema para la Actualización de Adobe AIR para Flash CS3	4
Desinstalación de la actualización de Adobe AIR para Flash CS3	4
Instalación de la actualización de Adobe AIR para Flash CS3	5
Influencia de Adobe AIR en Flash CS3	6

Capítulo 3: Presentación de Adobe AIR

Capítulo 4: Recursos sobre AIR

Capítulo 5: Creación de su primera aplicación de AIR en Flash CS3 o CS4

Creación de la aplicación Hello World en Flash	10
Prueba de la aplicación	10
Conversión de una aplicación de Flash en una de Adobe AIR	12

Capítulo 6: Actualización de Adobe AIR para Flash CS3 Professional

Creación de un archivo de Adobe AIR	13
Configuración de publicación de Adobe AIR	13
Vista previa de una aplicación de Adobe AIR	15
Depuración de una aplicación de Adobe AIR	15
Creación de archivos de aplicación e instalador de AIR	15
Creación de un archivo descriptor de aplicación personalizado	20
Firma de la aplicación	21

Capítulo 7: Seguridad en AIR

Aspectos básicos de la seguridad de AIR	23
Instalación y actualizaciones	23
Entornos limitados	27
Seguridad en HTML	30
Creación de scripts entre contenido en diferentes dominios	35
Escribir en el disco	39
Cómo trabajar de forma segura con contenido que no es de confianza	40
Prácticas recomendadas de seguridad para desarrolladores	41
Firma de código	43

Capítulo 8: Configuración de las propiedades de una aplicación de AIR

Estructura del archivo descriptor de la aplicación	44
Definición de propiedades en el archivo descriptor de la aplicación	45

Capítulo 9: Nuevas funciones en Adobe AIR

Nuevas clases del motor de ejecución	53
Clases del motor de ejecución con nuevas funciones	55
Clases del marco de supervisión del servicio	56

Capítulo 10: Trabajo con ventanas nativas

Información adicional sobre ventanas nativas	57
Aspectos básicos de ventanas de AIR	57
Creación de ventanas	62
Gestión de ventanas	69
Detección de eventos de ventanas	75
Visualización de ventanas a pantalla completa	76

Capítulo 11: Pantallas

Información suplementaria en línea sobre las pantallas	77
Aspectos básicos de las pantallas	77
Enumeración de las pantallas	78

Capítulo 12: Trabajo con menús nativos

Información adicional en línea de menús nativos	81
Aspectos básicos del menú AIR	81
Creación de menús nativos	85
Menús contextuales	87
Menús contextuales en HTML	88
Definición de menús nativos de forma descriptiva	89
Visualización de menús emergentes	91
Gestión de eventos de menú	91
Ejemplo: Menú de ventana y de aplicación	93

Capítulo 13: Iconos de la barra de tareas

Información adicional en línea sobre los iconos de la barra de tareas	96
Iconos de la barra de tareas	96
Iconos del Dock	97
Iconos de la bandeja del sistema	97
Botones e iconos de la barra de tareas de la ventana	99

Capítulo 14: Trabajo con el sistema de archivos

Información suplementaria en línea sobre la API de archivos de AIR	101
Aspectos básicos de los archivos de AIR	101
Trabajo con objetos File	102
Obtención de información sobre el sistema de archivos	110
Trabajo con directorios	111
Trabajo con archivos	113
Lectura y escritura de archivos	116

Capítulo 15: Arrastrar y colocar

Información suplementaria en línea sobre la función de arrastrar y colocar	127
Aspectos básicos de arrastrar y colocar	128

Compatibilidad con el gesto de arrastrar hacia fuera	129
Compatibilidad con el gesto de arrastrar hacia dentro	132
Arrastrar y colocar en HTML	135
Capítulo 16: Copiar y pegar	
Información suplementaria en línea sobre la función de copiar y pegar	143
Aspectos básicos de copiar y pegar	143
Lectura y escritura en el portapapeles del sistema	144
Copiar y pegar en HTML	145
Comandos de menú y pulsaciones de tecla para copiar y pegar	147
Formatos de datos para Clipboard	149
Capítulo 17: Trabajo con conjuntos de bytes	
Lectura y escritura de un ByteArray	154
Ejemplo de ByteArray: Lectura de un archivo .zip	159
Capítulo 18: Trabajo con bases de datos SQL locales	
Información adicional en línea sobre bases de datos SQL locales	164
Bases de datos SQL locales	165
Creación y modificación de una base de datos	169
Manipulación de los datos de una base de datos SQL	172
Utilización de operaciones sincrónicas y asíncronas de base de datos	190
Estrategias para la utilización de bases de datos SQL	194
Capítulo 19: Almacenamiento de datos cifrados	
Capítulo 20: Entorno HTML	
Información general sobre el entorno HTML	201
Extensiones de AIR y WebKit	204
Capítulo 21: Programación en HTML y JavaScript	
Información sobre la clase HTMLLoader	217
Cómo evitar errores de JavaScript relacionados con la seguridad	218
Acceso a las clases de API de AIR desde JavaScript	222
Información sobre URL en AIR	224
Disponibilidad de objetos ActionScript en JavaScript	225
Acceso a objetos JavaScript y DOM HTML desde ActionScript	226
Incorporación de contenido SWF en HTML	228
Utilización de las bibliotecas de ActionScript en una página HTML	228
Conversión de los objetos Date y RegExp	230
Manipulación de una hoja de estilo HTML de ActionScript	230
Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad	232
Capítulo 22: Gestión de eventos asociados con HTML	
Eventos HTMLLoader	236
Gestión de eventos DOM con ActionScript	236
Respuesta a excepciones en JavaScript sin capturar	237
Gestión de eventos del motor de ejecución con JavaScript	239

Capítulo 23: Utilización de scripts en el contenedor HTML

Propiedades de visualización de objetos HTMLLoader	242
Desplazamiento de contenido HTML	244
Acceso a la lista del historial HTML	245
Configuración del agente de usuario que se utiliza al cargar contenido HTML	246
Configuración de la codificación de caracteres para utilizar con el contenido HTML	246
Definición de interfaces de usuario del navegador para el contenido HTML	247
Creación de subclases de la clase HTMLLoader	254

Capítulo 24: Cómo añadir contenido PDF

Detección de la capacidad de PDF	256
Carga de contenido PDF	257
Uso de scripts con el contenido PDF	257
Limitaciones conocidas del contenido PDF en AIR	259

Capítulo 25: Utilización de la administración de derechos digitales

Información suplementaria en línea sobre la administración de derechos digitales	262
Aspectos básicos del flujo de trabajo con FLV cifrado	262
Cambios en la clase NetStream	263
Utilización de la clase DRMStatusEvent	265
Utilización de la clase DRMAuthenticateEvent	266
Utilización de la clase DRMErrorEvent	268

Capítulo 26: Opciones de inicio y cierre de aplicaciones

Invocación de aplicaciones	271
Captura de argumentos de la línea de comandos	272
Inicio de aplicaciones al iniciar sesión	274
Invocación desde el navegador	274
Cierre de una aplicación	276

Capítulo 27: Lectura de la configuración de una aplicación

Lectura del archivo descriptor de la aplicación	278
Obtención de los identificadores de la aplicación y del editor	278

Capítulo 28: Trabajo con información sobre el motor de ejecución y el sistema operativo

Gestión de asociaciones con archivos	280
Obtención de la versión y el nivel de revisión del motor de ejecución	281
Detección de las capacidades de AIR	281
Seguimiento de la presencia de usuarios	281

Capítulo 29: Supervisión de la conectividad de la red

Detección de cambios de conectividad de la red	283
Aspectos básicos de la supervisión del servicio	283
Detección de la conectividad de HTTP	284
Detección de la conectividad de sockets	284

Capítulo 30: Peticiones de URL y redes

Utilización de la clase URLRequest	286
Cambios en la clase URLStream	289
Apertura de una dirección URL en el navegador Web predeterminado del sistema	289

Capítulo 31: Distribución, instalación y ejecución de aplicaciones de AIR

Instalación y ejecución de una aplicación de AIR desde el escritorio	292
Instalación y ejecución de aplicaciones de AIR desde una página Web	293
Implementación en la empresa	301
Firma digital de archivos de AIR	301

Capítulo 32: Actualización de aplicaciones de AIR

Actualización de aplicaciones	308
Presentación de interfaz de usuario de actualización personalizada de la aplicación	310
Descarga de un archivo de AIR en el equipo del usuario	310
Ver si una aplicación se está ejecutando por primera vez	311

Capítulo 33: Localización de aplicaciones de AIR

Introducción a la localización	312
Localización del nombre y la descripción en el instalador de la aplicación	312
Elección de una configuración regional	313
Localización de contenido de Flash	313
Localización de contenido HTML	314
Localización de fechas, horas y monedas	322

Capítulo 34: Creación de aplicaciones de AIR con las herramientas de la línea de comandos

Utilización de AIR Debug Launcher (ADL)	324
Empaquetado de archivos de instalación de AIR con AIR Developer Tool (ADT)	326
Firma de un archivo AIR para cambiar el certificado de la aplicación	332
Creación de certificados autofirmados con ADT	333
Utilización de Apache Ant con las herramientas del SDK	334

Índice	338
---------------------	-----

Capítulo 1: Instalación de Adobe AIR

Adobe® AIR™ permite ejecutar aplicaciones de AIR en el escritorio. El motor de ejecución se puede instalar de cualquiera de las formas siguientes:

- Mediante la instalación independiente del motor de ejecución (sin instalar además una aplicación de AIR)
- Mediante la instalación de una aplicación de AIR por primera vez (aparecerá un mensaje sugiriendo que se instale el motor de ejecución)
- Mediante la instalación de un entorno de desarrollo de AIR como el kit de desarrollo de software de AIR, Adobe® Flex™ Builder™ 3 o el kit de desarrollo de software de Adobe Flex™ 3 (que incluye las herramientas de desarrollo de la línea de comandos de AIR)

El motor de ejecución sólo necesita instalarse una vez en cada ordenador.

Requisitos del sistema para Adobe AIR

Los requisitos del sistema para ejecutar Adobe AIR son:

- Para aplicaciones básicas de Adobe AIR:

	Windows	Macintosh
Procesador	Procesador Intel® Pentium® de 1,0 GHz o más	Procesador PowerPC® G3 de 1,0 GHz o más o Procesador Intel Core™ Duo de 1,83 GHz o más
Memoria	256 MB de RAM	256 MB de RAM
Sistema operativo	Windows 2000 Service Pack 4; Windows XP SP2; Vista	Mac OS X 10.4.10 ó 10.5.x (PowerPC); Mac OS X 10.4.x o 10.5.x (Intel)

- Para aplicaciones de Adobe AIR que utilizan vídeo de pantalla completa con ajuste de escala en el hardware:

	Windows	Macintosh
Procesador	Procesador Intel® Pentium® de 2,0 GHz o más	Procesador PowerPC® G4 de 1,8 GHz o más o Procesador Intel Core™ Duo de 1,33 GHz o más
Memoria	512 MB de RAM; 32 MB de RAM de vídeo	256 MB de RAM; 32 MB de RAM de vídeo
Sistema operativo	Windows 2000 Service Pack 4; Windows XP SP2; Vista	Mac OS X v.10.4.10 o v.10.5 (Intel o PowerPC) NOTA: el codec que se utiliza para visualizar vídeo H.264 requiere un procesador Intel

Instalación de Adobe AIR

Siga estas instrucciones para descargar e instalar las versiones de AIR para Windows® y Mac OS X.

Para actualizar el motor de ejecución, el usuario debe contar con privilegios de administrador del ordenador.

Instalación del motor de ejecución en un ordenador con Windows

- 1 Descargue el [archivo de instalación del motor de ejecución](#).
- 2 Haga doble clic en el archivo de instalación del motor de ejecución.
- 3 Siga las indicaciones que aparecen en la ventana de instalación para llevar a cabo la instalación.

Instalación del motor de ejecución en un ordenador con Mac

- 1 Descargue el [archivo de instalación del motor de ejecución](#).
- 2 Haga doble clic en el archivo de instalación del motor de ejecución.
- 3 Siga las indicaciones que aparecen en la ventana de instalación para llevar a cabo la instalación.
- 4 Si el instalador presenta una ventana de autenticación, escriba el nombre de usuario y la contraseña que utiliza para Mac OS.

Desinstalación de Adobe AIR

Una vez instalado el motor de ejecución, se puede desinstalar siguiendo los procedimientos que se explican a continuación.

Desinstalación del motor de ejecución en un ordenador con Windows

- 1 En el menú Inicio de Windows, seleccione Configuración > Panel de control.
- 2 Seleccione la opción Agregar o quitar programas.
- 3 Seleccione “Adobe AIR” para desinstalar el motor de ejecución.
- 4 Haga clic en el botón Cambiar o quitar.

Desinstalación del motor de ejecución en un ordenador con Mac

- Haga doble clic en el archivo de desinstalación de Adobe AIR, que se encuentra en la carpeta /Aplicaciones.

Instalación y ejecución de las aplicaciones de AIR de muestra

Hay algunas aplicaciones de muestra a disposición para demostrar las funciones de AIR. Para tener acceso a las mismas e instalarlas, siga estas instrucciones:

- 1 Descargue y ejecute las [aplicaciones de AIR de muestra](#). Están a disposición tanto las aplicaciones compiladas como el código fuente.

- 2 Para descargar y ejecutar una aplicación de muestra, haga clic en el botón Instalar ahora de la aplicación. Un mensaje indica instalar y ejecutar la aplicación.
- 3 Si opta por descargar aplicaciones de muestra y ejecutarlas más adelante, seleccione los vínculos de descarga. Las aplicaciones de AIR pueden ejecutarse en cualquier momento de la siguiente manera:
 - En Windows, haga doble clic en el icono de la aplicación que se encuentra en el escritorio o seleccione la aplicación en el menú Inicio.
 - En Mac OS, haga doble clic en el icono de la aplicación, que se instala por omisión en la carpeta Aplicaciones de su directorio de usuario (por ejemplo, en Macintosh HD/Usuarios/UsuarioFicticio/Aplicaciones/).

Nota: revise las notas de versión de AIR por si hubiera alguna actualización de estas instrucciones. Puede encontrarlas en: http://www.adobe.com/go/learn_air_relnotes_es.

Capítulo 2: Configuración de la Actualización de Adobe AIR para Flash CS3

La actualización de Adobe® AIR™ para Adobe® Flash® CS3 Professional aumenta el entorno de desarrollo de Flash con elementos que permiten al usuario crear aplicaciones de AIR con Flash. Permite crear, probar y depurar archivos de aplicaciones de AIR en Flash.

Adobe® Flash® CS4 Professional puede crear aplicaciones de AIR, ya que lo admite de forma nativa. Para obtener más información, consulte [Publicación para Adobe AIR](#) en Utilización de Flash.

Requisitos del sistema para la Actualización de Adobe AIR para Flash CS3

Para poder utilizar Flash CS3 en el desarrollo y la ejecución de aplicaciones de AIR, debe tener el siguiente software instalado:

- Flash CS3 Professional

Si no dispone de una copia de Flash CS3 Professional, puede adquirirla en el sitio Web de Adobe:

<http://www.adobe.com/es/products/flash/>

- Adobe AIR

Para obtener información sobre la instalación de Adobe AIR, consulte “[Instalación de Adobe AIR](#)” en la página 1.

- Actualización de Adobe AIR para Flash CS3

Si tiene instalada una versión anterior de la actualización de Adobe AIR para Flash CS3, siga los pasos que se describen en [Desinstalación de la actualización de Adobe AIR para Flash CS3](#) para desinstalarla. Si *no* tiene instalada ninguna actualización de Adobe AIR para Flash CS3, continúe con los pasos de la sección “[Instalación de la actualización de Adobe AIR para Flash CS3](#)” en la página 5.

Desinstalación de la actualización de Adobe AIR para Flash CS3

Si tiene instalada una versión anterior de la actualización de Adobe AIR para Flash CS3, siga estos pasos para desinstalarla antes de instalar la nueva actualización de Adobe AIR para Flash CS3.

- 1 Elimine la siguiente carpeta:

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\AIK

(Mac) HD:/Aplicaciones/Adobe Flash CS3/AIK

- 2 Vaya a la siguiente ubicación:

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\

(Mac) HD:/Aplicaciones/Adobe Flash CS3/First Run/Commands

y elimine las siguientes carpetas/archivos:

- Carpeta AIR
- AIR - Application and Installer Settings.jsfl
- AIR - Create AIR File.jsfl

3 Elimine el siguiente archivo:

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\

(Mac) HD:/Aplicaciones/Adobe Flash CS3/Configuration/External Libraries/FLAir.bundle.

4 Elimine el siguiente archivo:

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\

(Mac) HD:/Aplicaciones/Adobe Flash CS3/Configuration/Players/ AdobeAIR1_0.xml

5 Vaya a la siguiente ubicación:

(Windows) Disco duro:\Document and Settings\

(Mac) HD:/Usuarios/<nombre de usuario>/Librería/Application Support/Adobe/Flash CS3/<idioma>/Configuration/Commands/

y elimine las siguientes carpetas/archivos:

- Carpeta AIR
- AIR - Application and Installer Settings.jsfl
- AIR - Create AIR File.jsfl

Nota: si no ve la ubicación especificada en Windows, active la opción "Mostrar archivos/carpetas ocultos" en Opciones de carpeta.

Instalación de la actualización de Adobe AIR para Flash CS3

Antes de instalar la actualización de Adobe AIR para Flash CS3, salga de Flash y de todos los navegadores que tenga abiertos.

- Descargue la [actualización de Adobe AIR para Flash CS3](#).
- Una vez descargada la actualización, haga doble clic en el archivo para instalarlo.

Influencia de Adobe AIR en Flash CS3

Tras instalar la actualización de Adobe AIR, comprobará los cambios siguientes en Flash:

- En la ficha Flash del cuadro de diálogo Configuración de publicación (Archivo -> Configuración de publicación), hay una nueva entrada en el menú Versión de Adobe AIR 1.0.
- Una pantalla de bienvenida actualizada con una entrada para crear un archivo de Flash (Adobe AIR).

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\FirstRun\StartPage

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\FirstRun\StartPage\resources

Nota: si instala la aplicación en un ordenador Macintosh, si no aparece la opción Archivo de Flash (Adobe AIR) en la pantalla de bienvenida, elimine la siguiente carpeta y reinicie Flash:

HD:/Usuarios/<nombre de usuario>/Librería/Application Support/Adobe/Flash
CS3/<idioma>/Configuration/StartPage

- Un nuevo archivo playerglobal.swc que incluye todas las API de ActionScript 3.0 y de Adobe AIR en la carpeta ActionScript 3.0/Classes.

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\Configuration\ActionScript 3.0
Classes

(Mac) HD:/Aplicaciones/Adobe Flash CS3/Configuration/ActionScript 3.0/Classes/

- Nuevos archivos jsfl (AIR - Application and Installer Settings.jsfl, AIR - Publish AIR File.jsfl)

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\FirstRun\Commands

(Mac) HD:/Aplicaciones/Adobe Flash CS3/First Run/Commands/

- Kit de desarrollo de software de Adobe AIR (AIK)

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\AIK

- Biblioteca externa

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\Configuration\External Libraries

(Mac) HD:/Aplicaciones/Adobe Flash CS3/Configuration/External Libraries/

- Archivo de configuración de destino

(Windows) Disco duro:\Archivos de programa\Adobe\Adobe Flash CS3\en\Configuration\Players\

(Mac) HD:/Aplicaciones/Adobe Flash CS3/Configuration/Players

Capítulo 3: Presentación de Adobe AIR

Adobe® AIR™ es un motor de ejecución que funciona con distintos sistemas operativos para aprovechar sus aptitudes de desarrollo de Web existentes (Adobe® Flash® CS3 Professional, Adobe® Flex™, HTML, JavaScript®, Ajax) para elaborar e implementar aplicaciones de Internet enriquecidas (RIA) en el escritorio.

AIR permite trabajar en entornos que le son familiares para aprovechar las herramientas y los métodos que le resulten más cómodos. Su compatibilidad con Flash, Flex, HTML, JavaScript y Ajax le brinda la mejor experiencia posible para satisfacer sus necesidades.

Por ejemplo, se pueden desarrollar aplicaciones utilizando una de las tecnologías siguientes o combinando varias de ellas:

- Flash/Flex/ActionScript
- HTML/JavaScript/CSS/Ajax
- PDF (que puede aprovecharse con cualquier aplicación)

En consecuencia, las aplicaciones de AIR pueden ser:

- basadas en Flash o Flex: aplicación cuyo contenido raíz es Flash/Flex (SWF);
- basadas en Flash o Flex con HTML o PDF. Aplicaciones cuyo contenido raíz es Flash/Flex (SWF) y que incluyen contenido HTML (HTML, JS, CSS) o PDF;
- basadas en HTML. Aplicación cuyo contenido raíz es HTML, JS, CSS;
- basadas en HTML con Flash/Flex o PDF. Aplicaciones cuyo contenido raíz es HTML y que incluyen contenido de Flash/Flex (SWF) o PDF.

Los usuarios interactúan con las aplicaciones de AIR de la misma forma en que interactúan con las aplicaciones de escritorio nativas. El motor de ejecución se instala una vez en el ordenador del usuario y después se instalan y ejecutan las aplicaciones de AIR como cualquier otra aplicación de escritorio.

El motor de ejecución ofrece una arquitectura y plataforma compatibles con distintos sistemas operativos para la implementación de aplicaciones. La compatibilidad y constancia del funcionamiento y las interacciones en distintos escritorios obvia la necesidad de realizar pruebas en distintos navegadores. En lugar de desarrollar programas para un sistema operativo determinado, el desarrollador centra sus esfuerzos en el motor de ejecución, lo cual ofrece las siguientes ventajas:

- Las aplicaciones desarrolladas para AIR se ejecutan en varios sistemas operativos distintos sin suponer trabajo adicional para el desarrollador. El motor de ejecución asegura una presentación e interacciones constantes y predecibles en todos los sistemas operativos compatibles con AIR.
- La creación de aplicaciones se agiliza gracias a que se pueden aprovechar las tecnologías Web y los diseños existentes, así como extender las aplicaciones Web al escritorio sin necesidad de aprender las tradicionales tecnologías de desarrollo del escritorio o la complejidad del código nativo.
- El desarrollo de aplicaciones resulta más fácil que cuando se utilizan lenguajes de nivel inferior como C y C++. No hace falta gestionar las complejas API de nivel inferior que son específicas para cada sistema operativo.

Al desarrollar aplicaciones para AIR se puede aprovechar un juego enriquecido de arquitecturas e interfaces API:

- API específicas para AIR proporcionadas por el motor de ejecución y la arquitectura de AIR
- API de ActionScript utilizadas en archivos SWF y la arquitectura de Flex (además de otras bibliotecas y arquitecturas basadas en ActionScript)

- HTML, CSS y JavaScript
- La mayoría de las arquitecturas de Ajax

AIR es toda una novedad en la forma de crear, implementar y experimentar las aplicaciones. Permite tener un mayor control creativo y extender al escritorio las aplicaciones basadas en Flash, Flex, HTML y Ajax sin necesidad de aprender las tradicionales tecnologías de desarrollo del escritorio.

Capítulo 4: Recursos sobre AIR

Para obtener más información sobre el desarrollo de aplicaciones de Adobe® AIR™, consulte los recursos siguientes:

Fuente	Ubicación
<i>Programación con ActionScript 3.0</i>	http://www.adobe.com/go/learn_fl_cs4_programmingAS3_es
<i>Referencia del lenguaje y componentes ActionScript 3.0 (incluye AIR)</i>	http://www.adobe.com/go/learn_flashcs4_langref_es
<i>Guías de inicio rápido de Adobe AIR para Flash</i>	http://www.adobe.com/go/learn_air_flash_qs_es
<i>Utilización de Flash</i>	http://www.adobe.com/go/learn_fl_cs4_using_es
<i>Utilización de componentes ActionScript 3.0</i>	http://www.adobe.com/go/learn_fl_cs4_as3components_es

Encontrará artículos, muestras y presentaciones por expertos tanto de Adobe como de la comunidad en el Centro de desarrollo de Adobe AIR en <http://www.adobe.com/es/devnet/air/>. También se puede descargar de ahí Adobe AIR y software asociado.

Hay una sección específicamente dirigida a los desarrolladores de Flash en <http://www.adobe.com/es/devnet/air/flash/>.

Para obtener información sobre la resolución de problemas para su producto y sobre las opciones de asistencia técnica gratuitas y pagadas, visite el sitio Web del servicio de asistencia técnica de Adobe en <http://www.adobe.com/es/support/>. Siga el vínculo bajo Formación para tener acceso a los libros de Adobe Press, una variedad de recursos de formación, programas de certificación en software de Adobe, y mucho más.

Capítulo 5: Creación de su primera aplicación de AIR en Flash CS3 o CS4

A continuación se resume la demostración del funcionamiento de Adobe® AIR™. Siga las instrucciones de este tema para crear y empaquetar una sencilla aplicación “Hello World” de AIR.

Si aún no lo ha hecho, descargue e instale la actualización de Adobe AIR para Flash CS3. Para obtener más información sobre la instalación de Adobe AIR para Flash CS3, consulte [“Configuración de la Actualización de Adobe AIR para Flash CS3”](#) en la página 4.

Si utiliza Flash CS4, el soporte para Adobe AIR está integrado, por lo que no es necesario instalar ningún componente adicional para poder empezar.

Creación de la aplicación Hello World en Flash

Crear una aplicación de Adobe AIR en Flash es muy similar a crear cualquier otra aplicación de Flash. La diferencia es que se comienza creando un archivo de Flash (Adobe AIR) desde la pantalla de bienvenida y se finaliza creando la configuración de aplicación e instalador e instalando la aplicación de AIR. El siguiente procedimiento le guiará en el proceso de creación de una sencilla aplicación Hello World.

Para crear la aplicación Hello World

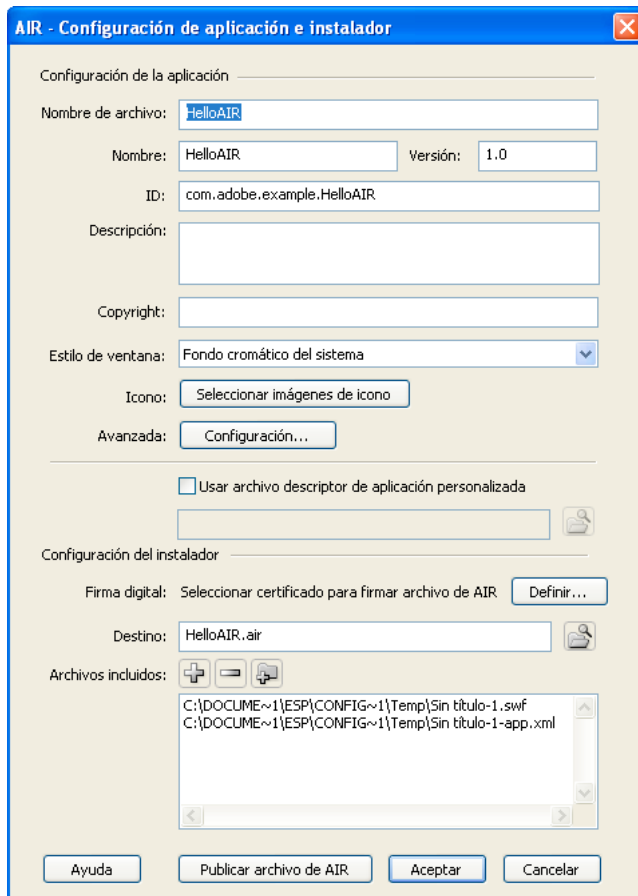
- 1 Inicie Flash.
- 2 En la pantalla de bienvenida, haga clic en Archivo de Flash (Adobe AIR) para crear un archivo FLA vacío con configuración de publicación de Adobe AIR.
- 3 Haga clic en Aceptar para responder al diálogo de resumen de edición de Adobe AIR con Flash CS3. Este diálogo tarda unos segundos en aparecer la primera vez. (Este diálogo no aparece en Flash CS4.)
- 4 Seleccione la herramienta Texto en el panel Herramientas y cree un campo de texto estático (valor predeterminado) en el centro del escenario. Dele una anchura suficiente para que pueda contener entre 15 y 20 caracteres.
- 5 Escriba el texto “Hello World” en el campo de texto.
- 6 Guarde el archivo y asígnele un nombre (por ejemplo, helloAIR).

Prueba de la aplicación

- 1 Pulse Ctrl + Intro o seleccione Control -> Probar película para probar la aplicación en Adobe AIR.
- 2 Para utilizar la función Depurar película, añada primero código ActionScript a la aplicación. Puede intentarlo rápidamente añadiendo una sentencia trace como ésta:

```
trace("Running AIR application using Debug Movie");
```
- 3 Pulse Ctrl + Mayús + Intro o seleccione Control -> Depurar película para ejecutar la aplicación con Depurar película.

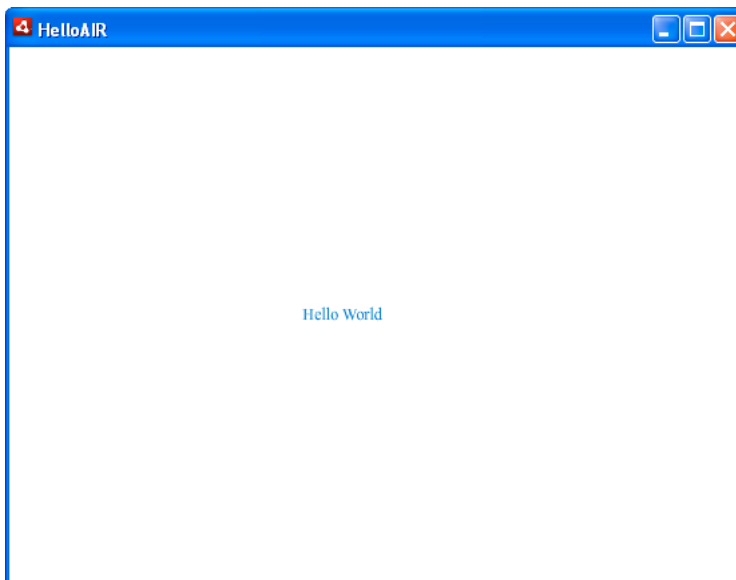
- 4 En Flash CS3, seleccione Comandos > AIR - Configuración de aplicación e instalador para abrir el diálogo correspondiente. En Flash CS4, puede abrir este diálogo seleccionando Archivo > Configuración de AIR.



- 5 Firme el paquete de Adobe AIR con el certificado digital con firma automática:
- Haga clic en el botón Definir en el mensaje de la firma digital para abrir el cuadro de diálogo Firma digital.
 - Haga clic en el botón Crear para abrir el cuadro de diálogo Crear certificado digital con firma automática.
 - Rellene los campos Nombre del editor, Unidad de organización, Nombre de organización, Correo electrónico, País, Contraseña y Confirmar contraseña.
 - Especifique el tipo de certificado. La opción Tipo de certificado hace referencia al nivel de seguridad: 1024-RSA utiliza una clave de 1.024 bits (menos segura) y 2048-RSA utiliza una clave de 2048 bits (más segura).
 - Guarde la información en un archivo de certificado en la opción Guardar como o haciendo clic en el botón Examinar... para acceder a la ubicación de la carpeta. (Por ejemplo, *C:/Temp/mycert.pfx*). Cuando haya terminado, haga clic en Aceptar.
 - Flash regresa al cuadro de diálogo Firma digital. La ruta y el nombre de archivo del certificado con firma automática creado aparece ahora en el cuadro de texto Certificado. Si no es así, introduzca la ruta y el nombre de archivo o haga clic en el botón Examinar para encontrarlo y seleccionarlo.
 - Escriba la misma contraseña en el campo de texto Contraseña del cuadro de diálogo Firma digital que la contraseña que asignó en el paso C y haga clic en Aceptar. Para obtener más información sobre la firma de aplicaciones de Adobe AIR, consulte ["Firma de la aplicación"](#) en la página 21.

- 6 Para crear el archivo aplicación y el instalador, haga clic en el botón Publicar archivo de AIR. Debe haber ejecutado previamente los comandos Probar película o Depurar película para crear los archivos SWF y application.xml antes de crear el archivo de AIR.
- 7 Para instalar la aplicación, haga doble clic en el archivo de AIR (*application.air*) en la misma carpeta en la que guardó la aplicación.
- 8 Haga clic en el botón Instalar del cuadro de diálogo Instalación de la aplicación.
- 9 Revise los parámetros de Preferencias de instalación y Ubicación y asegúrese de que la casilla de verificación 'Iniciar aplicación tras la instalación' está seleccionada. Seguidamente, haga clic en Continuar.
- 10 Haga clic en Finalizar cuando aparezca el mensaje Instalación completada.

La aplicación Hello World se asemeja a la de la ilustración:



Conversión de una aplicación de Flash en una de Adobe AIR

También es posible convertir una aplicación de Flash existente en una aplicación de AIR. Para obtener más información sobre el modo de hacerlo en Flash CS3, consulte "[Configuración de publicación de Adobe AIR](#)" en la página 13. Si utiliza Flash CS4, consulte [Publicación para Adobe AIR](#) en Utilización de Flash.

Capítulo 6: Actualización de Adobe AIR para Flash CS3 Professional

La actualización de Adobe® AIR™ para Adobe® Flash® CS3 Professional aumenta el entorno de edición para que pueda crear, depurar y empaquetar aplicaciones de Adobe AIR con Flash. El proceso de creación de una aplicación de Adobe AIR consiste en crear un archivo FLA de Adobe AIR, establecer la configuración de publicación adecuada, desarrollar la aplicación y crear los archivos de aplicación y de instalación que permitan implementarla.

Si utiliza Adobe® Flash® CS4 Professional, consulte [Publicación para Adobe AIR](#) en Utilización de Flash para obtener más información sobre la creación de aplicaciones de AIR.

Para obtener más información sobre las API de ActionScript™ de Adobe AIR que se pueden utilizar en su aplicación, consulte [Referencia del lenguaje y componentes ActionScript 3.0](#).

Si quiere obtener una lista con las API de ActionScript de Adobe AIR, consulte “[Nuevas funciones en Adobe AIR](#)” en la página 53.

Nota: para utilizar clases del paquete `air.net`, arrastre primero el componente `ServiceMonitorShim` del panel Componentes al panel Biblioteca y, después, añada la siguiente sentencia `import` al código ActionScript 3.0:

```
import air.net.*;
```

Creación de un archivo de Adobe AIR

Puede crear documentos de archivo de Flash (Adobe AIR) desde la pantalla de bienvenida de Flash, o bien crear un archivo de Flash (ActionScript™ 3.0) y convertirlo a archivo de Adobe AIR desde el cuadro de diálogo Configuración de publicación. No es posible crear un archivo de Adobe AIR desde el cuadro de diálogo Nuevo documento (Archivo > Nuevo). Para obtener más información sobre la conversión de un archivo de Flash en Adobe AIR, consulte “[Configuración de publicación de Adobe AIR](#)” en la página 13.

- 1 Inicie Flash o, si ya lo ha iniciado, cierre todos los documentos abiertos para volver a la pantalla de bienvenida.

Nota: si ha desactivado la aparición de la pantalla de bienvenida de Flash, puede volver a visualizarla si selecciona Edición > Preferencias y, después, Pantalla de bienvenida en el menú Al iniciar, en la categoría General.

- 2 En la pantalla de bienvenida, haga clic en Archivo de Flash (Adobe AIR).

Aparecerá un cuadro de diálogo de alerta para indicar cómo acceder a la configuración de la aplicación de Adobe AIR y cómo acceder a la documentación de la Ayuda. Si no desea que aparezca este cuadro de alerta en futuras ocasiones, seleccione No volver a mostrar. Debe tener en cuenta que no será posible volver a activar la aparición de este cuadro.

Configuración de publicación de Adobe AIR

Utilice la configuración de publicación de Flash para examinar o modificar la configuración de un archivo de AIR o para convertir un archivo de Flash (ActionScript 3.0) en un archivo de Flash (Adobe AIR).

Visualización de la configuración de publicación de Adobe AIR

- 1 Desde la pantalla de bienvenida de Flash, abra un documento Archivo de Flash (Adobe AIR).
- 2 Seleccione Archivo > Configuración de publicación y haga clic en la ficha Flash para ver la configuración de publicación de Adobe AIR.

Adobe AIR 1.0 está seleccionado automáticamente en el menú Versión cuando se abre un documento de Adobe AIR. La versión de ActionScript™ se establece automáticamente en ActionScript 3.0. La configuración local de seguridad de reproducción se obvia, ya que es irrelevante para un archivo SWF de AIR.

Si ha abierto un archivo FLA de Flash, puede convertirlo en archivo de AIR de Flash modificando su configuración de publicación.

Conversión de un archivo FLA de Flash en un archivo de AIR de Flash mediante el cuadro de diálogo Configuración de publicación

- 1 Realice uno de los siguientes pasos:
 - Abra un archivo de Flash existente.
 - Utilice la pantalla de bienvenida o seleccione Archivo > Nuevo para crear un nuevo archivo de Flash.
- 2 Seleccione Archivo > Configuración de publicación.
- 3 En la ficha Flash, seleccione Adobe AIR 1.0 en el menú Versión.

La opción de versión de ActionScript está desactivada porque ActionScript 3.0 es la única opción válida para un archivo de AIR.

El resto de opciones predeterminadas son las mismas para un archivo de Flash o uno de Adobe AIR.

- 4 Haga clic en el botón Publicar y luego en Aceptar para cerrar el cuadro de diálogo Configuración de publicación. Si está seleccionada la herramienta Selección, el inspector de propiedades ahora indica que el destino del reproductor es Adobe AIR 1.

***Nota:** si elige el perfil Adobe AIR 1.0, Flash añade automáticamente la ubicación del archivo AIR `playerglobal.swc` a la variable de entorno `Classpath`. El archivo AIR `playerglobal.swc` permite utilizar las API de AIR de ActionScript. Sin embargo, si cambia de Adobe AIR 1 a Adobe® Flash® Player 9, Flash no vuelve automáticamente al perfil predeterminado ni modifica la configuración de `Classpath` para que utilice el archivo `playerglobal.swc` para Flash Player 9. Si cambia la configuración de publicación de Adobe AIR 1 a Flash Player 9, debe cambiar también el perfil de publicación a Predeterminado.*

Para obtener información adicional sobre el cuadro de diálogo Configuración de publicación, consulte el manual Utilización de Flash en www.adobe.com/go/learn_fl_using_es.

Conversión de un archivo FLA de Flash en una aplicación AIR de Flash mediante el menú Comandos

- 1 Abra el archivo FLA de Flash.
- 2 Si va a abrir un archivo de Flash (ActionScript 3.0) nuevo, guárdelo. De no hacerlo, recibirá una advertencia cuando intente acceder al siguiente paso.
- 3 Seleccione Comandos > AIR - Configuración de aplicación e instalador.

Aparecerá un cuadro de alerta para preguntar si quiere convertir el archivo a la configuración de publicación de Adobe AIR.
- 4 Haga clic en Aceptar para convertir el archivo FLA en una configuración de publicación de Adobe AIR. Aparecerá el cuadro de diálogo AIR - Configuración de aplicación e instalador.

Para obtener información sobre el cuadro de diálogo AIR - Configuración de aplicación e instalador, consulte “Creación de archivos de aplicación e instalador de AIR” en la página 15.

Puede utilizar los comandos Probar película, Depurar película y Crear archivo de AIR en el archivo FLA de AIR convertido.

Vista previa de una aplicación de Adobe AIR

Puede obtener la vista previa de un archivo SWF de Adobe AIR como lo haría en la ventana de la aplicación de AIR. Consultar una vista previa resulta útil si se quieren ver los aspectos visibles de la aplicación sin tener que empaquetarla ni instalarla.

- 1 Es importante comprobar que se ha definido la configuración de publicación para una aplicación de Adobe AIR. Para obtener más información, consulte “[Configuración de publicación de Adobe AIR](#)” en la página 13.
- 2 Seleccione Control > Probar película o pulse Control+Intro.

Si no ha definido la configuración de la aplicación desde el cuadro de diálogo AIR - Configuración de aplicación e instalador, Flash genera un archivo descriptor predeterminado de la aplicación (*swfname-app.xml*) en la misma carpeta en la que se escribe el archivo SWF. Si ha definido la configuración de la aplicación desde el cuadro de diálogo AIR - Configuración de aplicación e instalador, el archivo descriptor de la aplicación reflejará dicha configuración.

Depuración de una aplicación de Adobe AIR

El archivo SWF de Adobe AIR se puede depurar como del mismo modo que un archivo SWF de ActionScript 3.0 de Flash Player 9 (con la excepción de la depuración remota).

- 1 Asegúrese de haber definido la configuración de publicación de Adobe AIR.
- 2 Añada código ActionScript al panel Acciones (Ventana > Acciones). Para realizar pruebas, basta con añadir una sentencia `trace()` como la siguiente en el panel Acciones (en el primer fotograma de la línea de tiempo):

```
trace("My application is running");
```

- 3 Seleccione Depurar > Depurar película o pulse la combinación de teclas Control+Mayús+Intro.

Flash inicia el depurador de ActionScript y exporta el archivo SWF con la información de depuración.

Si no ha definido la configuración de la aplicación desde el cuadro de diálogo AIR - configuración de aplicación e instalador, Flash genera un archivo descriptor predeterminado de la aplicación (*swfname-app.xml*) en la misma carpeta en la que se escribe el archivo SWF. Si ha definido la configuración de la aplicación desde el cuadro de diálogo AIR - Configuración de aplicación e instalador, el archivo descriptor de la aplicación reflejará dicha configuración.

Cuando selecciona Depurar > Depurar película p pulsa la combinación de teclas Control+Mayús+Intro para depurar la aplicación, Flash muestra una alerta si ésta no contiene ningún código ActionScript.

Creación de archivos de aplicación e instalador de AIR

Una vez finalizada la aplicación, puede crear los archivos de aplicación e instalador de AIR para implementarla. Adobe AIR añade dos nuevas opciones de menú al menú Comandos de Flash: AIR - Configuración de aplicación e instalador y AIR - Crear archivo de AIR. Una vez creada la configuración de aplicación e instalador de AIR, puede utilizar la opción AIR - Crear archivo de AIR para crear de nuevo el archivo de AIR (.air), pero con la configuración existente.

Creación de archivos de aplicación e instalador de Adobe

- 1 En Flash, abra la página o el grupo de páginas que forman la aplicación de Adobe AIR.
- 2 Guarde el archivo FLA de Adobe AIR antes de abrir el cuadro de diálogo AIR - Configuración de aplicación e instalador.
- 3 Seleccione Comandos > AIR - Configuración de aplicación e instalador.
- 4 Rellene el cuadro de diálogo AIR - Configuración de aplicación e instalador y haga clic en Publicar archivo de AIR.

Al hacer clic en el botón Publicar archivo de AIR, se empaquetan los siguientes archivos: el archivo FLA, el archivo SWF, el archivo descriptor de la aplicación, los archivos de iconos de la aplicación y los archivos presentes en el cuadro de texto Archivos incluidos. Si aún no ha creado un certificado digital, Flash muestra el cuadro de diálogo Firma digital al hacer clic en el botón Publicar archivo de AIR.

El cuadro de diálogo AIR - Configuración de aplicación e instalador está dividido en dos secciones: Configuración de la aplicación y Configuración del instalador. Para obtener más información sobre estas configuraciones, consulte las siguientes secciones.

Configuración de la aplicación

La sección Configuración de la aplicación del cuadro de diálogo AIR - Configuración de aplicación e instalador tiene las siguientes opciones:

Nombre de archivo Nombre del archivo principal de la aplicación. Utiliza el nombre del archivo SWF como nombre predeterminado.

Nombre Nombre utilizado por el instalador para generar el nombre de archivo y la carpeta de la aplicación. El nombre sólo puede contener caracteres válidos admitidos en nombres de archivo y de carpeta. Utiliza el nombre del archivo SWF como nombre predeterminado.

Versión Opcional. Especifica el número de versión de la aplicación. De forma predeterminada, está vacío.

ID Identifica la aplicación con un ID exclusivo. Si lo desea, puede cambiar el ID predeterminado. No utilice espacios ni caracteres especiales en el ID. Sólo se admiten como caracteres válidos: 0-9, a-z, A-Z, . (punto) y - (guión), de 1 a 212 caracteres de longitud. Su valor predeterminado es `com.adobe.example.nombre_aplicación`.

Descripción Opcional. Permite introducir una descripción de la aplicación para mostrarla mientras el usuario la instala. De forma predeterminada, está vacío.

Copyright Opcional. Permite introducir un aviso de copyright para mostrarlo mientras el usuario instala la aplicación.

Estilo de ventana Especifica el estilo de ventana (o del fondo cromático) que se utiliza en la interfaz de usuario cuando éste ejecuta la aplicación en su ordenador. Puede especificar la opción Fondo cromático del sistema. Hace referencia al estilo visual utilizado por el sistema operativo. También puede elegir Fondo cromático personalizado (opaco) o Fondo cromático personalizado (transparente). Para no mostrar el fondo cromático del sistema de la aplicación, seleccione Ninguno. El fondo cromático del sistema está presente en la aplicación junto con los controles de ventanas estándar del sistema operativo. El fondo cromático personalizado (opaco) elimina los controles estándar del sistema y permite crear controles propios para la aplicación. (El fondo cromático personalizado se crea directamente en el archivo FLA.) El fondo cromático personalizado (transparente) es como el personalizado (opaco), pero añade funcionalidad de transparencia en los bordes de la página. Esta funcionalidad permite que las ventanas de la aplicación no tengan forma cuadrada o rectangular.

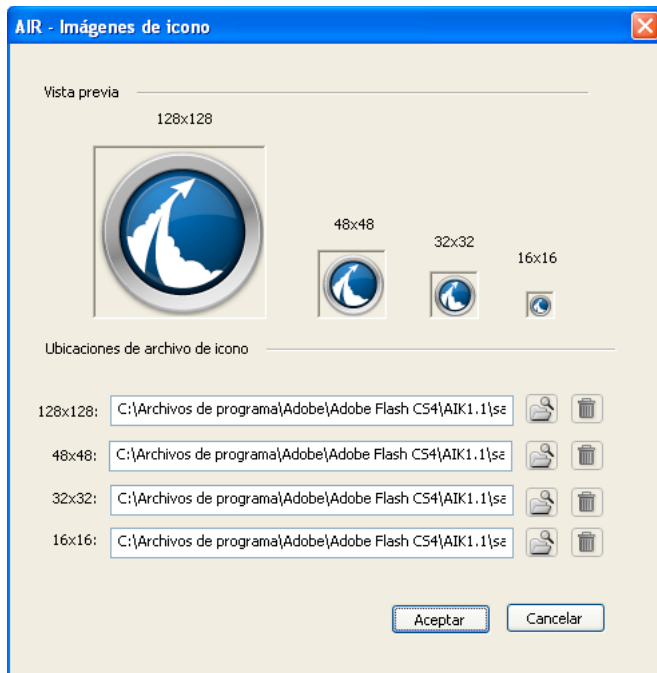
Icono Opcional. Permite especificar el icono de la aplicación. El icono aparece tras instalar la aplicación y ejecutarla en Adobe AIR. Puede elegir entre cuatro tamaños de icono distintos (128, 48, 32 y 16 píxeles) para que se visualice correctamente en las cuatro vistas en las que aparece. Por ejemplo, el icono puede aparecer en el explorador de archivos

como miniatura, detalle o título. También puede aparecer como icono de escritorio y en el título de una ventana de aplicación de AIR o en otros lugares.

La imagen predeterminada del icono es el icono de muestra de aplicación de AIR si no se especifica ninguna otra.

Para especificar un icono, haga clic en el botón Seleccionar imágenes de icono en el cuadro de diálogo AIR - Configuración de aplicación e instalador. En el cuadro de diálogo Imágenes de icono, haga clic en la carpeta de cada tamaño de icono y seleccione el archivo que quiera utilizar. Los archivos deben estar en formato PNG (Portable Network Graphics).

En la siguiente ilustración se muestra el cuadro de diálogo Imágenes de icono con los iconos de la aplicación predeterminados de Adobe AIR.



Especificación de distintos tamaños de imagen de icono de la aplicación

Si especifica una imagen, debe tener el tamaño exacto (128x128, 48x48, 32x32 o 16x16). Si no proporciona una imagen con tamaño de icono concreto, Adobe AIR la ajusta a una de las imágenes proporcionadas y crea la imagen.

Configuración avanzada

El botón Configuración del cuadro de diálogo AIR - Configuración de aplicación e instalador permite especificar la configuración avanzada del archivo descriptor de la aplicación. Al hacer clic en el botón Configuración, aparece el cuadro de diálogo Configuración avanzada.

En el cuadro de diálogo Configuración avanzada se puede especificar cualquier tipo de archivo asociado que deba manejar la aplicación. Por ejemplo, si quiere que la aplicación sea la aplicación principal para gestionar archivos HTML, debe especificarlo en el cuadro de texto Tipos de archivo asociados.

También puede especificar la configuración de los siguientes aspectos de la aplicación:

- El tamaño y la posición de la ventana inicial
- La carpeta de instalación de la aplicación
- La carpeta del menú Programa en la que se coloca la aplicación.

El cuadro de diálogo tiene las opciones siguientes:

Tipos de archivo asociados Permite especificar los tipos de archivo asociados que maneja la aplicación de AIR. Haga clic en el botón Más (+) para añadir un nuevo tipo de archivo en el cuadro de texto. Al hacer clic en el botón Más, aparece el cuadro de diálogo Configuración de tipo de archivo. Si hace clic en el botón Menos (-), eliminará el elemento seleccionado en el cuadro de texto. Si hace clic en el botón Lápiz, se abre el cuadro de diálogo Configuración de tipo de archivo para que pueda editar el elemento seleccionado en el cuadro de texto. De forma predeterminada, los botones Menos (-) y Lápiz aparecen desactivados. Al seleccionar un elemento en el cuadro de texto, se activan los botones Menos (-) y Lápiz para poder eliminar o editar el elemento. El valor predeterminado del cuadro de texto en Ninguno.

Para obtener más información sobre la configuración de tipos de archivo asociados, consulte [“Configuración de tipo de archivo”](#) en la página 19.

Configuración de la ventana inicial Permite especifica la configuración de tamaño y de posición de la ventana inicial de la aplicación.

- Anchura: especifica la anchura inicial de la ventana, en píxeles. De forma predeterminada, el campo está vacío.
- Altura: especifica la altura inicial de la ventana, en píxeles. De forma predeterminada, el campo está vacío.
- X: especifica la posición horizontal inicial de la ventana, en píxeles. De forma predeterminada, el campo está vacío.
- Y: especifica la posición vertical inicial de la ventana, en píxeles. De forma predeterminada, el campo está vacío.
- Anchura máxima y Altura máxima: especifican el tamaño máximo de la ventana, en píxeles. De forma predeterminada, estos campos están vacíos.
- Anchura mínima y Altura mínima: especifican el tamaño mínimo de la ventana, en píxeles. De forma predeterminada, estos campos están vacíos.
- Se puede maximizar: permite especificar si el usuario puede maximizar o no la ventana. De forma predeterminada, esta opción está seleccionada (o es true).
- Se puede minimizar: permite especificar si el usuario puede minimizar o no la ventana. De forma predeterminada, esta opción está seleccionada (o es true).
- Redimensionable: permite especificar si el usuario puede cambiar el tamaño de la ventana. Si esta opción no se selecciona, los campos Anchura máxima, Altura máxima, Anchura mínima y Altura mínima aparecen desactivados. De forma predeterminada, esta opción está seleccionada (o es true).
- Visible: permite especificar si la ventana de la aplicación está visible inicialmente. De forma predeterminada, la opción está seleccionada (o es true).

Otras opciones Permite especificar la siguiente información adicional relacionada con la instalación:

- Carpeta de instalación: especifica la carpeta en la que se instala la aplicación.
- Carpeta del menú Programa: especifica el nombre de la carpeta del menú Programa de la aplicación.
- Usar UI personalizada para actualizaciones: especifica qué ocurre cuando un usuario abre un archivo de AIR para una aplicación ya instalada. De forma predeterminada, AIR abre un cuadro de diálogo donde el usuario puede actualizar la versión instalada con la versión del archivo de AIR. Si no quiere que el usuario pueda tomar esta decisión y desea tener control total sobre la aplicación y sus actualizaciones, seleccione esta opción. Al seleccionar esta opción, se anula el comportamiento predeterminado y se permite que la aplicación controle sus propias actualizaciones.

Para obtener más información sobre la actualización de una aplicación de AIR mediante programación, consulte [“Actualización de aplicaciones de AIR”](#) en la página 308.

Configuración de tipo de archivo

Flash muestra el cuadro de diálogo Configuración de tipo de archivo al hacer clic en el botón Más (+) o en el botón Lápiz en el cuadro de diálogo Configuración avanzada, o bien al añadir o editar tipos de archivo asociados para la aplicación.

Los dos únicos campos necesarios de este cuadro de diálogo son Nombre y Extensión. Si hace clic en Aceptar sin haber rellenado estos campos, Flash mostrará un cuadro de diálogo de error.

Puede especificar las opciones siguientes para un tipo de archivo asociado:

Nombre El nombre del tipo de archivo (por ejemplo, Lenguaje de marcado de hipertexto, Archivo de texto o Ejemplo).

Extensión La extensión del nombre de archivo (por ejemplo, html, txt o xml); puede contener hasta 39 caracteres alfanuméricos básicos, (A-Za-z0-9) sin punto inicial.

Descripción Opcional. Una descripción del tipo de archivo (por ejemplo, Archivo de vídeo de Adobe).

Tipo de contenido Opcional. Especifica el tipo MIME para el archivo.

Configuración de icono de tipo de archivo Opcional. Permite especificar un icono asociado al tipo de archivo. Puede elegir entre cuatro tamaños de icono distintos (128x128, 48x48, 32x32 y 16x16 píxeles) para que se visualice correctamente en las cuatro vistas en las que aparece. Por ejemplo, el icono puede aparecer en el explorador de archivos como miniatura, detalle o título.

Si quiere especificar una imagen, ésta debe tener el tamaño indicado. Si no especifica ningún archivo para un tamaño concreto, AIR utiliza la imagen con tamaño más parecido y la escala para que se ajuste al caso concreto.

Para especificar un icono, puede hacer clic en la carpeta del tamaño de icono y seleccionar un archivo de icono, o bien facilitar la ruta de acceso y el nombre de archivo del archivo de icono en el cuadro de texto que aparece junto al mensaje. El archivo de icono debe estar en formato PNG.

Una vez creado el nuevo tipo de archivo, aparece en el cuadro de lista Tipo de archivo del cuadro de diálogo Configuración avanzada.

Configuración del archivo descriptor de aplicación

La configuración definida para la aplicación se guarda en el archivo *nombre_aplicación-app.xml*. Sin embargo, si lo desea, puede indicar a Flash que utilice un archivo descriptor de aplicación personalizado.

Usar archivo descriptor de aplicación personalizado Permite buscar un archivo descriptor de aplicación personalizado. Si selecciona la opción Usar archivo descriptor de aplicación personalizado, la sección Configuración de aplicación del cuadro de diálogo se desactiva. Para especificar la ubicación del archivo descriptor de aplicación personalizado, puede introducirla en el campo de texto situado debajo de Usar archivo descriptor de aplicación personalizado, o bien hacer clic en el icono de la carpeta y acceder a la ubicación. Para obtener más información sobre el archivo descriptor de aplicación, consulte [“Creación de un archivo descriptor de aplicación personalizado”](#) en la página 20.

Configuración del instalador

La segunda sección del cuadro de diálogo AIR - Configuración de aplicación e instalador contiene parámetros relacionados con la instalación de la aplicación.

Firma digital Todas las aplicaciones de Adobe AIR deben estar firmadas para poder instalarse en otro sistema. Para obtener más información sobre la asignación de una firma digital a una aplicación Flash de Adobe AIR, consulte [“Firma de la aplicación”](#) en la página 21.

Destino Especifica la ubicación en la que se guarda el archivo de AIR. La ubicación predeterminada es el directorio en el que se guardó el archivo FLA. Haga clic en el icono de la carpeta para seleccionar una ubicación diferente. El nombre predeterminado del paquete es el nombre de la aplicación seguido de la extensión de archivo .air.

Archivos/Carpetas incluidos Especifica qué archivos y carpetas adicionales se incluyen en la aplicación. Haga clic en el botón Más (+) para añadir archivos y en el botón de la carpeta para añadir carpetas. Para eliminar un archivo o una carpeta de la lista, seleccione el elemento correspondiente y haga clic en el botón Menos (-).

De forma predeterminada, el archivo descriptor de aplicación y el archivo SWF principal se añaden automáticamente a la lista del paquete. La lista del paquete muestra estos archivos aunque no se haya publicado aún el archivo FLA de Adobe AIR. La lista del paquete muestra los archivos y las carpetas con una estructura simple. No se muestran los archivos de cada carpeta; los nombres completos de las rutas sí se muestran, aunque pueden aparecer cortados.

La lista no incluye los archivos de icono. Cuando Flash empaqueta los archivos, copia los archivos de icono en una carpeta temporal relacionada con la ubicación del archivo SWF. Flash elimina la carpeta una vez finalizado el paquete.

¿No consigue crear los archivos de la aplicación y del instalador?

Los archivos de la aplicación y del instalador no se pueden crear en los casos siguientes:

- La cadena del identificador de la aplicación no tiene una longitud correcta o contiene caracteres no válidos. La cadena del identificador de la aplicación puede tener entre 1 y 212 caracteres dentro del rango 0-9, a-z, A-Z, . (punto), - (guión).
- No existen los archivos de la lista del instalador.
- Los tamaños de los archivos de icono personalizados son incorrectos.
- La carpeta de destino de AIR no tiene acceso de escritura.
- No ha firmado la aplicación o no ha especificado que se trata de una aplicación de Adobe AIR que se firmará más adelante.

Creación de un archivo descriptor de aplicación personalizado

El archivo descriptor de aplicación es un archivo XML que se puede editar con un simple editor de texto. Para crear un archivo descriptor de aplicación predeterminado, edite los valores para especificar los que desee. A continuación se muestran los valores predeterminados:

- id = com.adobe.example.nombreswf
- fileName = nombreswf
- name = nombreswf
- version = 1.0
- description = vacío
- copyright = vacío
- initialWindow
 - title = name
 - content = nombreswf.swf
 - systemChrome = standard, type = normal

- transparent = false
- visible = true
- icon
 - image128x128 = icons/AIRApp_128.png
 - image48x48 = icons/AIRApp_48.png
 - image32x32 = icons/AIRApp_32.png
 - image16x16 = icons/AIRApp_16.png
- customUpdateUI = false
- allowBrowserInvocation = false

Para obtener más información sobre el archivo descriptor de aplicación, consulte “[Configuración de las propiedades de una aplicación de AIR](#)” en la página 44.

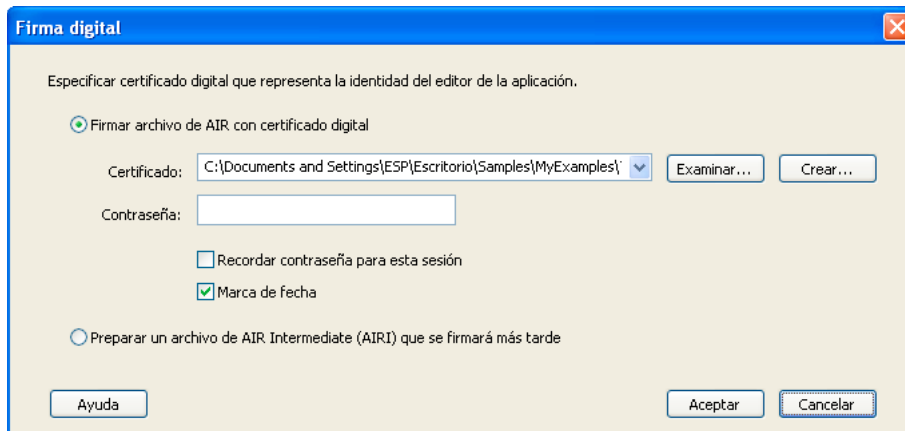
Firma de la aplicación

Todas las aplicaciones de Adobe AIR deben estar firmadas para poder instalarse en otro sistema. Sin embargo, Flash permite crear archivos instaladores de Adobe AIR sin firmar para que la aplicación se pueda firmar más adelante. Estos archivos de instalador sin firmar de Adobe AIR reciben el nombre de paquete AIRI. Esta funcionalidad permite manejar situaciones en las que el certificado se encuentra en otro ordenador o en las que la firma se gestiona independientemente del desarrollo de la aplicación.

Firma de una aplicación de Adobe AIR con un certificado de firma precomprado en una autoridad emisora de certificados raíz

- 1 Haga clic en el botón Definir de Firma digital en el cuadro de diálogo AIR - Configuración de aplicación e instalador. Se abre el cuadro de diálogo Firma digital.

Este cuadro de diálogo tiene dos botones de opción que permiten firmar la aplicación de Adobe AIR con un certificado digital o preparar un paquete AIRI. Si decide firmar la aplicación de AIR, puede utilizar para ello un certificado digital emitido por una entidad emisora de certificados raíz o crear un certificado con firma automática. Un certificado con firma automática es fácil de crear, pero no es tan fiable como uno emitido por una entidad de certificados raíz.



Cuadro de diálogo Firma digital. Permite firmar una aplicación de AIR

- 2 Seleccione un archivo de certificado en el menú o haga clic en el botón Examinar para buscar uno.
- 3 Seleccione el certificado.
- 4 Escriba la contraseña.
- 5 Haga clic en Aceptar.

Para obtener más información sobre el proceso de firma de la aplicación de AIR, consulte [“Firma digital de archivos de AIR”](#) en la página 301.

Creación de un certificado digital con firma automática

- 1 Haga clic en el botón Crear. Se abrirá el cuadro de diálogo Crear certificado digital con firma automática.
- 2 Rellene los campos Nombre del editor, Unidad de organización, Nombre de organización, País, Contraseña y Confirmar contraseña.
- 3 Especifique el tipo de certificado.
La opción Tipo hace referencia al nivel de seguridad del certificado: 1024-RSA utiliza una clave de 1.024 bits (menos segura), mientras que 2048-RSA utiliza una clave de 2.048 bits (más segura).
- 4 Guarde la información en un archivo de certificado en la opción Guardar como o haciendo clic en el botón Examinar para acceder a la ubicación de la carpeta.
- 5 Haga clic en Aceptar.
- 6 En el cuadro de diálogo Firma digital, escriba la contraseña asignada en el segundo paso de este procedimiento y haga clic en Aceptar.

Una vez definido el certificado digital, el botón Definir pasa a ser Cambiar.

Para que Flash recuerde la contraseña utilizada en esta sesión, haga clic en Recordar contraseña para esta sesión.

Si la opción Marca de fecha no está seleccionada al hacer clic en Aceptar, aparece un cuadro de diálogo para advertir que la aplicación no se instalará correctamente si el certificado digital vence. Si hace clic en Sí en la advertencia, la marca de fecha se desactivará. Si hace clic en No, la opción Marca de fecha se selecciona automáticamente y se activa.

Para obtener más información sobre la creación de un certificado digital con firma automática, consulte [“Firma digital de archivos de AIR”](#) en la página 301.

También es posible crear una aplicación intermedia de AIR (AIRI) sin firma digital. El usuario no podrá instalar la aplicación en su escritorio hasta que no haya añadido una firma digital.

Preparar un paquete AIRI para firmarlo más tarde

- ❖ En el cuadro de diálogo Firma digital, seleccione Preparar un archivo de AIR Intermediate (AIRI) que se firmará más tarde y haga clic en Aceptar.
El estado de la firma digital cambia para indicar que ha escogido preparar un paquete AIRI para firmarlo más tarde y el botón Definir pasa a Cambiar.

Capítulo 7: Seguridad en AIR

En este capítulo se describen los temas de seguridad que se deben tener en cuenta cuando se desarrolla una aplicación de AIR.

Aspectos básicos de la seguridad de AIR

Las aplicaciones de AIR se ejecutan con los mismos privilegios de usuario que las aplicaciones nativas. En general, estos privilegios permiten un amplio acceso a las prestaciones del sistema operativo como leer y escribir archivos, iniciar aplicaciones, dibujar en la 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. Éstas son algunas de las vulnerabilidades más comunes que afectan a las aplicaciones de escritorio escritas en código nativo.

Instalación y actualizaciones

Las aplicaciones de AIR se distribuyen a través de los archivos de instalación de AIR que usan la extensión `air`. Cuando se instala Adobe AIR y se abre un archivo de instalación de AIR, el motor de ejecución administra el proceso de instalación.

***Nota:** los desarrolladores pueden especificar una versión y un nombre de aplicación y un origen de editor, 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

Las aplicaciones 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 el equipo del usuario:

- Mac OS: `/Library/Frameworks/`
- Windows: `C:\Archivos de programa\Archivos comunes\Adobe AI`

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, un usuario debe contar con privilegios administrativos.

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. Para más información, consulte [“Distribución, instalación y ejecución de aplicaciones de AIR”](#) en la página 292.

Instalación integrada (motor de ejecución y aplicación)

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

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.

Para más información sobre este proceso, consulte [“Distribución, instalación y ejecución de aplicaciones de AIR”](#) en la página 292

Flujo de instalación de la aplicación

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

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/

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

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.
- 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 está ejecutada por:

- 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)
- 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

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 es `<appData>/<appId>/Local Store/` donde `<appData>` es la "carpeta de preferencias" del usuario, normalmente `/Usuarios/<usuario>/Librería/Preferences`
- 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\<Nombre de usuario>\Application Data`

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 101.

Actualización de Adobe AIR

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

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 sólo para proporcionar nueva funcionalidad sino también para responder a las vulnerabilidades potenciales de seguridad.

Nota: los desarrolladores pueden especificar la versión de una aplicación configurando la propiedad `version` del archivo descriptor de la aplicación. AIR no interpreta la cadena de versión de ningún modo. Por consiguiente, la versión "3.0" no significa que sea más actual que la versión "2.0." Depende del desarrollador mantener el significado de las versiones. Para más información, consulte "Definición de propiedades en el archivo descriptor de la aplicación" en la página 45.

Desinstalación de una aplicación de AIR

Un usuario puede desinstalar una aplicación de AIR:

- En Windows: usando el panel Agregar o quitar programas para quitar la aplicación.
- En Mac OS: eliminando el archivo `app` de la ubicación de instalación.

Al quitar una aplicación de AIR se quitan todos los archivos en el directorio de la aplicación. Sin embargo, no se quitan 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.

Desinstalación de Adobe AIR

Se puede desinstalar AIR de la siguiente manera:

- En Windows: ejecutando Agregar o quitar programas del Panel de control, seleccionando Adobe AIR y "Quitar".
- En Mac OS: ejecutando la aplicación Uninstaller de Adobe AIR en el directorio de aplicaciones.

Parámetros del registro de Windows para administradores

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
<code>AppInstallDisabled</code>	Especifica que se permite la instalación y desinstalación de la aplicación de AIR. Configurada en 0 para "autorizado", y en 1 para "no autorizado".
<code>UntrustedAppInstallDisabled</code>	Especifica que se permite la instalación de aplicaciones de AIR que no es de confianza (aplicaciones que no incluyen un certificado de confianza; para más información consulte "Firma digital de archivos de AIR" en la página 301). Configurada en 0 para "autorizado", y en 1 para "no autorizado".
<code>UpdateDisabled</code>	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".

Entornos limitados

AIR proporciona una arquitectura completa de seguridad que define autorizaciones según cada archivo en una aplicación de AIR, tanto interno como externo. Las autorizaciones se conceden a los archivos según el origen y se asignan en agrupaciones lógicas de seguridad llamados entornos limitados.

Entornos limitados de aplicación de AIR

El modelo de seguridad del motor de ejecución de los entornos limitados está compuesto por el modelo de seguridad de Flash Player con la adición del entorno limitado de aplicación. Los archivos que no se encuentran en el entorno limitado de la aplicación tienen restricciones de seguridad similares a aquellos especificados por el modelo de seguridad de Flash Player.

El motor de ejecución usa estos entornos limitados de seguridad para definir el rango de datos al que puede acceder el código y las operaciones que puede realizar. Para mantener la seguridad local, los archivos en cada entorno limitado se aíslan de los archivos de los otros entornos limitados. Por ejemplo, un archivo SWF cargado en una aplicación de AIR desde una URL de Internet externa se coloca en un entorno limitado remoto y, de forma predeterminada, no tiene permiso para escribir un script en los archivos que residen en el directorio de la aplicación, que se asignan al entorno limitado de la aplicación.

La siguiente tabla describe cada tipo de entorno limitado:

Entorno limitado	Descripción
aplicación	El archivo reside en el directorio de la aplicación y funciona con el conjunto completo de los privilegios de AIR.
remoto	El archivo procede de una URL de Internet y se rige por las reglas de un entorno limitado basado en dominios equivalentes a las reglas que se aplican a los archivos remotos en Flash Player. (Hay entornos limitados remotos individuales para cada dominio de red, como http://www.example.com y https://foo.example.org).
local de confianza	El archivo es un archivo local y el usuario lo ha designado como de confianza, usando ya sea un archivo de configuración de confianza de Flash Player o Settings Manager. El archivo puede leer en orígenes de datos locales y puede comunicarse con Internet, pero no tiene todos los privilegios de AIR.
local con acceso a la red	El archivo es un archivo SWF local publicado con una designación de acceso a la red, pero el usuario no ha confiado explícitamente en el mismo. El archivo se puede comunicar con Internet pero no puede leer en orígenes de datos locales. Este entorno limitado sólo está disponible para el contenido SWF.
local con sistema de archivos	El archivo es un archivo local de script que no se publicó con una designación de acceso a la red y el usuario no ha confiado explícitamente en el mismo. Esto incluye a los archivos JavaScript que no son de confianza. El archivo puede leer en orígenes de datos locales pero no puede comunicarse con Internet.

Este tema se centra principalmente en el entorno limitado de la aplicación y las relaciones con otros entornos limitados en la aplicación de AIR. Los desarrolladores que usan contenido asignado a otros entornos limitados deben consultar más documentación sobre el modelo de seguridad de Flash Player. Consulte el capítulo “Seguridad de Flash Player” en *Programación con ActionScript 3.0* (http://www.adobe.com/go/flashCS3_progAS3_security_es) y el [documento técnico sobre la seguridad de Flash Player 9](http://www.adobe.com/go/fp9_0_security_es) (http://www.adobe.com/go/fp9_0_security_es).

Entorno limitado de la aplicación

Cuando se instala una aplicación, todos los archivos incluidos en un archivo de instalación de AIR se instalan en el equipo del usuario en un directorio de la aplicación. Los desarrolladores pueden hacer referencia a este directorio en código a través del esquema de URL `app:/` (consulte [“Utilización de esquemas de URL de AIR en direcciones URL”](#) en la página 288). Todos los archivos dentro 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 a solo 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 ActionScript que sólo están disponibles para el contenido en el entorno limitado de seguridad de la aplicación se indican con el logo de AIR en la *Referencia del lenguaje ActionScript 3.0 para Adobe AIR*. 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.

Restricciones de JavaScript y HTML

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 32.

Restricciones en etiquetas `img` en el contenido del campo de texto ActionScript

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.

Restricciones en `asfunction`

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.

Acceso prohibido a la caché persistente entre dominios

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).

Privilegios de contenido en entornos limitados que no pertenecen a la aplicación

Los archivos cargados desde una red o Internet se asignan al entorno limitado `remoto`. Los archivos cargados fuera del directorio de la aplicación se asignan al entorno limitado `local con sistema de archivos, local con acceso a la red o local de confianza`; esto depende de cómo se creó el archivo y si el usuario tiene confianza explícita en el archivo a través del Administrador de configuración global de Flash Player. Para más información, consulte http://www.macromedia.com/support/documentation/es/flashplayer/help/settings_manager.html.

Restricciones de JavaScript y HTML

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, hay restricciones en JavaScript en un entorno limitado de seguridad que no pertenece a la aplicación. Las restricciones son:

- 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 más información, consulte “[Creación de scripts entre contenido en diferentes dominios](#)” en la página 35.
- Hay restricciones para llamar al método `window.open()` JavaScript. Para más información, consulte “[Restricciones para llamar al método window.open\(\) JavaScript](#)” en la página 35.

Para más información, consulte “[Restricciones de código del contenido en entornos limitados diferentes](#)” en la página 32

Restricciones para cargar elementos CSS, frame, iframe e img

El contenido HTML en los entornos limitados de seguridad remotos (`red`) sólo 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).

Seguridad en HTML

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 (consulte “[Entornos limitados](#)” en la página 27). Cuando se carga contenido en un iframe o en un fotograma, el motor de ejecución proporciona un mecanismo seguro de *punto 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.

Este tema describe la arquitectura de seguridad HTML de AIR y cómo utilizar iframes, fotogramas y el puente de entorno limitado para configurar la aplicación.

Para más información, consulte “[Cómo evitar errores de JavaScript relacionados con la seguridad](#)” en la página 218.

Información general sobre la configuración de la aplicación basada en HTML

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. Éstas incluyen cookies, objetos compartidos locales, almacenamiento en archivo local, el almacén de archivo cifrado y el almacenamiento local en base de datos.)

HTML en AIR conserva su característica confusa entre código ejecutable y datos. Por esto, AIR coloca el contenido en el fotograma superior del entorno HTML en el entorno limitado de la aplicación y restringe cualquier operación, como `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 trabajar de forma segura con contenido HTML remoto en AIR, se deben utilizar fotogramas o iframes. Aun si no se carga contenido remoto, puede ser más conveniente ejecutar el contenido en un fotograma secundario de entorno limitado para que el contenido se pueda ejecutar sin restricciones en `eval()`. (El uso de entornos limitados puede ser necesario cuando se utiliza algunos fotogramas de trabajo de aplicaciones JavaScript.) 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 32.

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

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

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

Como se describe en la introducción a este tema, “[Seguridad en HTML](#)” en la página 30, 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 218.

Restricciones en el uso de la función `eval()` JavaScript y técnicas similares

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`, sólo 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 riesgo 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 listan maneras 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 `setTimeout()` 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 más información, consulte: “Privilegios de contenido en entornos limitados que no pertenecen a la aplicación” en la página 29 y “Utilización de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación” en la página 40. Adobe mantiene una lista de marcos de Ajax que admiten el entorno limitado de seguridad de aplicación, en <http://www.adobe.com/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)

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 [“Utilización de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación”](#) en la página 40.

Restricciones para usar llamadas XMLHttpRequest

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 `allowcrossdomainxhr`. 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 35.

Restricciones para cargar elementos CSS, frame, iframe e img (para el contenido en entornos limitados que no pertenecen a la aplicación)

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

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 exitosamente 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) sólo 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](#)” en la página 27) 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

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 218.

Creación de scripts entre contenido en diferentes dominios

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

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()`. La llamada de este método del entorno limitado de seguridad de la aplicación no tiene efecto.
- 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 *puede 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)

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 precio 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)

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 233.

Limitación de la exposición de API

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.

Véase también

“[Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad](#)” en la página 232

“[Entorno limitado de la aplicación](#)” en la página 28

“[Privilegios de contenido en entornos limitados que no pertenecen a la aplicación](#)” en la página 29

Escribir en el disco

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 sólo 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 41.

Los desarrolladores de AIR pueden 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 hayan 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`

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 41.

Cómo trabajar de forma segura con contenido que no es de confianza

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()

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 de confianza` llama al método `System.allowDomain()`, se concede el acceso al uso de scripts a cualquier SWF cargado del dominio especificado, volviendo a asignar este archivo remoto del entorno limitado `remoto` al entorno limitado `local de confianza`. 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()`.

Utilización de scripts entre contenido de la aplicación y contenido que no pertenece a la aplicación

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 sólo 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 35.

Protección contra la generación dinámica de contenido SWF no seguro

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: tal vez en una futura versión de Adobe AIR, esta API cambie. Cuando ocurra, tal vez tenga que recompilar el contenido que usa la propiedad `allowLoadBytesCodeExecution` de la clase `LoaderContext`.

Prácticas recomendadas de seguridad para desarrolladores

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

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) sólo si es necesario.
- Incluir un archivo de script en un archivo de AIR (en la aplicación instalada) sólo 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 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

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 intrínsecamente 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
- Usen 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

El almacenamiento de credenciales de usuario en el sistema de archivos local del usuario intrínsecamente 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, 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 más información, consulte “[Almacenamiento de datos cifrados](#)” en la página 199.
- No se deben transmitir credenciales de usuario no cifradas a un origen de red a menos que el origen sea de confianza.
- 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

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. Para más información, consulte [“Definición de propiedades en el archivo descriptor de la aplicación”](#) en la página 45.

Firma de código

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 ya sea vinculando un certificado de una autoridad de certificación externa (CA) o bien construyendo su propio certificado. 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 autofirmados se pueden crear usando `adt` del SDK o usando Flash, Flex Builder u otra aplicación que usa `adt` para la generación de certificados. Los certificados autofirmados no proporcionan ninguna garantía que la aplicación que se está instalando es genuina.

Para más información acerca de la firma digital de AIR, consulte [“Firma digital de archivos de AIR”](#) en la página 301 y [“Creación de aplicaciones de AIR con las herramientas de la línea de comandos”](#) en la página 324.

Capítulo 8: Configuración de las propiedades de una aplicación de AIR

Aparte de todos los archivos y demás objetos que conforman una aplicación de AIR, cada aplicación de AIR requiere un archivo descriptor de la aplicación. El archivo descriptor de la aplicación es un archivo XML que define las propiedades básicas de la aplicación.

Cuando se desarrollan aplicaciones de AIR con Adobe® AIR™ Update para Adobe® Flash® CS3 Professional, el archivo descriptor de la aplicación se genera automáticamente al crear un proyecto de AIR. Para abrir una ventana que permite modificar la configuración del descriptor de la aplicación, vaya al menú Comandos > AIR - Configuración de aplicación e instalador. El archivo descriptor de la aplicación también puede modificarse manualmente.

Estructura del archivo descriptor de la aplicación

El archivo descriptor de la aplicación contiene propiedades que afectan al conjunto de la aplicación, como son el nombre, la versión, el aviso de copyright, etc. El archivo descriptor puede tener cualquier nombre de archivo. Al crear un archivo de AIR con la configuración predeterminada en Flash CS3 o CS4, el nombre del archivo descriptor de la aplicación cambia a `application.xml` y el archivo se coloca en un directorio especial en el paquete de AIR.

El siguiente es un ejemplo de archivo descriptor de una aplicación:

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/1.1">
  <id>com.example.HelloWorld</id>
  <version>2.0</version>
  <filename>Hello World</filename>
  <name>Example Co. AIR Hello World</name>
  <description>
    <text xml:lang="en">This is a example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
  </description>
  <copyright>Copyright (c) 2006 Example Co.</copyright>
  <initialWindow>
    <title>Hello World</title>
    <content>
      HelloWorld-debug.swf
    </content>
    <systemChrome>none</systemChrome>
    <transparent>true</transparent>
    <visible>true</visible>
    <minimizable>true</minimizable>
    <maximizable>false</maximizable>
    <resizable>false</resizable>
    <width>640</width>
    <height>480</height>
    <minSize>320 240</minSize>
    <maxSize>1280 960</maxSize>
  </initialWindow>
  <installFolder>Example Co/Hello World</installFolder>
```

```

<programMenuFolder>Example Co</programMenuFolder>
<icon>
  <image16x16>icons/smallIcon.png</image16x16>
  <image32x32>icons/mediumIcon.png</image32x32>
  <image48x48>icons/bigIcon.png</image48x48>
  <image128x128>icons/biggestIcon.png</image128x128>
</icon>
<customUpdateUI>true</customUpdateUI>
<allowBrowserInvocation>>false</allowBrowserInvocation>
<fileTypes>
  <fileType>
    <name>adobe.VideoFile</name>
    <extension>avf</extension>
    <description>Adobe Video File</description>
    <contentType>application/vnd.adobe.video-file</contentType>
    <icon>
      <image16x16>icons/avfIcon_16.png</image16x16>
      <image32x32>icons/avfIcon_32.png</image32x32>
      <image48x48>icons/avfIcon_48.png</image48x48>
      <image128x128>icons/avfIcon_128.png</image128x128>
    </icon>
  </fileType>
</fileTypes>
</application>

```

Definición de propiedades en el archivo descriptor de la aplicación

La raíz del archivo descriptor de la aplicación contiene una propiedad `application` que tiene varios atributos:

```
<application version="1.0" xmlns="http://ns.adobe.com/air/application/1.1">
```

xmlns El espacio de nombre de AIR, el cual debe definirse como espacio de nombre XML predeterminado. El espacio de nombre cambia con cada edición principal de AIR (pero no con las revisiones menores). El último segmento del espacio de nombre, por ejemplo “1.0”, indica la versión del motor de ejecución que requiere la aplicación.

minimumPatchLevel Opcional. Utilice el atributo `minimumPatchLevel` para especificar el nivel de revisión mínimo de Adobe AIR que requiere la aplicación. Las aplicaciones de AIR suelen especificar qué versión de AIR necesitan mediante la definición del espacio de nombre en el archivo descriptor de la aplicación. El espacio de nombre cambia para cada edición principal de AIR (por ejemplo, 1.0 ó 1.1). El nombre de espacio no cambia para las revisiones. Las revisiones sólo contienen una serie limitada de soluciones y ninguna modificación de las API. En general las aplicaciones no indican qué revisión requieren. No obstante, una solución incluida en una revisión puede resolver un problema con una aplicación. En esta situación, una aplicación puede especificar un valor para el atributo `minimumPatchLevel` para asegurar que la revisión se aplique antes de instalar la aplicación. En caso necesario, el instalador de la aplicación de AIR indica al usuario que descargue e instale la versión o revisión que se requiera. El ejemplo siguiente muestra un elemento de la aplicación que especifica un valor para el atributo `minimumPatchLevel`:

```

<application version="1.0"
  xmlns="http://ns.adobe.com/air/application/1.1"
  minimumPatchLevel="5331">

```

Definición de los datos básicos de la aplicación

Los elementos siguientes definen el ID, la versión, el nombre, el nombre de archivo, la descripción y el aviso de copyright de la aplicación:

```
<id>com.example.samples.TestApp</id>
<version>2.0</version>
<filename>TestApp</filename>
<name>
  <text xml:lang="en">Hello AIR</text>
  <text xml:lang="fr">Bonjour AIR</text>
  <text xml:lang="es">Hola AIR</text>
</name>
<description>An MP3 player.</description>
<copyright>Copyright (c) 2008 YourCompany, Inc.</copyright>
```

id Una cadena de identificación exclusiva de la aplicación, conocida como ID de la aplicación. El valor del atributo se limita a los siguientes caracteres:

- 0–9
- a–z
- A–Z
- . (punto)
- - (guión)

El valor debe contener entre 1 y 212 caracteres. Este elemento es obligatorio.

La cadena `id` normalmente utiliza una jerarquía separada por puntos, en consonancia con una dirección de dominio DNS inversa, un nombre de clase o paquete de Java™ o un identificador de tipo universal de Mac OS® X. La forma tipo DNS no es obligatoria y AIR no crea ninguna asociación entre el nombre y los dominios DNS reales.

version Especifica la información sobre la versión para la aplicación. (No tiene nada que ver con la versión del motor de ejecución). La cadena de la versión es un designador definido por la aplicación. AIR no realiza ninguna interpretación de la cadena de la versión. Por ejemplo, no supone que la versión “3.0” es más actualizada que la versión “2.0”. Ejemplos: "1.0", ".4", "0.5", "4.9", "1.3.4a". Este elemento es obligatorio.

filename La cadena que se deberá utilizar como nombre de archivo de la aplicación (sin la extensión) al instalar ésta. El archivo de la aplicación inicia la aplicación de AIR en el motor de ejecución. Si no se facilita ningún valor para `name`, el nombre de archivo (valor de `filename`) se utilizará también como nombre de la carpeta de instalación. Este elemento es obligatorio.

La propiedad `filename` puede contener cualquier carácter Unicode (UTF-8) salvo los siguientes, cuyo uso en nombres de archivo está prohibido en diversos sistemas de archivos:

Carácter	Código hexadecimal
<i>diversos</i>	0x00 – x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E

Carácter	Código hexadecimal
?	x3F
\	x5C
	x7C

El valor de `filename` no puede terminar con un punto.

name (opcional, pero recomendado) El título que aparece en el instalador de aplicaciones de AIR.

Si especifica un solo nodo de texto (en lugar de varios elementos `text`), el instalador de aplicaciones de AIR utiliza este nombre, cualquiera que sea el idioma seleccionado para el sistema:

```
<name>Test Application</name>
```

El esquema del descriptor de aplicaciones de AIR 1.0 sólo permite definir un nodo de texto para el nombre (y no varios elementos de `text`).

En AIR 1.1 se pueden especificar varios idiomas en el elemento `name`. En el ejemplo siguiente se especifica el nombre en tres idiomas (inglés, francés y español):

```
<name>
  <text xml:lang="en">Hello AIR</text>
  <text xml:lang="fr">Bonjour AIR</text>
  <text xml:lang="es">Hola AIR</text>
</name>
```

El atributo `xml:lang` para cada elemento de texto especifica un código de idioma, de conformidad con lo definido en [RFC4646](http://www.ietf.org/rfc/rfc4646.txt) (<http://www.ietf.org/rfc/rfc4646.txt>).

El instalador de aplicaciones de AIR utiliza el nombre que mejor se corresponde con el idioma de la interfaz de usuario del sistema operativo del usuario. Por ejemplo, tomemos una instalación en la que el elemento `name` del archivo descriptor de la aplicación incluye un valor para la configuración regional "es" (español). El instalador de aplicaciones de AIR utiliza el nombre "es" si el sistema operativo identifica "es" (español) como el idioma de la interfaz de usuario. También utiliza el nombre "es" si el idioma de la interfaz de usuario del sistema es es-ES (español de España). Sin embargo, si el idioma de la interfaz de usuario es es-ES y el archivo descriptor de la aplicación define tanto el nombre es-ES como el nombre es-UY, el instalador de aplicaciones de AIR utiliza el valor es-ES. Si la aplicación no define ningún nombre que coincide con el idioma de la interfaz de usuario del sistema, el instalador de aplicaciones de AIR utiliza el primer valor de `name` que se define en el archivo descriptor de la aplicación.

Si no se especifica ningún elemento `name`, el instalador de aplicaciones de AIR muestra el nombre de archivo definido para `filename` como el nombre de la aplicación.

El elemento `name` sólo define el título de la aplicación que se utiliza en el instalador de aplicaciones de AIR. El instalador de aplicaciones de AIR 1.1 admite varios idiomas: alemán, chino tradicional, chino simplificado, coreano, español, francés, inglés, italiano, japonés, portugués brasileño y ruso. El instalador de aplicaciones de AIR selecciona el idioma visualizado (para texto que no sea el título y la descripción de la aplicación) con base en el idioma de la interfaz de usuario del sistema. Esta selección de idioma es independiente de la configuración del archivo descriptor de la aplicación.

El elemento `nameo` define las configuraciones regionales disponibles para la aplicación instalada y en funcionamiento. Para obtener más información sobre el desarrollo de aplicaciones multilingües, consulte [“Localización de aplicaciones de AIR”](#) en la página 312.

description (opcional) La descripción de la aplicación. Se muestra en el instalador de aplicaciones de AIR.

Si especifica un solo nodo de texto (en lugar de varios elementos para text), el instalador de aplicaciones de AIR utiliza este nombre, cualquiera que sea el idioma seleccionado para el sistema:

```
<description>This is a sample AIR application.</description>
```

El esquema del descriptor de aplicaciones de AIR 1.0 sólo permite definir un nodo de texto para el nombre (y no varios elementos de text).

En AIR 1.1 se pueden especificar varios idiomas en el elemento `description`. En el ejemplo siguiente se especifica una descripción en tres idiomas (inglés, francés y español):

```
<description>
  <text xml:lang="en">This is a example.</text>
  <text xml:lang="fr">C'est un exemple.</text>
  <text xml:lang="es">Esto es un ejemplo.</text>
</description>
```

El atributo `xml:lang` para cada elemento de texto especifica un código de idioma, de conformidad con lo definido en [RFC4646](http://www.ietf.org/rfc/rfc4646.txt) (<http://www.ietf.org/rfc/rfc4646.txt>).

El instalador de aplicaciones de AIR utiliza la descripción que es más similar al idioma de la interfaz de usuario del sistema operativo del usuario. Por ejemplo, tomemos una instalación en la que el elemento `description` del archivo descriptor de la aplicación incluye un valor para la configuración regional "es" (española). El instalador de aplicaciones de AIR utiliza el nombre "es" si el sistema del usuario identifica "es" (español) como el idioma de la interfaz de usuario. También utiliza el nombre "es" si el idioma de la interfaz de usuario del sistema es es-ES (español de España). Sin embargo, si el idioma de la interfaz de usuario del sistema es es-ES y el archivo descriptor de la aplicación define tanto el nombre es-ES como el nombre es-UY, el instalador de aplicaciones de AIR utiliza el valor es-ES. Si la aplicación no define ningún nombre que coincide con el idioma de la interfaz de usuario del sistema, el instalador de aplicaciones de AIR utiliza el primer valor de `description` que se define en el archivo descriptor de la aplicación.

Para obtener más información sobre el desarrollo de aplicaciones multilingües, consulte “[Localización de aplicaciones de AIR](#)” en la página 312.

copyright (opcional) La información de copyright para la aplicación de AIR. En Mac OS el aviso de copyright aparece en el cuadro de diálogo Acerca de para la aplicación instalada. En Mac OS, la información de copyright también se utiliza en el campo `NSHumanReadableCopyright` del archivo `Info.plist` para la aplicación.

Definición de la carpeta de instalación y la carpeta de menús de programa

Las carpetas de instalación y de menús de programa se definen con la siguiente configuración de propiedades:

```
<installFolder>Acme</installFolder>
<programMenuFolder>Acme/Applications</programMenuFolder>
```

installFolder (opcional) Identifica el subdirectorio del directorio de instalación predeterminado.

En Windows el subdirectorio de instalación predeterminado es el directorio Archivos de programa. En Mac OS es el directorio /Aplicaciones. Por ejemplo, si la propiedad `installFolder` está definida en "Acme" y una aplicación lleva el nombre "EjemploApl", la aplicación se instala en `C:\Archivos de programa\Acme\EjemploApl` en Windows y en `/Aplicaciones/Acme/Ejemplo.apl` en Mac OS.

Utilice la barra diagonal (/) como carácter separador entre directorios si desea especificar un subdirectorio anidado, como en el ejemplo siguiente:

```
<installFolder>Acme/Power Tools</installFolder>
```

La propiedad `installFolder` puede contener cualquier carácter Unicode (UTF-8) excepto aquellos cuyo uso en nombres de archivo está prohibido en diversos sistemas de archivos (para ver la lista de excepciones, consulte la anterior propiedad `filename`).

La propiedad `installFolder` es opcional. Si no se especifica ninguna propiedad para `installFolder`, la aplicación se instala en un subdirectorio del directorio de instalación predeterminado basado en la propiedad `name`.

programMenuFolder (Opcional) Identifica dónde deben guardarse los accesos directos a la aplicación en el menú Todos los programas del sistema operativo Windows. (En otros sistemas operativos, en la actualidad se hace caso omiso a esta opción). Las restricciones sobre el número de caracteres que se admiten en el valor de la propiedad son las mismas que para la propiedad `installFolder`. No utilice una barra diagonal (/) como último carácter de este valor.

Definición de las propiedades de la ventana inicial de la aplicación

Cuando se carga una aplicación de AIR, el motor de ejecución utiliza los valores del elemento `initialWindow` para crear la ventana inicial de la aplicación. Luego el motor de ejecución carga en la ventana el archivo SWF o HTML especificado en el elemento `content`.

El siguiente es un ejemplo del elemento `initialWindow`:

```
<initialWindow>
  <content>AIRTunes.swf</content>
  <title>AIR Tunes</title>
  <systemChrome>none</systemChrome>
  <transparent>true</transparent>
  <visible>true</visible>
  <minimizable>true</minimizable>
  <maximizable>true</maximizable>
  <resizable>true</resizable>
  <width>400</width>
  <height>600</height>
  <x>150</x>
  <y>150</y>
  <minSize>300 300</minSize>
  <maxSize>800 800</maxSize>
</initialWindow>
```

Los elementos secundarios del elemento `initialWindow` definen las propiedades de la ventana en la que se carga el archivo de contenido raíz.

content El valor especificado para el elemento `content` es la URL para el archivo principal de contenido de la aplicación. Éste será un archivo SWF o un archivo HTML. La URL se especifica relativa a la raíz de la carpeta de instalación de la aplicación. (Al ejecutar una aplicación de AIR con ADL, la URL se indica relativa a la carpeta que contiene el archivo descriptor de la aplicación. Se puede utilizar el parámetro `root-dir` de ADL para especificar otro directorio raíz).

Nota: al tratarse como una URL el valor del elemento `content`, los caracteres del nombre del archivo de contenido deben codificarse como URL de acuerdo con las reglas definidas en [RFC 1738](#). Los caracteres de espacio, por ejemplo, deben codificarse como `%20`.

title (Opcional) El título de la ventana.

systemChrome (Opcional) Si este atributo se define en `standard`, se mostrará el fondo cromático estándar proporcionado por el sistema operativo. Si lo define en `none`, no se mostrará ningún fondo cromático del sistema. La opción de fondo cromático no puede modificarse durante el tiempo de ejecución.

transparent (opcional) Defínalo en "true" si desea que la ventana de la aplicación admita la mezcla alfa. Una ventana con transparencia puede dibujarse más lentamente y necesitar más memoria. La opción de transparencia no puede modificarse durante el tiempo de ejecución.

Importante: Sólo se puede definir `transparent` en `true` si `systemChrome` tiene el valor.

visible (opcional) Defínalo en `true` si desea que la ventana principal quede visible en cuanto se haya creado. El valor predeterminado es `false`.

Puede convenir dejar la ventana principal oculta al principio para que no se muestren los ajustes de la posición y el tamaño de la ventana y la disposición del contenido. Se podrá mostrar después la ventana llamando al método `activate()` de la ventana o cambiando la propiedad `visible` a `true`. Para obtener más información, consulte "Trabajo con ventanas nativas" en la página 57.

x, y, width, height (opcionales) Límites iniciales de la ventana principal de la aplicación. Si no se definen estos valores, el tamaño de la ventana quedará definido por las opciones del archivo SWF raíz o, en el caso de HTML, por el sistema operativo.

minSize, maxSize (opcionales) Tamaños mínimo y máximo de la ventana. Si no se definen estos valores, los determinará el sistema operativo.

minimizable, maximizable, resizable (opcionales) Especifican si se puede minimizar, maximizar y redimensionar la ventana. Estas opciones tienen el valor predeterminado `true`.

Nota: en los sistemas operativos como Mac OS X en los cuales la maximización de las ventanas es una operación de redimensionamiento, para que la ventana no cambie de tamaño, tanto "maximizable" como "resizable" deben definirse en `false`.

Especificación de archivos de iconos

La propiedad `icon` especifica un archivo de icono (o varios) a utilizar para la aplicación. Los iconos son opcionales. Si no se define la propiedad `icon`, el sistema operativo muestra un icono predeterminado.

La ruta se indica relativa al directorio raíz de la aplicación. Los archivos de iconos deben tener el formato PNG. Se pueden especificar todos los tamaños de icono siguientes:

```
<icon>
  <image16x16>icons/smallIcon.png</image16x16>
  <image32x32>icons/mediumIcon.png</image32x32>
  <image48x48>icons/bigIcon.png</image48x48>
  <image128x128>icons/biggestIcon.png</image128x128>
</icon>
```

Si hay un elemento para un tamaño determinado, la imagen que contiene el archivo debe ser exactamente del tamaño especificado. Si no se proporcionan todos los tamaños, el sistema operativo ajusta el tamaño más parecido para un uso determinado del icono.

Nota: los iconos especificados no se añaden automáticamente al paquete de AIR. Los archivos de iconos deben estar incluidos en sus lugares relativos correctos cuando se empaqueta de la aplicación.

Para obtener el mejor resultado posible, proporcione una imagen para cada uno de los tamaños disponibles. Asegúrese también de que los iconos sean de buen aspecto tanto en el modo de color de 16 bits como en el de 32 bits.

Interfaz de usuario personalizada para actualizaciones

AIR instala y actualiza las aplicaciones utilizando los cuadros de diálogo de instalación predeterminados. No obstante, puede proporcionar su propia interfaz para actualizar una aplicación. Para indicar que la aplicación debe encargarse del proceso de actualización, defina el elemento `customUpdateUI` en `true`:

```
<customUpdateUI>true</customUpdateUI>
```

Cuando la versión instalada de la aplicación tiene el elemento `customUpdateUI` definido en `true` y el usuario hace doble clic en el archivo de AIR para una nueva versión o instala una actualización de la aplicación utilizando la función de instalación integrada, el motor de ejecución abre la versión instalada de la aplicación en lugar del instalador de aplicaciones de AIR predeterminado. La lógica de la aplicación determina entonces cómo continuar con la operación de actualización. (Para que se lleve a cabo una actualización, el ID de la aplicación y el ID del editor que figuran en el archivo de AIR deben coincidir con los de la aplicación instalada).

Nota: el mecanismo `customUpdateUI` sólo entra en juego si la aplicación ya está instalada y el usuario hace doble clic en el archivo de instalación de AIR que contiene una actualización, o si instala una actualización de la aplicación utilizando la función de instalación integrada. Puede descargar una actualización e iniciarla a través de la lógica de su propia aplicación, visualizando la interfaz de usuario personalizada según convenga, esté o no `customUpdateUI` definido en `true`.

Para obtener más información, consulte “[Actualización de aplicaciones de AIR](#)” en la página 308.

Inicio de la aplicación desde el navegador

Si se especifica la siguiente opción, se podrá iniciar la aplicación de AIR instalada utilizando la función de invocación desde el navegador (el usuario selecciona un vínculo en una página en el navegador Web):

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

El valor predeterminado es `false`.

Si este valor se define en `true`, asegúrese de tener en cuenta las posibles consecuencias para la seguridad, descritas en “[Invocación desde el navegador](#)” en la página 274.

Para obtener más información, consulte “[Instalación y ejecución de aplicaciones de AIR desde una página Web](#)” en la página 293.

Declaración de asociaciones con tipos de archivos

El elemento `fileTypes` permite declarar los tipos de archivos con que se puede asociar una aplicación de AIR. Cuando se instala una aplicación de AIR, los tipos de archivos declarados se registran en el sistema operativo y, si estos tipos de archivos aún no se encuentran asociados con ninguna otra aplicación, se asocian con la aplicación de AIR. Para suprimir una asociación existente entre un tipo de archivo y otra aplicación, utilice el método en tiempo de ejecución `NativeApplication.setAsDefaultApplication()` (preferentemente con el permiso del usuario).

Nota: los métodos del motor de ejecución sólo pueden manejar asociaciones para los tipos de archivos declarados en el descriptor de la aplicación.

```
<fileTypes>
  <fileType>
    <name>adobe.VideoFile</name>
    <extension>avf</extension>
    <description>Adobe Video File</description>
    <contentType>application/vnd.adobe.video-file</contentType>
    <icon>
      <image16x16>icons/AIRApp_16.png</image16x16>
      <image32x32>icons/AIRApp_32.png</image32x32>
      <image48x48>icons/AIRApp_48.png</image48x48>
      <image128x128>icons/AIRApp_128.png</image128x128>
    </icon>
  </fileType>
</fileTypes>
```

El elemento `fileTypes` es opcional. Si está presente, puede contener varios elementos `fileType`.

Los elementos `name` y `extension` son obligatorios para cada declaración de `fileType` que se incluya. Se puede utilizar el mismo nombre con varias extensiones. La extensión identifica el tipo de archivo. (Cabe observar que la extensión se especifica sin el punto que la precede). El elemento `description` es opcional; lo presenta al usuario la interfaz de usuario del sistema operativo. La propiedad `contentType` también es optativa; ayuda al sistema operativo a localizar la mejor aplicación para abrir un archivo en determinadas circunstancias. El valor debe ser el tipo MIME del contenido del archivo.

Se pueden especificar iconos para la extensión de archivo utilizando el mismo formato que el elemento "icon" de la aplicación. Los archivos de iconos también deben incluirse en el archivo de instalación de AIR (no se colocan automáticamente en el paquete).

Si hay un tipo de archivo asociado con una aplicación de AIR, cada vez que un usuario abra un archivo de ese tipo se invocará la aplicación. Si la aplicación ya está funcionando, AIR distribuye el objeto `InvokeEvent` a la instancia que se esté ejecutando. En caso contrario, AIR iniciará primero la aplicación. En ambos casos la ruta al archivo puede recuperarse del objeto `InvokeEvent` distribuido por el objeto `NativeApplication`. Puede utilizarse esta ruta para abrir el archivo.

Para obtener más información, consulte [“Gestión de asociaciones con archivos”](#) en la página 280 y [“Captura de argumentos de la línea de comandos”](#) en la página 272.

Capítulo 9: Nuevas funciones en Adobe AIR

En este capítulo se presenta información general sobre las nuevas funciones de Adobe® AIR™ que no están disponibles para el contenido SWF que se ejecuta en Adobe® Flash® Player.

Nuevas clases del motor de ejecución

Las siguientes clases del motor de ejecución son nuevas para Adobe AIR. No están disponibles para el contenido SWF que se ejecute en el navegador:

Clase	Paquete
BrowserInvokeEvent	flash.events
Clipboard	flash.desktop
ClipboardFormats	flash.desktop
ClipboardTransferMode	flash.desktop
CompressionAlgorithm	flash.utils
DockIcon	flash.desktop
DRMAuthenticateEvent	flash.events
DRMErrorEvent	flash.events
DRMStatusEvent	flash.events
EncryptedLocalStore	flash.data
File	flash.filesystem
FileListEvent	flash.events
FileMode	flash.filesystem
FileStream	flash.filesystem
FocusDirection	flash.display
HTMLHistoryItem	flash.html
HTMLHost	flash.html
HTMLLoader	flash.html
HTMLPDFCapability	flash.html
HTMLUncaughtScriptExceptionEvent	flash.events
HTMLWindowCreateOptions	flash.html
Icon	flash.desktop
InteractiveIcon	flash.desktop

Clase	Paquete
InvokeEvent	flash.events
NativeApplication	flash.desktop
NativeDragActions	flash.desktop
NativeDragEvent	flash.events
NativeDragManager	flash.desktop
NativeDragOptions	flash.desktop
NativeMenu	flash.display
NativeMenuItem	flash.display
NativeWindow	flash.display
NativeWindowBoundsEvent	flash.events
NativeWindowDisplayState	flash.display
NativeWindowDisplayStateEvent	flash.events
NativeWindowInitOptions	flash.display
NativeWindowResize	flash.display
NativeWindowSystemChrome	flash.display
NativeWindowType	flash.display
NotificationType	flash.desktop
OutputProgressEvent	flash.events
RevocationCheckSettings	flash.security
Screen	flash.display
ScreenMouseEvent	flash.events
SignatureStatus	flash.security
SignerTrustSettings	flash.security
SQLCollationType	flash.data
SQLColumnNameStyle	flash.data
SQLColumnSchema	flash.data
SQLConnection	flash.data
SQLException	flash.errors
SQLExceptionEvent	flash.events
SQLExceptionOperation	flash.errors
SQLEvent	flash.events
SQLIndexSchema	flash.data
SQLResult	flash.data
SQLSchema	flash.data

Clase	Paquete
SQLSchemaResult	flash.data
SQLStatement	flash.data
SQLTableSchema	flash.data
SQLTransactionLockType	flash.data
SQLTriggerSchema	flash.data
SQLUpdateEvent	flash.events
SQLViewSchema	flash.data
SystemTrayIcon	flash.desktop
Updater	flash.desktop
URLRequestDefaults	flash.net
XMLSignatureValidator	flash.utils

Además, el paquete flash.security incluye la interfaz [IURIDreferencer](#).

Clases del motor de ejecución con nuevas funciones

Las siguientes clases están disponibles para el contenido SWF que se ejecuta en el navegador, pero AIR ofrece propiedades o métodos adicionales:

Clase	Método o propiedad
Capabilities	languages
Event	DISPLAYING EXITING HTML_BOUNDS_CHANGE HTML_DOM_INITIALIZE HTML_RENDER LOCATION_CHANGE NETWORK_CHANGE USER_IDLE USER_PRESENT
FileReference	uploadUnencoded()
HTTPStatusEvent	HTTP_RESPONSE_STATUS responseURL responseHeaders
KeyboardEvent	commandKey controlKey
LoaderContext	allowLoadBytesCodeExecution

Clase	Método o propiedad
LoaderInfo	parentSandboxBridge childSandboxBridge
NetStream	resetDRMVouchers() setDRMAuthenticationCredentials()
URLRequest	followRedirects manageCookies shouldAuthenticate shouldCacheResponse userAgent userCache setLoginCredentials()
URLStream	Evento httpResponseStatus
Stage	nativeWindow
Security	APPLICATION

La mayoría de estos nuevos métodos y propiedades sólo están disponibles para contenido que se encuentra en el entorno limitado de seguridad de la aplicación de AIR. No obstante, los nuevos integrantes de las clases `URLRequest` también están disponibles para el contenido que se ejecuta en otros entornos limitados.

Los métodos `ByteArray.compress()` y `ByteArray.uncompress()` incluyen cada uno un nuevo parámetro, `algorithm`, que permite seleccionar entre la compresión `deflate` y `zlib`.

Clases del marco de supervisión del servicio

El paquete `air.net` contiene clases para la detección a través de la red. Este paquete está disponible solamente para contenido que se ejecuta en Adobe AIR. Viene incluido en el archivo `ServiceMonitor.swc`.

El paquete incluye las siguientes clases:

- [ServiceMonitor](#)
- [SocketMonitor](#)
- [URLMonitor](#)

Capítulo 10: Trabajo con ventanas nativas

Puede utilizar las clases proporcionadas por la API de manejo de ventanas nativas de Adobe® AIR® para crear y gestionar ventanas de escritorio.

Información adicional sobre ventanas nativas

Dispone de información adicional sobre la API de manejo de ventanas nativas y el proceso de trabajo con ellas en las siguientes referencias:

Inicios rápidos (Centro de desarrollo de Adobe AIR)

- [Interacción con una ventana \(en inglés\)](#)
- [Personalización del aspecto de una ventana nativa \(en inglés\)](#)
- [Creación de ventanas superpuestas](#)
- [Control del orden de visualización de las ventanas](#)
- [Creación de ventanas no rectangulares de tamaño modificable](#)

Referencia del lenguaje

- [NativeWindow](#)
- [NativeWindowInitOptions](#)

Artículos y ejemplos del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR windows"](#)

Aspectos básicos de ventanas de AIR

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 creadas pueden tener el aspecto de una aplicación estándar de escritorio de Apple cuando se ejecutan en Mac o ajustarse a las convenciones de Microsoft cuando se ejecutan en Windows. 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 sus propias ventanas 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

AIR admite tres API distintas para trabajar con ventanas: la clase `NativeWindow` orientada a `ActionScript`, las clases `mx:WindowedApplication` y `mx:Window` de la arquitectura de Flex (que “agrupan” la clase `NativeWindow`) y, en el entorno HTML, la clase `Window` de `JavaScript`.

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 objeto de visualización del escenario.

Ventanas de la arquitectura de Flex

La arquitectura de Flex define sus propios componentes que agrupan la API `NativeWindow`. Estos componentes (`mx:WindowedApplication` y `mx:Window`) no se pueden utilizar fuera de la arquitectura, por lo que tampoco se pueden utilizar en aplicaciones de AIR desarrolladas con herramientas de edición de Flash.

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.*

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.

Para obtener más información sobre las propiedades de ventana especificadas en el descriptor de la aplicación, consulte [“Estructura del archivo descriptor de la aplicación”](#) en la página 44.

Clases de ventanas nativas

La API de gestión de ventanas nativas contiene las siguientes clases:

Paquete	Clases
flash.display	<ul style="list-style-type: none"> • NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType <p>Las constantes de cadena de la ventana se definen en las siguientes clases:</p> <ul style="list-style-type: none"> • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	<ul style="list-style-type: none"> • NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

Flujo de eventos de ventanas nativas

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 sólo 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, ésta sólo 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 el manual [Referencia del lenguaje y componentes ActionScript 3.0](http://www.adobe.com/go/learn_flash_aslr_es) (http://www.adobe.com/go/learn_flash_aslr_es).

Para obtener información general sobre la lista de visualización de Flash, consulte la sección “Programación de visualización” del manual [Programación con ActionScript 3.0](http://www.adobe.com/go/programmingAS3_es) (http://www.adobe.com/go/programmingAS3_es).

Propiedades que controlan el estilo y el comportamiento de una ventana nativa

Las siguientes propiedades controlan el aspecto y el comportamiento básicos de una ventana:

- `type`
- `systemChrome`
- `transparent`

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

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:

Tipo	Descripción
Normal	Una ventana normal. Las 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 <code>type</code> en ventanas ligeras, <code>systemChrome</code> debe establecerse como <code>none</code> .

Fondo cromático de una ventana

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 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.

Transparencia de la ventana

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. La representación de zonas grandes de mezcla alfa lleva su tiempo, por lo que el efecto debe utilizarse con precaución.

Las ventanas transparentes resultan útiles cuando se quieren crear aplicaciones con bordes de forma irregular, aplicaciones que “desaparecen” o aplicaciones invisibles.

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 243.

En algunos sistemas operativos, no se admite la transparencia debido a la configuración concreta del hardware o del software, o como consecuencia de las opciones de visualización del usuario. Si no se admite transparencia, la aplicación se compone con un fondo negro. En estos casos, cualquier área completamente transparente de la aplicación se visualiza en negro opaco.

La propiedad `NativeWindow.supportsTransparency` indica si la ventana puede ser transparente. Si esta propiedad es `false`, por ejemplo, puede mostrar un diálogo de advertencia al usuario, o una interfaz rectangular, básica y no transparente. Tenga en cuenta que la transparencia se admite siempre en los sistemas operativos Mac y Windows. La compatibilidad de transparencia en sistemas operativos Linux requiere un gestor de composición de ventanas, pero no siempre está disponible debido a las opciones de visualización del usuario o a la configuración del hardware.

Transparencia en una ventana de aplicación HTML

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 html:HTMLLoader = new HTMLLoader();  
html.paintsDefaultBackground = false;
```

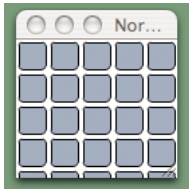

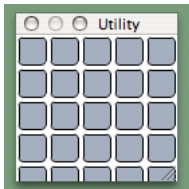

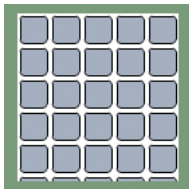
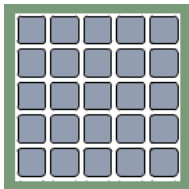
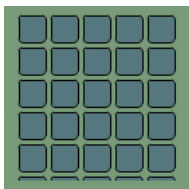
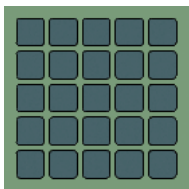

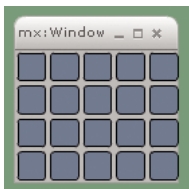
Este ejemplo utiliza JavaScript para desactivar el fondo predeterminado de una ventana HTML:

```
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.

Catálogo de ventana visual

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 y Windows:

Parámetros de ventana	Mac OS X	Microsoft Windows
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		

Nota: los siguientes elementos del fondo cromático del sistema no se admiten en AIR: la barra de herramientas de OS X, el icono proxy de OS X, los iconos de barra de título de Windows y otros fondos cromáticos alternativos.

Creación de ventanas

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.

Especificación de propiedades de inicialización de una ventana

Las propiedades de inicialización de una ventana 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
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 252 para obtener más información.

Creación de la ventana inicial de la aplicación

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 contenido debe ser 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.

La herramienta de edición de Flash crea automáticamente el archivo SWF y añade la referencia adecuada al descriptor de la aplicación cuando se prueba o se publica un proyecto de AIR. La línea de tiempo principal sirve de punto de entrada para la aplicación.

Cuando se inicia la aplicación, AIR crea una ventana y carga el archivo SWF de la aplicación. Para controlar la ventana de escritorio con ActionScript, utilice la propiedad `nativeWindow` del objeto `Stage` para obtener una referencia al objeto `NativeWindow`. Seguidamente, puede establecer las propiedades de la ventana y empezar a llamar a los métodos.

El siguiente ejemplo activa la ventana principal maximizada (desde el primer fotograma de un archivo FLA de Flash):

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

Creación de una ventana nativa

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.

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

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()` sólo 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("xml.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 247.

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`.

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.

Cómo añadir contenido a una ventana

La forma en que se añade contenido a las ventanas de AIR depende del tipo de ventana. Puede crear un clip de película y utilizar la línea de tiempo para controlar el estado de la aplicación. Con HTML, puede declarar definiciones del contenido básico de la ventana. Puede incorporar recursos en el archivo SWF de la aplicación o cargarlos desde distintos archivos de aplicación. El contenido de 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 la aplicación para que el código de dicha página pueda utilizar el contenido externo de los scripts. Consulte “[Seguridad en AIR](#)” en la página 23.

Carga de un archivo SWF o una imagen

Puede cargar contenido 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);
        }
    }
}

```

Es posible cargar un archivo SWF que contenga código de biblioteca para utilizarlo en una aplicación basada en HTML. El modo más sencillo de cargar un archivo SWF en una ventana HTML es utilizar la etiqueta `script`, pero también puede utilizarse la API `Loader` directamente.

Nota: los archivos SWF antiguos creados con *ActionScript 1 ó 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 ) );

```

Nota: el contenido SWF o PDF de un archivo HTML no se visualiza si la ventana utiliza transparencia (es decir, si la propiedad `transparent` de la ventana está establecida como `true`) o si se escala el control `HTMLLoader`.

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 `HTMLLoader` 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 la sección de programación de visualización del manual [Programación con ActionScript 3.0](#) para obtener más información.

Ejemplo: Creación de una ventana nativa

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

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.

Obtención de una instancia de `NativeWindow`

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 la ventana Por ejemplo, el constructor de la ventana del nuevo elemento `NativeWindow`.

El escenario de la ventana Por ejemplo, `stage.nativeWindow`.

Cualquier objeto de visualización del escenario Por ejemplo, `myDisplayObject.stage.nativeWindow`.

Un evento de ventana La propiedad `target` del objeto de evento al que hace referencia la ventana distribuida por el evento.

La propiedad global `nativeWindow` de un objeto `HTMLLoader` o ventana `HTML` Por ejemplo, `window.nativeWindow`.

El objeto `nativeApplication` `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.

Activación, visualización y ocultación de ventanas

Para activar una ventana, llame al método `activate()` de `NativeWindow`. Al activar una ventana, ésta 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, ésta 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.

Nota: en Mac OS X, no es posible ocultar completamente una ventana minimizada si ésta tiene un icono en el 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

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, sólo se debe establecer `alwaysInFront` como `true` cuando sea realmente necesario. Algunos casos de ejemplos justificados son :

- 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.

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 <code>alwaysInFront</code>	Especifica si la ventana se muestra en el grupo superior de ventanas. En casi todos los casos, el mejor ajuste es <code>false</code> . Si cambia el valor de <code>false</code> a <code>true</code> , la ventana se coloca delante del resto (aunque no se activa). Si cambia el valor de <code>true</code> a <code>false</code> , 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.
<code>orderToFront()</code>	Trae la ventana al frente.
<code>orderInFrontOf()</code>	Coloca la ventana directamente delante de una ventana concreta.
<code>orderToBack()</code>	Envía la ventana detrás del resto de ventanas.
<code>orderBehind()</code>	Envía la ventana directamente detrás de una ventana concreta.
<code>activate()</code>	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.

Cómo cerrar una ventana

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, éste 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.

Permiso para cancelar operaciones con ventanas

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

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();
```

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.

Ejemplo: Minimizar, maximizar, restaurar y cerrar una ventana

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

Si una ventana utiliza fondo cromático del sistema, éste 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 69.*

Cambio de tamaño de una ventana

Para cambiar el tamaño de una ventana, 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.

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` 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 `NativeWindow.startMove()`. 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 el manual [Referencia del lenguaje y componentes ActionScript 3.0](http://www.adobe.com/go/learn_air_aslr_es) (http://www.adobe.com/go/learn_air_aslr_es).

Ejemplo: Cambio de tamaño y desplazamiento de ventanas

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(0xFFFFFFFF);
            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.CLICK, 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.CLICK, 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

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, se distribuye el mensaje del evento `closing`. Si ningún detector cancela el evento, la ventana se cierra y el evento `close` se distribuye a todos los detectores.

Normalmente, los eventos de advertencia como `closing` sólo 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` (sólo 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
<code>activate</code>	Se distribuye cuando la ventana recibe la selección.
<code>deactivate</code>	Se distribuye cuando la ventana deja de recibir la selección.
<code>closing</code>	Se distribuye cuando la ventana va a cerrarse. Esto sólo ocurre automáticamente al pulsar el botón de cierre del fondo cromático o, en Mac OS X, al invocar el comando Salir.
<code>close</code>	Se distribuye cuando la ventana se ha cerrado.

Los eventos de ventana que distribuye un objeto `NativeWindowBoundsEvent` son:

Evento	Descripción
<code>moving</code>	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.
<code>move</code>	Se distribuye una vez que la esquina superior izquierda de la ventana ha cambiado de posición.
<code>resizing</code>	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.
<code>resize</code>	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
<code>displayStateChanging</code>	Se distribuye inmediatamente antes de que cambie el estado de visualización de la ventana.
<code>displayStateChange</code>	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.

Visualización de ventanas a pantalla completa

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.

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 11: Pantallas

Para ver información sobre las pantallas del escritorio que estén conectadas a un ordenador, utilice la clase `Screen` de Adobe® AIR®.

Información suplementaria en línea sobre las pantallas

Encontrará más información sobre la clase `Screen` y cómo trabajar con las pantallas en las fuentes siguientes:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Medición del escritorio virtual \(en inglés\)](#)

Referencia del lenguaje

- [Screen](#)

Artículos y muestras del Centro de desarrollo de Adobe

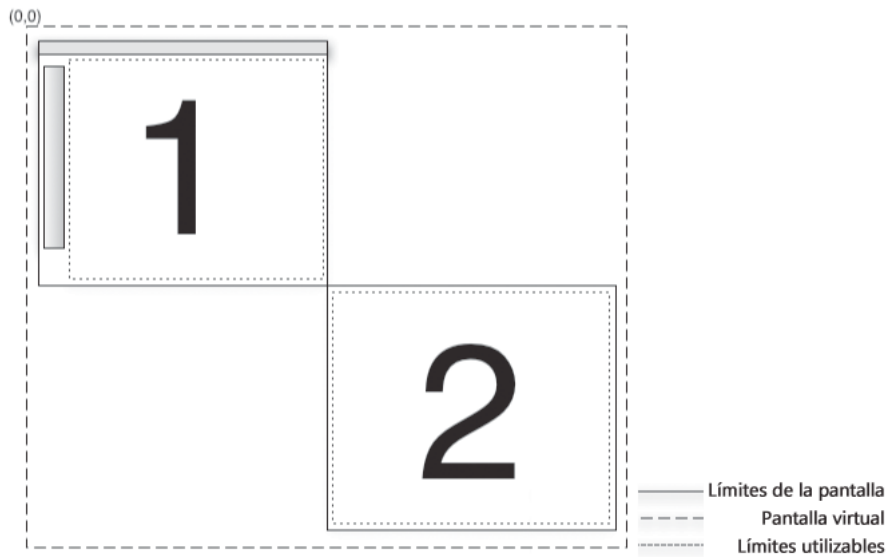
- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR screens"](#)

Aspectos básicos de las pantallas

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, los métodos, las propiedades y los eventos de la API de la pantalla, consulte [Referencia del lenguaje y componentes ActionScript 3](http://www.adobe.com/go/learn_air_aslr_es). (http://www.adobe.com/go/learn_air_aslr_es).

Enumeración de las pantallas

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. Obsérvese que el orden del conjunto no tiene importancia.
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 12: Trabajo con menús nativos

Se deben utilizar las clases en la API del menú nativo para definir los menús de aplicación, ventana, contextuales y emergentes.

Información adicional en línea de menús nativos

Se puede encontrar más información sobre la API de menú nativo y la utilización de menús nativos de los siguientes recursos:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Adición de menús nativos a una aplicación de AIR \(en inglés\)](#)

Referencia del lenguaje

- [NativeMenu](#)
- [NativeMenuItem](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flex \(en inglés\); busque "AIR menus"](#)

Aspectos básicos del menú AIR

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), menús contextuales y menús emergentes.

Clases del menú AIR

Las clases de Menú AIR™ de Adobe® incluyen:

Paquete	Clases
flash.display	<ul style="list-style-type: none"> • NativeMenu • NativeMenuItem
flash.ui	<ul style="list-style-type: none"> • ContextMenu • ContextMenuItem
flash.events	<ul style="list-style-type: none"> • Evento • ContextMenuEvent

Variedades de menú

AIR admite los siguientes tipos de menús:

Menús de aplicación 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. 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 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 el sistema operativo Windows, pero no en Mac OS X. Los menús nativos solo se pueden utilizar con ventanas que tienen fondo cromático del sistema.

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. Se puede crear un menú contextual utilizando la clase `NativeMenu` de AIR. (También se puede usar la clase heredada `ContextMenu` de Adobe® Flash®). En el contenido HTML, se puede utilizar el Webkit HTML y las API JavaScript para añadir menús contextuales a un elemento HTML.

Menús del icono de acoplamiento y bandeja del sistema Estos menús de iconos son similares a los menús contextuales y se asignan a un icono de aplicación en el área de notificación de Windows o en el acoplamiento de Mac OS X. 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, no hay un menú estándar.

Menús emergentes 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. Se pueden crear los propios menús no nativos utilizando MXML, ActionScript o JavaScript. Las clases de menú AIR no proporcionan ninguna capacidad para controlar la invocación de menús nativos.

Menús predeterminados

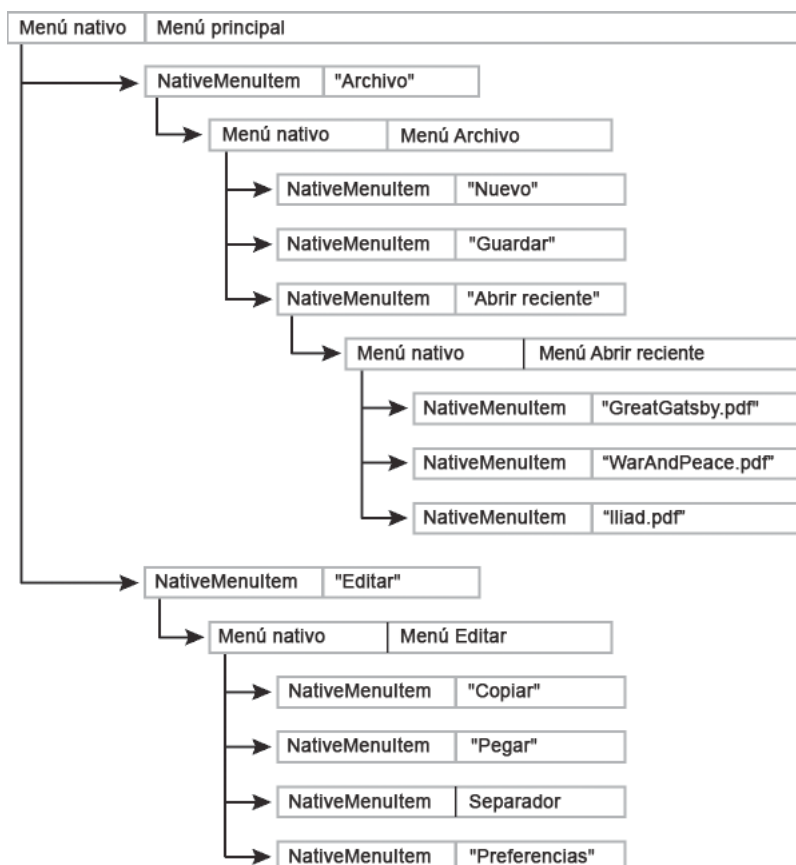
El sistema operativo o una clase incorporada de AIR proporciona los siguientes menús predeterminados:

- 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`)

Estructura de los menús

Los menús, 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: Menú de ventana y de aplicación”](#) en la página 93.

Eventos de menú

Los objetos `NativeMenu` y `NativeMenuItem` ambos distribuyen eventos `displaying` y `select`:

Displaying: Inmediatamente antes de visualizar un menú, el menú y los elementos de menú distribuyen un evento `displaying` a cualquier detector registrado. El evento `displaying` proporciona 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.

La propiedad `target` del objeto de evento es siempre el menú que se está por mostrar. `currentTarget` es el objeto donde el detector está registrado: ya sea el mismo menú o uno de sus elementos.

Nota: también se distribuye el evento `displaying` cuando se accede al estado del menú o a uno de sus elementos.

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` y `displaying`.

Equivalentes de teclas para comandos de menú

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, 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

Las letras de selección son parte de la interfaz del teclado del sistema operativo a los menús. Tanto Mac OS X como 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.

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, se aconseja asignar una letra de selección exclusiva para cada elemento en un menú para los menús de ventana.

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

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

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

Este tema describe la manera de crear los diferentes tipos de menús nativos admitidos en AIR.

Creación de un objeto de menú raíz

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

```
NativeApplication.nativeApplication.menu = root;
```

Nota: Mac OS X 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 reemplaza el menú estándar. También se puede utilizar el menú estándar en lugar de sustituirlo.

Definición de un menú de ventana

```
nativeWindowObject.menu = root;
```

Definición de un menú contextual en un objeto interactivo

```
interactiveObject.contextMenu = root;
```

Definición de un menú de icono de acoplamiento

```
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.

Definición de un menú de icono de bandeja del sistema

```
SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
```

Visualización de un menú como un menú emergente

```
root.display(stage, x, y);
```

Creación de un submenú

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

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

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 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`. El objeto de menú asignado a `contextMenu` puede ser del tipo `NativeMenu` o `ContextMenu`.

Las clases API de menú contextual heredadas permiten utilizar el código ActionScript existente que ya contiene menús contextuales. Si se utiliza la clase `ContextMenu` se debe utilizar la clase `ContextMenuItem`; no se pueden añadir objetos `NativeMenuItem` a un objeto `ContextMenu`, ni se pueden añadir objetos `ContextMenuItem` a un objeto `NativeMenu`. El inconveniente principal de utilizar la API de menú contextual es que no admite submenús.

Aunque la clase `ContextMenu` incluye métodos como `addItem()`, que se heredan de la clase `NativeMenu`, estos métodos añaden elementos al conjunto de elementos incorrecto. En un menú contextual, todos los elementos se deben añadir al conjunto `customItems` no al conjunto `elements`. Se deben utilizar objetos `NativeMenu` para menús contextuales o se deben utilizar solo métodos `ContextMenu` no heredados y propiedades para añadir y gestionar elementos en el menú.

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:ContextMenuItems = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItems = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItems = 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.

Menús contextuales en HTML

En el contenido HTML, 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` 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>
```

Definición de menús nativos de forma descriptiva

La codificación de las propiedades de un menú y de los elementos de menú puede ser un poco tedioso. Sin embargo, debido a que los menús tienen una estructura jerárquica natural, es muy simple escribir una función que crea un menú utilizando una definición con formato XML.

La siguiente clase amplía `NativeMenu`, tomando un objeto XML del constructor, para hacer justamente eso:


```
package
{
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.events.Event;

    public class DeclarativeMenu extends NativeMenu
    {
        public function DeclarativeMenu(XMLMenuDefinition:XML):void
        {
            super();
            addChildrenToMenu(this, XMLMenuDefinition.children());
        }

        private function addChildrenToMenu(menu:NativeMenu,
            children:XMLList):NativeMenuItem
        {
            var menuItem:NativeMenuItem;
            var submenu:NativeMenu;

            for each (var child:XML in children)
            {
                if (String(child.@label).length > 0)
                {
                    menuItem = new NativeMenuItem(child.@label);
                    menuItem.name = child.name();
                }
                else
                {
                    menuItem = new NativeMenuItem(child.name());
                    menuItem.name = child.name();
                }
                menu.addItem(menuItem);
                if (child.children().length() > 0)
                {
                    menuItem.submenu = new NativeMenu();
                    addChildrenToMenu(menuItem.submenu, child.children());
                }
            }
            return menuItem;
        }
    } //End class
} //End package
```

Para crear un menú con esta clase, se debe transmitir una definición de menú XML de la siguiente manera:

```
var menuDefinition:XML =
    <root>
        <FileMenu label='File'>
            <NewMenu label='New'>
                <NewTextFile label='Text file' />
                <NewFolder label='Folder' />
                <NewProject label='Project' />
            </NewMenu>
            <OpenCommand label='Open' />
            <SaveCommand label='Save' />
        </FileMenu>
        <EditMenu label='Edit'>
            <CutCommand label='Cut' />
            <CopyCommand label='Copy' />
            <PasteCommand label='Paste' />
        </EditMenu>
        <FoodItems label='Food Items'>
            <Jellyfish />
            <Tripe />
            <Gizzard />
        </FoodItems>
    </root>;
var test:DeclarativeMenu = new DeclarativeMenu(menuDefinition);
```

Para detectar eventos de menús, se puede detectar el nivel del menú raíz y utilizar la propiedad `event.target.name` para detectar el comando seleccionado. También se pueden buscar elementos en el menú por nombre y añadir detectores de eventos individuales.

Visualización de menús emergentes

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, sólo 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

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

Añada detectores de eventos a los menús o elementos individuales para gestionar eventos de menús.

Objeto	Eventos detectados
NativeMenu	NativeMenuEvent.DISPLAYING NativeMenuEvent.SELECT (propagado de elementos secundarios y submenús)
NativeMenuItem	NativeMenuEvent.SELECT NativeMenuEvent.DISPLAYING (propagado del menú principal)
ContextMenu	ContextMenuEvent.MENU_SELECT
ContextMenuItem	ContextMenuEvent.MENU_ITEM_SELECT NativeMenu.SELECT

Selección de eventos de menú

Para gestionar un clic en un elemento de menú, añade un detector de evento para el evento `select` al objeto `NativeMenuItem`:

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommand.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

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.

Ejemplo: Menú de ventana y de aplicación

En el siguiente ejemplo se crea el menú que se muestra en “Estructura de los menús” en la página 82.

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 13: Iconos de la barra de tareas

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`.

Información adicional en línea sobre los iconos de la barra de tareas

Puede encontrar más información sobre el funcionamiento de las barras de tareas en las siguientes referencias:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Iconos del Dock y de la bandeja del sistema](#) (en inglés)

Referencia del lenguaje

- [DockIcon](#)
- [SystemTrayIcon](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash](#) (en inglés); busque "AIR taskbar icons"

Iconos de la barra de tareas

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 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

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 sólo se aplican cuando la aplicación se está ejecutando (el icono recupera su aspecto normal cuando se cierra la aplicación).

Menús de iconos del Dock

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

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

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

AIR admite iconos de bandeja del sistema si `NativeApplication.supportsSystemTrayIcon` es `true` (esto sólo sucede en Windows). En Windows, 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

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

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

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 ésta no tenga ninguna ventana visible.

Ejemplo: Creación de una aplicación sin ventanas

El siguiente ejemplo crea una aplicación de AIR con un icono de bandeja del sistema pero sin ventanas visibles. El icono de la bandeja del sistema tiene un menú con un único comando que permite cerrar 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;
}
stage.nativeWindow.close();
}

private function iconLoadComplete(event:Event):void
{
    NativeApplication.nativeApplication.icon.bitmaps =
        [event.target.content.bitmapData];
}
}
}

```

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

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. La barra de tareas de Microsoft Windows muestra 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

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 97). En Windows, 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.

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

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 `<initialWindow>` del archivo descriptor de la aplicación (y no establezca la propiedad `visible` como `true` ni llame al método `activate()` de la ventana).

En las 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 14: Trabajo con el sistema de archivos

Las clases que proporciona la interfaz API del sistema de archivos de Adobe® AIR™ facilitan el acceso al sistema de archivos del ordenador host. Estas clases sirven para tener acceso a los directorios y archivos y gestionarlos, crear directorios y archivos, escribir datos en los archivos, etc.

Información suplementaria en línea sobre la API de archivos de AIR

Encontrará más información sobre el uso de las clases de API File en las fuentes siguientes:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Creación de un editor de archivos de texto \(en inglés\)](#)

Referencia del lenguaje

- [File](#)
- [FileStream](#)
- [FileMode](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR filesystem"](#)

Aspectos básicos de los archivos de AIR

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 <code>fileMode</code> de los métodos <code>open()</code> y <code>openAsync()</code> de la clase FileStream. El parámetro <code>fileMode</code> 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 anexo 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 manipular los datos en el archivo.

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 quedar en pausa procesos importantes como la animación o la representación de objetos de visualización.

Trabajo con objetos File

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, pero 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

La clase `File` sirve para:

- 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

Cada objeto `File` tiene dos propiedades que definen su ruta:

Propiedad	Descripción
<code>nativePath</code>	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 <code>nativePath</code> utiliza la barra diagonal inversa (\) como carácter separador de directorios, mientras que en Mac OS utiliza la barra diagonal (/).
<code>url</code>	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 <code>file</code> que se describen en "Esquemas de URL compatibles" en la página 107.

La clase `File` incluye propiedades para apuntar a directorios estándar tanto en Mac como en Windows.

Configuración de un objeto `File` para que apunte a un directorio

Existen distintas formas de configurar un objeto `File` para que apunte a un directorio.

Apuntar al directorio de inicio del usuario

Un objeto `File` puede apuntar al directorio de inicio del usuario. En Windows, el directorio de inicio es el directorio superior del directorio "Mis documentos" (por ejemplo, "C:\Documents and Settings*nombreUsuario*\Mis documentos"). En Mac OS es el directorio `Usuarios/nombreUsuario`. 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

Un objeto `File` puede apuntar al directorio de documentos del usuario. En Windows, éste suele ser el directorio "Mis documentos" (por ejemplo, "C:\Documents and Settings*nombreUsuario*\Mis documentos"). En Mac OS es el directorio `Usuarios/nombreUsuario/Documentos`. 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

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

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. Puede ser conveniente 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 basa en el nombre de usuario, el ID de la aplicación y el ID del editor:

- En Mac OS, en:

```
/Usuarios/nombre de usuario/Library/Preferences/applicationID.IDeditor/Local Store/
```

Por ejemplo:

```
/Users/babbage/Library/Preferences/com.example.TestApp.02D88EEED35F84C264A183921344EEA353A629FD.1/Local Store
```

- En Windows, en el directorio Documents and Settings en:

```
nombre de usuario/Application Data/applicationID.IDeditor/Local Store/
```

Por ejemplo:

```
C:\Documents and Settings\babbage\Application Data\com.example.TestApp.02D88EEED35F84C264A183921344EEA353A629FD.1\Local Store
```

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](#)” en la página 107), 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

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](#)” en la página 107), como en el ejemplo siguiente:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

Apuntar a la raíz del sistema de archivos

El método `File.getRootDirectories()` enumera todos los volúmenes raíz -como C: y los volúmenes montados- de un ordenador con Windows. En un Mac, este método siempre produce como resultado el directorio raíz exclusivo del ordenador (el directorio `/`).

Apuntar a un directorio explícito

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\";
```

Desplazarse a rutas relativas

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 107.

Dejar que el usuario utilice la función Examinar para seleccionar un directorio

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. 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);  
}
```

Apuntar al directorio desde el cual se invocó la aplicación.

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 272.

Configuración de un objeto File para que apunte a un archivo

Existen distintas formas de configurar el archivo al que apunta un objeto `File`.

Apuntar a una ruta de archivo explícita

El método `resolvePath()` sirve para obtener una ruta relativa a otra ruta determinada. Por ejemplo, el siguiente código configura un objeto `File` para que apunte a un archivo `log.txt` en el directorio de almacenamiento de la aplicación:

```
var file:File = File.applicationStorageDirectory;  
file = file.resolvePath("log.txt");
```

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");
```

En Windows se puede utilizar el carácter de barra diagonal (`/`) o barra diagonal inversa (`\`) como delimitador de ruta para la propiedad `nativePath`. En Mac OS se utiliza el carácter de barra diagonal (`/`) como delimitador para `nativePath`:

```
var file:File = new File(/Users/dijkstra/AIR Test/test.txt");
```

Para ver más información, consulte “[Modificación de rutas de archivos](#)” en la página 107.

Enumeración de los archivos de un directorio

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 112.

Dejar que el usuario utilice la función Examinar para seleccionar un archivo

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.

Modificación de rutas de archivos

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
```

Con la propiedad `nativePath`, en Windows se utiliza la barra diagonal (/) o la barra diagonal inversa (\) como carácter separador de directorios, mientras que en Mac OS se utiliza la barra diagonal (/). En Windows, recuerde escribir dos veces el carácter de barra diagonal inversa en un literal de cadena.

Esquemas de URL compatibles

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: <code>file:///c:/AIR Test/test.txt</code> La norma para URL especifica que la URL tipo "file" debe tener el formato <code>file://<host>/<ruta></code> . 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 <code>aplicacion.xml</code> para la aplicación instalada). Por ejemplo, la ruta siguiente apunta a un subdirectorio de imágenes del directorio de la aplicación instalada: <code>app:/images</code>
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: <code>app-storage:/settings/prefs.xml</code>

Búsqueda de la ruta relativa entre dos archivos

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

Para los nombres de archivo y de ruta no se suele distinguir entre mayúsculas y minúsculas. 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.

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

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 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.

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:File = 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:File = 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:File = 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

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 116.

Obtención de información sobre el sistema de archivos

La clase `File` incluye las siguientes propiedades fijas que proporcionan información útil sobre el sistema de archivos:

Propiedad	Descripción
<code>File.lineEnding</code>	La secuencia de caracteres de final de línea que utiliza el sistema operativo del ordenador host. En Mac OS es el carácter de salto de línea. En Windows es el carácter de retorno de carro seguido del carácter de salto de línea.
<code>File.separator</code>	El carácter separador de componentes de rutas en el sistema operativo del ordenador host. En Mac OS es el carácter de barra diagonal (/). En Windows es el carácter de barra diagonal inversa (\).
<code>File.systemCharset</code>	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
<code>Capabilities.hasIME</code>	Especifica si el reproductor se ejecuta en un sistema que tiene (<code>true</code>) o no tiene (<code>false</code>) un editor de método de entrada (IME) instalado.
<code>Capabilities.language</code>	Especifica el código de idioma del sistema en el que se está ejecutando el reproductor.
<code>Capabilities.os</code>	Especifica el sistema operativo actual.

Trabajo con directorios

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 103.

Creación de directorios

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 116.

Creación de directorios temporales

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 ser conveniente eliminar el directorio temporal antes de cerrar la aplicación, dado que *no* se elimina de forma automática.

Enumeración de directorios

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);
    }
}
```

Copiar y mover directorios

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 “[Copiar y mover archivos](#)” en la página 114.

Eliminación del contenido de los directorios

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 101). 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, sólo 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 115.

Trabajo con archivos

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 23 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 280.

Obtención de información sobre los archivos

La clase `File` incluye las siguientes propiedades que brindan información sobre un archivo o directorio al que apunta un objeto `File`:

Propiedad File	Descripción
<code>creationDate</code>	La fecha de creación del archivo en el disco local.
<code>creator</code>	Obsoleto: utilice la propiedad <code>extension</code> . (Esta propiedad notifica el tipo de creador Macintosh del archivo, que sólo se utiliza en las versiones de Mac OS anteriores a Mac OS X).
<code>exists</code>	Si existe o no el archivo o directorio al que se remite.
<code>extension</code>	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 <code>null</code> .
<code>icon</code>	Un objeto <code>Icon</code> que contiene los iconos definidos para el archivo.

Propiedad File	Descripción
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 102.
parent	La carpeta que contiene la carpeta o el archivo que representa el objeto File. Esta propiedad es <code>null</code> 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 <code>extension</code> . (En un ordenador Macintosh esta propiedad es el tipo de archivo de cuatro caracteres, que sólo 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 102.

Para obtener más información sobre estas propiedades, consulte la entrada sobre la clase File en [Referencia del lenguaje y componentes ActionScript 3.0](#) (http://www.adobe.com/go/learn_air_aslr_es).

Copiar y mover archivos

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 101).

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 definirlo en `true`, si el archivo de destino ya existe éste 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

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 101).

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 115.

Traslado de archivos a la papelera

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 101).

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();
```

Creación de archivos temporales

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 ser conveniente eliminar el archivo temporal antes de cerrar la aplicación, dado que *no* se elimina de forma automática.

Lectura y escritura de archivos

La clase `FileStream` permite a las aplicaciones de AIR leer y escribir en el sistema de archivos.

Flujo de trabajo de lectura y escritura de archivos

El flujo de trabajo para la lectura y escritura de archivos es el siguiente.

Inicialice un objeto `File` que apunte a la ruta.

Ésta es la ruta del archivo con 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 105.

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 118 y [“Modos de abrir de FileStream”](#) en la página 117.

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 diversas situaciones, por ejemplo cuando se leen datos provenientes del archivo, cuando se encuentran errores de E/S o cuando se ha terminado de escribir todos los datos.

Para obtener más información, consulte [“Programación asíncrona y eventos generados por un objeto FileStream abierto de forma asíncrona”](#) en la página 122.

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 a utilizar”](#) en la página 123.

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 120.

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 110.

Llame al método `close()` del objeto `FileStream` cuando haya terminado de trabajar con el archivo.

De esta forma el archivo queda a disposición de otras aplicaciones.

Para obtener más información, consulte [“Inicialización de un objeto FileStream, y apertura y cierre de archivos”](#) en la página 118.

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:

- [Creación de un editor de archivos de texto](#)

Trabajo con objetos `FileStream`

La clase `FileStream` define métodos para abrir, leer y escribir archivos.

Modos de abrir de `FileStream`

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
<code>FileMode.READ</code>	Especifica que el archivo está abierto sólo para fines de lectura.
<code>FileMode.WRITE</code>	Especifica que el archivo está abierto con posibilidad de escritura. Si el archivo no existe, se crea al abrirse el objeto <code>FileStream</code> . Si el archivo existe, se eliminan los datos existentes.
<code>FileMode.APPEND</code>	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.
<code>FileMode.UPDATE</code>	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, sólo los bytes que se escriben nuevamente sobrescriben los bytes existentes (todos los demás permanecen sin modificar).

Inicialización de un objeto `FileStream`, y apertura y cierre de archivos

Al abrir un objeto `FileStream`, éste 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, anexo o actualización. Para ver más información, consulte la sección anterior, “[Modos de abrir de `FileStream`](#)” en la página 117.

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.IOError, errorHandler);
myFileStream.open(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 101.

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 120.

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

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` sólo 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 anexo, la propiedad `position` no se modifica tras llamar a un método de escritura. (En el modo de anexo, 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`

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, sólo se pueden leer los datos que se encuentran en el búfer de lectura. Además, a medida que el usuario lee los datos, éstos 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

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 a utilizar

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). ActionScript admite códigos de juegos de caracteres comunes, que se enumeran en la referencia del lenguaje de ActionScript 3.0 (consulte [Juegos de caracteres admitidos](http://livedocs.macromedia.com/flex/2/langref/charset-codes.html) en <http://livedocs.macromedia.com/flex/2/langref/charset-codes.html>).

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. (Éste 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 124.

`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). Este formato es propio de `ActionScript`. Las únicas aplicaciones que cuentan con API integradas para trabajar con datos que tienen este formato son AIR, Flash Player, Flash Media Server y Flex Data Services.

Existen algunos otros métodos de lectura y escritura (como `readDouble()` y `writeDouble()`). Si se utiliza alguno de éstos, 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 sólo pueden interpretarse con algoritmos de descompresión y decodificación 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 125.

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.

Ejemplo: Lectura de un archivo XML para ponerlo en un objeto XML

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 a utilizar”](#) en la página 123).

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.

Ejemplo: Lectura y escritura de datos con acceso directo

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 leerlos (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 15: Arrastrar y colocar

Utilice las clases de la API de arrastrar y colocar para tener 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
- Objetos serializados
- Referencias de objetos (válidas solamente en la aplicación de origen)

Información suplementaria en línea sobre la función de arrastrar y colocar

Encontrará más información sobre cómo trabajar con la API de arrastrar y colocar en las fuentes siguientes:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Compatibilidad con las funciones de arrastrar y colocar, y copiar y pegar \(en inglés\)](#)

Referencia del lenguaje

- [NativeDragManager](#)
- [NativeDragOptions](#)
- [Clipboard](#)
- [NativeDragEvent](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR drag and drop"](#)

Aspectos básicos de arrastrar y colocar

La API de arrastrar y colocar contiene las clases siguientes.

Paquete	Clases
flash.desktop	<ul style="list-style-type: none"> • NativeDragManager • NativeDragOptions • Clipboard • NativeDragActions • ClipboardFormat • ClipboardTransferModes <p>Las constantes que se utilizan con la API de arrastrar y colocar se definen en las clases siguientes:</p> <ul style="list-style-type: none"> • 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`.

Arrastrar *El usuario mantiene pulsado el botón del ratón al desplazar el cursor a otro componente, a otra aplicación o al escritorio.* AIR puede mostrar una imagen proxy durante la operación. Siempre y cuando esté en curso la operación de arrastrar, el objeto iniciador distribuye eventos `nativeDragUpdate`. 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 componente 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, sólo 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 sólo son válidas en la aplicación de origen. Las transferencias de objetos serializados son válidos entre aplicaciones de AIR, pero sólo 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.doDrag()`). 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

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

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, consulte [“Formatos de datos para Clipboard”](#) en la página 149.

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

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 <code>dragStart</code> y <code>dragComplete</code> . El iniciador debe ser un objeto interactivo.
clipboard	El objeto <code>Clipboard</code> que contiene los datos a transferirse. Hay referencias al objeto <code>Clipboard</code> en los objetos <code>NativeDragEvent</code> distribuidos durante la secuencia de arrastrar y colocar.
dragImage	(Opcional) Un objeto <code>BitmapData</code> para mostrar durante la operación de arrastrar. La imagen puede especificar un valor <code>alpha</code> . (Nota: Microsoft Windows aplica siempre un difuminado alfa fijo a las imágenes arrastradas).
offset	(Opcional) Un objeto <code>Point</code> 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 <code>NativeDragOptions</code> 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 <code>DragOptions</code> en los objetos <code>NativeDragEvent</code> 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

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

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

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

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 sólo 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

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

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.
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 interactivo, 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

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.

Arrastrar y colocar en HTML

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

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, sólo se puede arrastrar texto para introducirlo en un elemento y sólo 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-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 ver más información, consulte “[Extensiones de CSS](#)” en la página 215.

Eventos de arrastrar y colocar en HTML

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 <code>preventDefault()</code> del objeto de evento. Para controlar si se pueden copiar, vincular o mover los datos arrastrados, configure la propiedad <code>effectAllowed</code> . 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 <code>dataTransfer</code> 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 <code>dataTransfer.dropEffect</code> 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 <code>dataTransfer</code> en un controlador de eventos <code>dragenter</code> , esas modificaciones se verán suprimidas en cuanto se produzca el siguiente evento <code>dragover</code> . Por otro lado, existe un breve retardo entre un evento <code>dragenter</code> y el primer evento <code>dragover</code> 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. Sólo 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
<code>effectAllowed</code>	El efecto admitido por el origen de la operación de arrastrar. Lo habitual es que este valor lo defina el controlador del evento <code>dragstart</code> . Consulte "Efectos de arrastrar en HTML" en la página 137.
<code>dropEffect</code>	El efecto elegido por el destino o el usuario. Si se define la propiedad <code>dropEffect</code> en un controlador de eventos <code>dragover</code> o <code>dragenter</code> , AIR actualiza el cursor para mostrar el efecto que se produciría si el usuario soltara el ratón. Si la definición de <code>dropEffect</code> no coincide con uno de los efectos admitidos, no se permite colocar y se presenta el cursor de <i>indisposición</i> . Si usted no ha definido la propiedad <code>dropEffect</code> como respuesta al último evento <code>dragover</code> o <code>dragenter</code> , el usuario podrá elegir entre los efectos admitidos con las teclas modificadoras normales del sistema operativo. El efecto final lo notifica la propiedad <code>dropEffect</code> del objeto distribuido para <code>dragend</code> . Si el usuario abandona la operación de colocar, soltando el ratón fuera de un destino idóneo, <code>dropEffect</code> se define en <code>none</code> .
<code>types</code>	Un conjunto que contiene las cadenas de tipos MIME para cada formato de datos presente en el objeto <code>dataTransfer</code> .
<code>getData(mimeType)</code>	Obtiene los datos en el formato especificado por el parámetro <code>mimeType</code> . El método <code>getData()</code> sólo puede llamarse como respuesta al evento <code>drop</code> .
<code>setData(mimeType)</code>	Añade datos a <code>dataTransfer</code> en el formato especificado por el parámetro <code>mimeType</code> . Para añadir datos en varios formatos, llame a <code>setData()</code> para cada tipo MIME. Los datos que el comportamiento predeterminado de arrastrar haya colocado en el objeto <code>dataTransfer</code> se eliminan. El método <code>setData()</code> sólo puede llamarse como respuesta al evento <code>dragstart</code> .
<code>clearData(mimeType)</code>	Elimina datos en el formato especificado por el parámetro <code>mimeType</code> .
<code>setDragImage(image, offsetX, offsetY)</code>	Configura una imagen de arrastrar personalizada. El método <code>setDragImage()</code> sólo puede llamarse como respuesta al evento <code>dragstart</code> .

Tipos MIME para la operación de arrastrar y colocar en HTML

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: Sólo 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 141.

Efectos de arrastrar en HTML

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. <i>All</i> 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

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

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 hacia dentro de un elemento en HTML

El comportamiento predeterminado sólo 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

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 de 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

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 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

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></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>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</td></tr>
    </table>
  </div>
</div>
</body>
</html>

```

Gestión de la colocación de archivos en entornos limitados de HTML ajenos a la aplicación

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 un 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, impidiendo que ocurra el método `preventDefault()` del objeto `Event` en los controladores de eventos en HTML `dragenter` y `dragover`, o el evento `drop` no podrá nunca suceder.

```
<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>
```

Para ver más información, consulte “[Programación en HTML y JavaScript](#)” en la página 217.

Capítulo 16: Copiar y pegar

Utilice las clases de la API del portapapeles para copiar información al portapapeles del sistema y pegarla en otro lado. Los formatos de datos que se pueden transferir a o de una aplicación de Adobe® AIR™ incluyen:

- Mapas de bits
- Archivos
- Texto
- Texto en formato HTML
- Datos en formato de texto enriquecido
- Cadenas de URL
- Objetos serializados
- Referencias de objetos (válidas solamente en la aplicación de origen)

Información suplementaria en línea sobre la función de copiar y pegar

Encontrará más información sobre la función de copiar y pegar en las fuentes siguientes:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

- [Compatibilidad con las funciones de arrastrar y colocar, y copiar y pegar \(en inglés\)](#)

Referencia del lenguaje

- [Clipboard](#)
- [ClipboardFormats](#)
- [ClipboardTransferMode](#)

Más información

- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR copy and paste"](#)

Aspectos básicos de copiar y pegar

La API de copiar y pegar contiene las clases siguientes.

Paquete	Clases
flash.desktop	<ul style="list-style-type: none"> • Clipboard • ClipboardFormats • ClipboardTransferMode <p>Las constantes que se utilizan con la API de copiar y pegar se definen en las clases siguientes:</p> <ul style="list-style-type: none"> • 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. También se pueden crear nuevos objetos `Clipboard` para transferir datos a través de la API de arrastrar y colocar.

El entorno HTML proporciona otra API para copiar y pegar. El código que se ejecuta en el entorno limitado de la aplicación puede utilizar cualquiera de las dos API, pero en contenido ajeno a la aplicación sólo se puede utilizar la API de HTML. (Consulte “[Copiar y pegar en HTML](#)” en la página 145).

Las clases `HTMLLoader` y `TextField` implementan el comportamiento predeterminado para los métodos abreviados de teclado normales para 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 tecla. También se pueden utilizar comandos de menú nativos junto con equivalentes a las teclas para responder a las pulsaciones de tecla 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.

Nota: no hay garantía de que los datos escritos en el portapapeles permanezcan accesibles después de salir la aplicación de AIR.

Lectura y escritura en el portapapeles del sistema

Para leer el portapapeles del sistema operativo, llame al método `getData()` del objeto `Clipboard.generalClipboard`, pasando el nombre del formato que se necesita 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);
}
```

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: sólo el código que se ejecute en el entorno limitado de la aplicación tiene acceso directo al portapapeles del sistema. En el contenido HTML ajeno a la aplicación, sólo se tiene acceso al portapapeles a través de la propiedad `clipboardData` de un objeto de evento distribuido por uno de los eventos de copiar o pegar en HTML.

Copiar y pegar en HTML

El entorno HTML proporciona su propia serie de eventos y comportamiento predeterminado para copiar y pegar. Sólo 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

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.

Utilización de la propiedad `clipboardData` del objeto de evento

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");
}
```

Sólo 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>
```

Comandos de menú y pulsaciones de tecla para copiar y pegar

El funcionamiento de copiar y pegar normalmente se activa mediante comandos de menú y métodos abreviados de teclado. En OS X, el sistema operativo crea automáticamente un menú de edición con los comandos de copiar y pegar, pero deberá añadir detectores a estos comandos de menú para habilitar sus propias funciones de copiar y pegar. En Windows se puede añadir un menú de edición nativo a cualquier ventana que utilice el fondo cromático del sistema. (También se pueden crear menús no nativos con Flex y ActionScript o, en contenido HTML, se puede utilizar DHTML, pero esta discusión no abarca ese tema).

El funcionamiento de copiar y pegar normalmente se activa mediante comandos de menú y métodos abreviados de teclado. En OS X, el sistema operativo crea automáticamente un menú de edición con los comandos de copiar y pegar, pero deberá añadir detectores a estos comandos de menú para habilitar sus propias funciones de copiar y pegar. En Windows se puede añadir un menú de edición nativo a cualquier ventana que utilice el fondo cromático del sistema. (También se pueden crear menús no nativos con ActionScript o, en contenido HTML, se puede utilizar DHTML, pero esta discusión no abarca ese tema).

Para activar comandos de copiar y pegar como respuesta a los métodos abreviados de teclado, se pueden asignar equivalentes de las teclas a los elementos de comando apropiados en un menú nativo de la aplicación o de una ventana, o se pueden detectar directamente las pulsaciones de tecla.

Inicio de una operación de copiar o pegar con un comando de menú

Para activar una operación de copiar o pegar con un comando de menú hay que añadir detectores para el evento `select` en los elementos de menú que llaman a las funciones del controlador.

Cuando se llama a la función del controlador, se puede localizar el objeto del que se va a copiar o en el que se va a pegar mediante uso de la propiedad `focus` del escenario. Seguidamente se puede llamar al método apropiado del objeto seleccionado (o un método de opción general, si no hay ningún objeto seleccionado) para llevar a cabo la lógica de copia, corte o pegado. Por ejemplo, el siguiente controlador de eventos `copy` comprueba si el objeto seleccionado es del tipo correcto, en este caso, una clase denominada *Scrap*, y después llama al método `doCopy()` del objeto.

```
function copyCommand(event:Event):void{
    if(NativeApplication.nativeApplication.activeWindow.stage.focus is Scrap){
        Scrap(NativeApplication.nativeApplication.activeWindow.stage.focus).doCopy();
    } else {
        NativeApplication.nativeApplication.copy();
    }
}
```

Si `copyCommand()` en el ejemplo no reconoce la clase de objeto seleccionado, llama al método `copy()`. El método `copy()` de `NativeApplication` envía un comando interno de copia al objeto seleccionado. Al comando interno sólo lo reconocen los objetos `TextArea` y `HTMLLoader`. Existen comandos similares para cortar, pegar, seleccionar todo y para `TextArea` solamente, borrar, deshacer y rehacer.

Nota: no hay ninguna API que responda a estos comandos internos en un componente personalizado. Se deberá ampliar las clases `TextArea` o `HTMLLoader`, o bien incluir uno de estos objetos en el componente personalizado. Si se incluye `TextArea` o `HTMLLoader`, el componente debe manejar la selección de modo que el objeto `TextArea` o `HTMLLoader` conserve siempre la selección cuando el componente mismo está seleccionado.

Con contenido HTML, se puede activar el comportamiento predeterminado de copiar y pegar mediante comandos de edición de `NativeApplication`. En el siguiente ejemplo se crea un menú de edición para un documento HTML editable:

El ejemplo anterior sustituye el menú de la aplicación en Mac OS X pero también se puede utilizar el menú Editar predeterminado, localizando los elementos existentes y añadiéndoles detectores de eventos.

Si se utiliza un menú contextual para invocar un comando de copiar o pegar, se puede utilizar la propiedad `contextMenuOwner` del objeto `ContextMenuEvent` que se distribuye al abrir el menú o seleccionar un elemento para determinar qué objeto es el destino correcto del comando de copiar o pegar.

Localización de elementos de menú predeterminados en Mac OS X

Para localizar el menú de edición predeterminado y los elementos de comando específicos de copiar, cortar y pegar en el menú de la aplicación en Mac OS X, busque en la jerarquía de menús usando la propiedad `label` de los objetos `NativeMenuItem`. Por ejemplo, la siguiente función toma un nombre y localiza el elemento con la etiqueta correspondiente en el menú:

```
private function findItemByName(menu:NativeMenu,
                                name:String,
                                recurse:Boolean = false):NativeMenuItem{
    var searchItem:NativeMenuItem = null;
    for each (var item:NativeMenuItem in menu.items){
        if(item.label == name){
            searchItem = item;
            break;
        }
        if((item.submenu != null) && recurse){
            searchItem = findItemByName(item.submenu, name, recurse);
        }
        if(searchItem != null){ break; }
    }
    return searchItem;
}
```

Se puede definir el parámetro `recurse` en `true` para incluir submenús en la búsqueda, o en `false` para que sólo se incluya el menú que se pasa.

Inicio de un comando de copiar o pegar con una pulsación de teclas

Si la aplicación utiliza menús nativos de ventana o de la aplicación para copiar y pegar, se pueden añadir a los elementos de menú las teclas equivalentes para implementar métodos abreviados de teclado. En caso contrario, puede detectar uno mismo las pulsaciones de tecla pertinentes, como se demuestra en el ejemplo siguiente:

```
private function init():void{
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyListener);
}
private function keyListener(event:KeyboardEvent):void{
    if(event.ctrlKey){
        event.preventDefault();
        switch(String.fromCharCode(event.charCode)){
            case "c":
                NativeApplication.nativeApplication.copy();
                break;
            case "x":
                NativeApplication.nativeApplication.cut();
                break;
            case "v":
                NativeApplication.nativeApplication.paste();
                break;
            case "a":
                NativeApplication.nativeApplication.selectAll();
                break;
            case "z":
                NativeApplication.nativeApplication.undo();
                break;
            case "y":
                NativeApplication.nativeApplication.redo();
                break;
        }
    }
}
```

Con contenido HTML, los métodos abreviados de teclado para los comandos de copiar y pegar se implementan de forma predeterminada. No es posible capturar todas las pulsaciones de tecla que suelen emplearse para copiar y pegar utilizando un detector de eventos de teclas. Si necesita suprimir el comportamiento predeterminado, como estrategia es mejor detectar los propios eventos `copy` y `paste`.

Formatos de datos para Clipboard

Los formatos para Clipboard describen los datos que se colocan en un objeto Clipboard. AIR traduce automáticamente los formatos de datos estándar entre tipos de datos ActionScript y formatos del portapapeles del sistema. Por otra parte, los objetos de una aplicación pueden transferirse dentro de y entre aplicaciones de AIR utilizando formatos definidos por las aplicaciones.

Un objeto Clipboard puede contener representaciones de la misma información en distintos formatos. Por ejemplo, un objeto Clipboard que representa un elemento Sprite podría incluir un formato de referencia para uso en la misma aplicación, un formato serializado para uso en otra aplicación de AIR, un formato de mapa de bits para uso con un editor de imágenes, y un formato de lista de archivos, quizá con representación aplazada para codificar un archivo PNG, para copiar o arrastrar una representación del elemento Sprite al sistema de archivos.

Formatos de datos estándar

Las constantes que definen los nombres de formatos estándar se 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	Los datos en formato de mapa de bits se traducen a y de la clase BitmapData de ActionScript.
FILE_LIST_FORMAT	Los datos en formato de lista de archivos se traducen a y de un conjunto de objetos File de ActionScript.
URL_FORMAT	Los datos en formato URL se traducen a y de la clase String de ActionScript.

Nota: en la actualidad no existen componentes o controles de texto de Flash que admitan datos en formato de texto enriquecido. Para pegar datos en RTF en un control, traduzca primero el marcado RTF en marcado HTML (o texto sin formato). Asimismo, para copiar datos en RTF en el portapapeles, convierta una cadena de texto o HTML en objeto ByteArray con el marcado RTF correspondiente. Flash no tiene una clase de utilidad para convertir datos en RTF en otro formato.

Al copiar y pegar datos como respuesta a un evento `copy`, `cut`, o `paste` en contenido HTML hay que 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

Se pueden utilizar formatos personalizados definidos por la aplicación para transferir objetos como referencias o como copias serializadas. Las referencias sólo son válidas en la misma aplicación de AIR. Los objetos serializados se pueden transferir entre aplicaciones de Adobe AIR, pero sólo 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, defina 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

Cuando se escribe un objeto en el portapapeles empleando un formato de datos personalizado, los datos del objeto pueden leerse desde el portapapeles como referencia o como copia serializada del objeto original. AIR define cuatro modos de transferencia que determinan si los objetos se transfieren como referencias o como copias serializadas:

Modo de transferencia	Descripción
ClipboardTransferModes.ORIGINAL_ONLY	Sólo se devuelve una referencia. Si no se dispone de ninguna referencia, se devuelve un valor nulo.
ClipboardTransferModes.ORIGINAL_PREFERRED	Se devuelve una referencia, si hay una disponible. En caso contrario, se devuelve una copia serializada.
ClipboardTransferModes.CLONE_ONLY	Sólo se devuelve una copia serializada. Si no se dispone de ninguna copia serializada, se devuelve un valor nulo.
ClipboardTransferModes.CLONE_PREFERRED	Se devuelve una copia serializada, si hay una disponible. En caso contrario, se devuelve una referencia.

Lectura y escritura de formatos de datos personalizados

Al escribir un objeto en el portapapeles se puede utilizar cualquier cadena que no empiece con el prefijo reservado `air:` para el parámetro de formato. 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 = new Clipboard();
    transfer.setData("object", object, true);
}
```

Para extraer un objeto serializado del objeto portapapeles (tras una operación de colocar o pegar), utilice el mismo nombre de formato y el modo de transferencia `cloneOnly` o `clonePreferred`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

Siempre se añade una referencia al objeto Clipboard. Para extraer la referencia del objeto portapapeles (tras una operación de colocar o pegar), en lugar de la copia serializada utilice el modo de transferencia `originalOnly` o `originalPreferred`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Las referencias sólo son válidas si el objeto Clipboard viene de la aplicación de AIR actual. Utilice el modo de transferencia `originalPreferred` para tener acceso a la referencia cuando está disponible, y la copia serializada cuando la referencia no está disponible.

Representación aplazada

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. Sólo 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, al utilizar los formatos del portapapeles de AIR estándar no se produce la representación aplazada. La función de representación se llama inmediatamente.

Pegar texto con una función de representación aplazada

El ejemplo siguiente ilustra cómo se implementa una función de representación aplazada.

Al pulsar el botón Copiar en el ejemplo, la aplicación vacía el portapapeles del sistema para asegurarse de que no queden datos de otras operaciones del portapapeles y después pone la función `renderData()` en el portapapeles utilizando el método `setDataHandler()`.

Al pulsar el botón Pegar, la aplicación accede al clipboard y configura el texto de destino. Dado que el formato de datos de texto en el portapapeles se ha definido con una función en lugar de 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).

```
package
{
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events.MouseEvent;

    public class DeferredRenderingExample extends Sprite
    {
        var sourceTxt:TextField;
        var destinationTxt:TextField;

        public function DeferredRenderingExample():void
        {
            sourceTxt = createTextField(10, 10, 210, 380, false);
            addChild(sourceTxt);
            sourceTxt.text = "Neque porro quisquam est qui dolorem "
                + "ipsum quia dolor sit amet, consectetur, adipisci velit."

            destinationTxt = createTextField(330, 10, 210, 380, false);
            addChild(destinationTxt);

            var copyBtn:TextField = createTextField(230, 50, 90, 20, true);
            copyBtn.text = "Copy";
            addChild(copyBtn);
            copyBtn.addEventListener(MouseEvent.CLICK, onCopy);

            var pasteBtn:TextField = createTextField(230, 80, 90, 20, true);
            pasteBtn.text = "Paste";
            addChild(pasteBtn);
            pasteBtn.addEventListener(MouseEvent.CLICK, onPaste);
        }

        private function createTextField(x:Number, y:Number,
            width:Number, height:Number, isBtn:Boolean = false):TextField
        {
            var newTxt:TextField = new TextField();
            newTxt.x = x;
```

```
newTxt.y = y;
newTxt.height = height;
newTxt.width = width;
newTxt.border = true;
newTxt.background = true;

if (isBtn)
{
    newTxt.backgroundColor = 0xDDDDDDDEE;
    newTxt.selectable = false;
}
else
{
    newTxt.multiline = true;
    newTxt.wordWrap = true;
    newTxt.backgroundColor = 0xEEEEEEEEE;
}
return newTxt;
}
public function onCopy(event:MouseEvent):void
{
    Clipboard.generalClipboard.clear();
    Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
        renderData);
}
public function onPaste(event:MouseEvent):void
{
    destinationTxt.text =
        Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT) as String;
}
public function renderData():String
{
    trace("Rendering data");
    var sourceStr:String = sourceTxt.text;
    if (sourceTxt.selectionEndIndex > sourceTxt.selectionBeginIndex)
    {
        // something is selected
        return sourceStr.substring(sourceTxt.selectionBeginIndex,
            sourceTxt.selectionEndIndex);
    }
    else
    {
        return sourceStr;
    }
}
}
```


Capítulo 17: Trabajo con conjuntos de bytes

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.

Lectura y escritura de un ByteArray

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 de ByteArray

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

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

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, consectetur 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

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 algunos 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()`

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. AIR es compatible con los algoritmos `deflate` y `zlib`. El algoritmo de compresión `deflate` se utiliza en varios formatos de compresión, como `zlib`, `gzip` y algunas implementaciones de `zip`. El formato de datos comprimidos `zlib` se describe en <http://www.ietf.org/rfc/rfc1950.txt> y 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 `deflate`:

```
bytes.compress(CompressionAlgorithm.DEFLATE);
```

En el siguiente ejemplo se descomprime un ByteArray comprimido con el algoritmo deflate:

```
bytes.uncompress(CompressionAlgorithm.DEFLATE);
```

Lectura y escritura de objetos

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.objectEncoding` 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`.

```
import flash.filesystem.*;  
import flash.utils.ByteArray;  
  
// Label component must be in Library  
import fl.controls.Label;
```

```
var bytes:ByteArray = new ByteArray();
var myLabel:Label = new Label();
myLabel.move(150, 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
outFile("order", bytes);
myLabel.text = "Wrote order file to desktop!";

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 outputStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outputStream.open(outFile, FileMode.WRITE);
    // write out the file
    outputStream.writeBytes(data, 0, data.length);
    // close it
    outputStream.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.utils.ByteArray;

// TextArea component must be in Library
import fl.controls.TextArea;
```

```
var inBytes:ByteArray = new ByteArray();
// define text area for displaying XML content
var myTxt:TextArea = new TextArea();
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";
readFile("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
function readFile(fileName:String, data:ByteArray) {
    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, 0, data.length);
    inStream.close();
}
```

Ejemplo de ByteArray: Lectura de un archivo .zip

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 sólo 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 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

```
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();
```

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
fileNameLength = bytes.readShort(); // store file name
offset += fileNameLength; // add length of file name
bytes.position = 28; // offset to extra field length
extraFieldLength = bytes.readShort();
offset += extraFieldLength; // 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).

```
bytes.position = 30;
fileName = bytes.readUTFBytes(fileNameLength); // 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');
```

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
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
```

La llave final indica el final del bucle `while` y del código de la aplicación, con la excepción del 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
```

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.

```
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();
}
```

Capítulo 18: Trabajo con bases de datos SQL locales

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 compatibles 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.

Información adicional en línea sobre bases de datos SQL locales

Puede encontrar más información sobre el uso de bases de datos SQL locales en los siguientes recursos:

Guías de inicio rápido (Centro de desarrollo de Adobe AIR)

Referencia del lenguaje

- [SQLCollationType](#)
- [SQLColumnNameStyle](#)
- [SQLColumnSchema](#)
- [SQLConnection](#)
- [SQLError](#)
- [SQLErrorEvent](#)
- [SQLErrorOperation](#)
- [SQLEvent](#)
- [SQLIndexSchema](#)
- [SQLMode](#)
- [SQLResult](#)
- [SQLSchema](#)
- [SQLSchemaResult](#)
- [SQLStatement](#)
- [SQLTableSchema](#)
- [SQLTransactionLockType](#)
- [SQLTriggerSchema](#)
- [SQLUpdateEvent](#)
- [SQLViewSchema](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash \(en inglés\); busque "AIR SQL"](#)

Bases de datos SQL locales

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 de 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

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 cualquier aplicación de AIR puede abrir el archivo de base de datos, por lo que se recomienda el uso de un mecanismo de cifrado individual para los documentos confidenciales).
- 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.

Bases de datos y archivos de base de datos de AIR

Una base de datos SQL local de Adobe AIR individual se almacena como un sólo 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

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, ésta 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

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 el apéndice “[Compatibilidad SQL en bases de datos locales](#)” en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

Clases de bases de datos SQL

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
<code>flash.data.SQLConnection</code>	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.
<code>flash.data.SQLStatement</code>	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.
<code>flash.data.ResultSet</code>	Proporciona una manera de obtener información o resultados de la ejecución de una declaración, como las filas resultantes de una declaración <code>SELECT</code> , el número de filas afectadas por una declaración <code>UPDATE</code> o <code>DELETE</code> 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
<code>flash.data.SQLSchemaResult</code>	Actúa como un contenedor para los resultados de esquema de la base de datos generados al llamar al método <code>SQLConnection.loadSchema()</code> .
<code>flash.data.SQLTableSchema</code>	Proporciona información que describe un sola tabla en una base de datos.
<code>flash.data.SQLViewSchema</code>	Proporciona información que describe un sola vista en una base de datos.
<code>flash.data.SQLIndexSchema</code>	Proporciona información que describe una sola columna de una tabla o vista en una base de datos.
<code>flash.data.SQLTriggerSchema</code>	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
<code>flash.data.SQLMode</code>	Define un conjunto de constantes que representan los valores posibles del parámetro <code>openMode</code> de los métodos <code>SQLConnection.open()</code> y <code>SQLConnection.openAsync()</code> .
<code>flash.data.SQLColumnNameStyle</code>	Define un conjunto de constantes que representan los valores posibles de la propiedad <code>SQLConnection.columnNameStyle</code> .
<code>flash.data.SQLTransactionLockType</code>	Define un conjunto de constantes que representan los valores posibles del parámetro de opción del método <code>SQLConnection.begin()</code> .
<code>flash.data.SQLCollationType</code>	Define un conjunto de constantes que representan los valores posibles de la propiedad <code>SQLColumnSchema.defaultCollationType</code> y el parámetro <code>defaultCollationType</code> del constructor <code>SQLColumnSchema()</code> .

Además, las siguientes clases en el paquete `flash.events` representan los eventos (y constantes admitidas) que usa:

Clase	Descripción
flash.data.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.data.SQLErrorEvent	Define el evento que una instancia SQLConnection o SQLStatement distribuye cuando cualquiera de sus operaciones resulta en error.
flash.data.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.data.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.data.SQLErrorEvent	Define un conjunto de constantes que representa los valores posibles para la propiedad <code>operation</code> de la clase SQLError, que indica la operación de la base de datos que resultó en error.

Modos de ejecución sincrónicos y asíncronos

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 éste 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 múltiples 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 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 declaraciones 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 al 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 “[Utilización de operaciones sincrónicas y asíncronas de base de datos](#)” en la página 190.

Creación y modificación de una base de datos

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.

Creación de una base de datos

Para crear un archivo de base de datos, primero debe crear una instancia `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 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 directorio de almacenamiento de la aplicación 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);
var dbFile:File = File.applicationStorageDirectory.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);
}
```

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.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;
var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.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);
}
```

Creación de tablas en la base de datos

La creación de una tabla en una base de datos implica ejecutar una declaración SQL en dicha base de datos, usando el mismo proceso que usa 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 173.

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 este código supone que existe una instancia `SQLConnection` denominada `conn` que ya ha sido creada y ya 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);
}
```

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 este código supone que existe una instancia `SQLConnection` denominada `conn` que ya ha sido creada y ya 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;
try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

Manipulación de los datos de una base de datos SQL

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 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 169.

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. Los ejemplos en esta sección demuestran cómo realizar las operaciones de forma asíncrona como sincrónicamente. Para más información sobre el uso del modo de ejecución sincrónico o asíncronos, consulte [“Utilización de operaciones sincrónicas y asíncronas de base de datos”](#) en la página 190.

Conexión a una base de datos

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 `SQLConnection` para representar una conexión a una o más 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 directorio de almacenamiento de la aplicación del usuario.

```
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");
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);
}
```

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 directorio de almacenamiento de la aplicación del usuario.

```
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();
var dbFile:File = File.applicationStorageDirectory.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);
}
```

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. Asimismo puede especificar `SQLMode.READ` para el parámetro `openMode` para abrir una base de datos existente en el modo de sólo 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

Una declaración SQL individual (una consulta o comando) está representada en el motor de ejecución como un 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 (sólo 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 177



“[Inserción de datos](#)” en la página 183



“[Cambio o eliminación de datos](#)” en la página 186

Utilización de parámetros en declaraciones

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. Esto proporciona ventajas de rendimiento así como facilita la codificación de una aplicación.

Aspectos básicos de parámetros de declaración

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 reemplaza con un valor real cada vez que se ejecuta la declaración. Para utilizar parámetros en una declaración SQL, se crea una instancia `SQLStatement` como siempre. 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.

Utilización de parámetros con nombre

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();
```

Utilización de parámetros sin nombre

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

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 más información sobre el almacenamiento de clases y afinidad de columnas, consulte la sección "[Compatibilidad de tipos de datos](#)" en el apéndice "[Compatibilidad SQL en bases de datos locales](#)" en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

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

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`

Para recuperar datos existentes de una base de datos, se usa una instancia `SQLStatement`. Asigne la declaración `SELECT` a la propiedad `text` de la instancia y llame al método `execute`.

Para obtener información sobre la sintaxis de la declaración `SELECT`, consulte el apéndice "Compatibilidad con SQL en bases de datos locales" en [Referencia del lenguaje y componentes ActionScript 3.0](#).

En el siguiente ejemplo se demuestra 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";
// The resultHandler and errorHandler are listener methods are
// described in a subsequent code listing
selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
selectStmt.execute();
```

En el siguiente ejemplo se demuestra la ejecución de una declaración `SELECT` para recuperar datos de la tabla denominada "products" utilizando el modo de ejecución sincrónico:


```
var selectStmt:SQLStatement = new SQLStatement();
// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;
selectStmt.text = "SELECT itemId, itemName, price FROM products";
// This try..catch block is fleshed out in
// a subsequent code listing
try
{
    selectStmt.execute();
    // accessing the data is shown in a subsequent code listing
}
catch (error:SQLError)
{
    // error handling is shown in a subsequent code listing
}
```

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 pasa un objeto `Responder` como un argumento en la llamada de `execute()`, se llama a la función de control de resultados del objeto `Responder`. En el modo de ejecución síncrono, 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

Una vez que la declaración `SELECT` se ha terminado de ejecutar, el siguiente paso es acceder a los datos que se recuperaron. Cada fila de datos en el conjunto de resultados `SELECT` se vuelve una instancia `Object`. 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 continúa al ejemplo de código anterior para recuperar datos en el modo de ejecución asíncrono. Se muestra la manera de acceder a los datos recuperados en el método de detector de evento de resultado.

```
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.
}
```

El siguiente ejemplo de código amplía el ejemplo de código anterior para recuperar datos en el modo de ejecución sincrónico. Se amplía el bloque `try..catch` en el ejemplo anterior de ejecución sincrónica, mostrando la manera de acceder a los datos recuperados.

```
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.
}
```

Como muestran los ejemplos anteriores de código, los objetos de resultado se encuentran en un conjunto que está disponible como la propiedad `data` de una instancia `SQLResult`. Si está utilizando la ejecución asíncrona con un detector de evento, para recuperar dicha instancia `SQLResult` se llama al método `getResult()` de la instancia `SQLStatement`. Si especifica un argumento `Responder` en la llamada `execute()`, la instancia `SQLResult` se pasa a la función de control de resultados como un argumento. En el modo de ejecución sincrónico, llama al método `getResult()` de la instancia `SQLStatement` en cualquier momento después de la llamada del método `execute()`. En cualquier caso, una vez que tiene el objeto `SQLResult` puede acceder a las filas de resultados usando la propiedad del conjunto `data`.

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 `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.ResultSet;
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:ResultSet = 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);
}
```

El siguiente ejemplo de código demuestra las mismas técnicas como la anterior pero utiliza el modo de ejecución sincrónico. El ejemplo define una instancia `SQLStatement` cuyo texto es una declaración `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 `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.ResultSet;
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;
try
{
    // execute the statement
    selectStmt.execute();
    // access the result data
    var result:ResultSet = 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);
}
```

Definición de tipos de datos de los datos del resultado SELECT

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

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 sólo 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 llama al método `ejecutar()` de la instancia `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 `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 más información, consulte las descripciones en la referencia del lenguaje para el método `SQLStatement.execute()` (la descripción del parámetro `prefetch`) y el método `SQLStatement.next()`.

Inserción de datos

La recuperación de datos de una base de datos implica ejecutar una declaración `INSERT SQL`. 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`

Para añadir datos a una tabla en una base de datos, se crea y ejecuta una instancia `SQLStatement` cuyo texto es una declaración `INSERT SQL`.

En el siguiente ejemplo se utiliza una instancia `SQLStatement` para añadir una fila de datos a la tabla de empleados 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 ha sido creada y ya está conectada a la base de datos. Asimismo supone que la tabla “employees” ya ha sido creada.

```
import flash.data.SQLConnection;
import flash.data.ResultSet;
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);
}
```

En el siguiente ejemplo se añade una fila de datos a la tabla de empleados existente, usando el modo de ejecución sincrónico. Observe que este código supone que existe una instancia `SQLConnection` denominada `conn` que ya ha sido creada y ya está conectada a la base de datos. Asimismo supone que la tabla “employees” ya ha sido creada.

```

import flash.data.SQLConnection;
import flash.data.ResultSet;
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;
try
{
    // execute the statement
    insertStmt.execute();
    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}

```

Recuperación de la clave principal generada por la base de datos de una fila insertada

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 `ResultSet` generado por 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 `ResultSet`. Se accede a la instancia `ResultSet` llamando al método `getResult()` del objeto `SQLStatement` si se usa un detector de evento o si usa el modo de ejecución síncronico. Como alternativa, si usa el modo de ejecución asíncrono y pasa una instancia `Responder` a la llamada `execute()`, la instancia `ResultSet` se pasa como un argumento a la función de control de resultados. En cualquier caso, la instancia `ResultSet` tiene una 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();
private function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:ResultSet = 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 ...";
insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);
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 la siguiente regla:

- 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 para la 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`. En 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`. Observe que si la tabla no incluye una columna explícita `INTEGER PRIMARY KEY`, es 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 más información sobre las claves principales y los identificadores de fila generados, consulte la sección “[CREATE TABLE](#)” y “[Expresiones](#)” en el apéndice “[Compatibilidad con SQL en bases de datos locales](#)” en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

Cambio o eliminación de datos

El proceso para ejecutar otras operaciones de manipulación de datos es idéntico al proceso utilizado para ejecutar una declaración `SELECT` o `INSERT SQL`. 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`.
- Par eliminar una o más filas de datos de una tabla, use una declaración `DELETE`.

Para obtener descripciones de estas declaraciones, consulte el apéndice “[Compatibilidad con SQL en bases de datos locales](#)” en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

Trabajo con múltiples bases de datos

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

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

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 sólo archivo de base de datos para todos los usuarios que usan la aplicación en un sólo 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 sólo 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 sólo 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

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

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 189 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 cabo 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 explícito.

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 reemplaza) 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 más información consulte la sección “[ON CONFLICT \(algoritmos de conflicto\)](#)” en el apéndice “[Compatibilidad con SQL en bases de datos locales](#)” en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

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

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, sólo 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 más información sobre los tipos de afinidad de columnas disponibles y el uso de tipos de datos en declaraciones SQL, consulte la sección "Compatibilidad de tipos de datos" en el apéndice "Compatibilidad con SQL en bases de datos locales" en [Referencia de Componentes y Lenguaje ActionScript 3.0](#).

Utilización de operaciones sincrónicas y asíncronas de base de datos

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 `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.

Utilización de operaciones sincrónicas de base de datos

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

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();
var dbFile:File = File.applicationStorageDirectory.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

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();
var dbFile:File = File.applicationStorageDirectory.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

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();
var dbFile:File = File.applicationStorageDirectory.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

Una preocupación común acerca del uso del modo de ejecución asíncrono es la suposición de que no se puede comenzar a ejecutar una instancia `SQLStatement` si otra instancia `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 `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.

Estrategias para la utilización de bases de datos SQL

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 base de datos previamente llenada

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, sólo 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 sólo 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 directorio de almacenamiento de la aplicación del usuario (u otra ubicación) y use esa base de datos dentro de la aplicación.

Cómo mejorar el rendimiento de la base de datos

Varias técnicas incorporadas en Adobe AIR permiten mejorar el rendimiento de las operaciones de la base de datos en la aplicación.

Además de las técnicas descritas aquí, el modo en que una declaración SQL está escrita también puede afectar el rendimiento de la base de datos. Con frecuencia, hay varias maneras de escribir una declaración SQL `SELECT` para recuperar un determinado conjunto de resultados. En algunos casos, los diferentes métodos requieren más o menos esfuerzo por parte del motor de la base de datos. Este aspecto de mejorar el rendimiento de la base de datos, diseñando declaraciones SQL para un mejor rendimiento, no se incluye en la documentación de Adobe AIR.

Utilizar una instancia `SQLStatement` para cada declaración SQL

Antes de que se ejecute una declaración SQL, el motor de ejecución la prepara (compila) para determinar los pasos que se llevan a cabo internamente para realizar la declaración. Cuando llama al método `SQLStatement.execute()` en una instancia `SQLStatement` que no se ha ejecutado anteriormente, la declaración se prepara automáticamente antes de que se ejecute. En llamadas posteriores al método `execute()`, siempre que la propiedad `SQLStatement.text` no haya cambiado, aún se prepara la declaración. En consecuencia, se ejecuta de forma más rápida.

Para obtener el mayor beneficio de reutilizar las declaraciones, si se deben cambiar los valores entre cada ejecución de declaraciones, utilice los parámetros de declaración para personalizar la declaración. (Los parámetros de declaración se definen usando la propiedad de conjunto asociativa `SQLStatement.parameters`. A diferencia de cambiar la propiedad `text` de la instancia `SQLStatement`, si cambia los valores de los parámetros de declaración el motor de ejecución no necesita preparar la declaración nuevamente. Para más información acerca del uso de parámetros en las declaraciones, consulte [“Utilización de parámetros en declaraciones”](#) en la página 175.

Dado que la preparación y la ejecución de una declaración es una operación que es potencialmente exigente, una buena estrategia es cargar previamente los datos iniciales y luego ejecutar las otras declaraciones en segundo plano. Cargue primero los datos que necesita la aplicación. Cuando se completan las operaciones de inicio de la aplicación, o en otro momento de “inactividad” de la aplicación, ejecute otras declaraciones. Por ejemplo, si la aplicación nunca accede a la base de datos para visualizar la pantalla inicial, espere hasta que se muestre dicha pantalla, luego abra la conexión de la base de datos y finalmente cree las instancias `SQLStatement` y ejecute las que pueda. Como alternativa, supongamos que cuando inicia la aplicación inmediatamente muestra algunos datos como el resultado de una determinada consulta. En ese caso, proceda a ejecutar la instancia `SQLStatement` para esa consulta. Después de que se cargan y se muestran los datos iniciales, cree las instancias `SQLStatement` para las otras operaciones de la base de datos y, si es posible, ejecute las otras declaraciones que se necesitan más adelante.

Cuando vuelve a utilizar una instancia `SQLStatement`, la aplicación necesita mantener una referencia a la instancia `SQLStatement` una vez que se ha preparado. Para mantener una referencia a la instancia, declare la variable como una variable del ámbito de la clase en lugar de una variable en el ámbito de la función. Para ello, una buena manera es estructurar la aplicación para que la declaración SQL se agrupe en una sola clase. Un grupo de declaraciones que se ejecutan en combinación también se pueden agrupar en una sola clase. Al definir la instancia `SQLStatement` o las instancias como variables de miembros de la clase, éstas permanecen siempre que la instancia de la clase agrupada existe en la aplicación. Como mínimo, simplemente puede definir una variable que contiene la instancia `SQLStatement` fuera de una función para que la instancia permanezca en la memoria. Por ejemplo, declare la instancia `SQLStatement` como una variable de miembro en una clase `ActionScript` o como una variable sin función en un archivo `JavaScript`. A continuación puede definir los valores de la declaración y llamar al método `execute()` cuando quiere realmente ejecutar la consulta.

Agrupar múltiples operaciones en una transacción

Supongamos que está ejecutando un gran número de declaraciones SQL que implican añadir o cambiar datos (declaraciones `INSERT` o `UPDATE`). Puede aumentar el rendimiento considerablemente ejecutando todas las declaraciones en una transacción explícita. Si comienza una transacción explícitamente, cada una de las declaraciones se ejecuta en su propia transacción creada automáticamente. Después de que cada transacción (cada declaración) termina de ejecutarse, el motor de ejecución escribe los datos resultantes en el archivo de la base de datos en el disco. Por otra parte, considere qué sucede si crea explícitamente una transacción y ejecuta las declaraciones en el contexto de dicha transacción. El motor de ejecución hace todos los cambios en la memoria, luego escribe todos los cambios en el archivo de la base de datos de una vez cuando se confirma la transacción. Generalmente, escribir los datos en el disco es la parte que lleva más tiempo en la operación. En consecuencia, si se escribe en el disco una sola vez en lugar de una vez por cada declaración SQL puede mejorar significativamente el rendimiento.

Reducir el procesamiento del motor de ejecución

El uso de las siguientes técnicas pueden evitar trabajo innecesario por parte del motor de la base de datos y mejorar el rendimiento de las aplicaciones:

- Siempre especifique explícitamente los nombres de las bases de datos junto con los nombres de las tablas en una declaración. (Use “main” si es la base de datos principal). Por ejemplo, use `SELECT employeeId FROM main.employees` en lugar de `SELECT employeeId FROM employees`. Si se especifica explícitamente el nombre de la base de datos se evita que el motor de ejecución tenga que verificar cada base de datos para encontrar la tabla correspondiente. Asimismo, evita que el motor de ejecución elija la base de datos incorrecta. Siga esta regla aun si `SQLConnection` está conectada a una sola base de datos, ya que en segundo plano `SQLConnection` también está conectada a una base de datos temporal que se accede a través de las declaraciones SQL.
- Siempre especifique explícitamente los nombres de columna en una declaración `SELECT` o `INSERT`.
- Divida las filas devueltas por una declaración `SELECT` que recupera un gran número de filas: consulte [“Recuperación de resultados SELECT en partes”](#) en la página 182.

Evitar cambios de esquemas

Si es posible, se debe evitar cambiar el esquema (estructura de la tabla) de una base de datos una vez que se han añadido los datos en las tablas de la base de datos. Normalmente, un archivo de base de datos está estructurado con las definiciones de tabla al inicio del archivo. Cuando se abre una conexión a una base de datos, el motor de ejecución carga dichas definiciones. Cuando se añaden datos a las tablas de la base de datos, dichos datos se añaden al archivo después de los datos de definición de la tabla. Sin embargo, si se hacen cambios al esquema como añadir una columna a una tabla o añadir una nueva tabla, los nuevos datos de definición de tabla se combinan con los datos de la tabla en el archivo de la base de datos. Si los datos de definición de tabla no se encuentran al inicio del archivo de la base de datos, se tarda más en abrir una conexión a la base de datos ya que el motor de ejecución lee los datos de definición de tabla de diferentes partes en el archivo.

Si se deben realizar cambios al esquema, se puede llamar al método `SQLConnection.compact()` después de completar los cambios. Esta operación reestructura el archivo de la base de datos para que los datos de definición de tabla se ubiquen al inicio del archivo. Sin embargo, la operación `compact()` puede demorar, especialmente a medida que se amplía el archivo de base de datos.

Prácticas recomendadas para utilizar bases de datos SQL locales

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. Para obtener técnicas adicionales para mejorar las aplicaciones de la base de datos, consulte [“Cómo mejorar el rendimiento de la base de datos”](#) en la página 195.

Crear previamente conexiones a la base de datos

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 172

Volver a utilizar las conexiones a la base de datos

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 172

Utilizar el modo de ejecución asíncrono

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 [“Utilización de operaciones sincrónicas y asíncronas de base de datos”](#) en la página 190, 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 sólo 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 múltiples 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. En consecuencia, el tiempo acumulado de múltiples 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 182. Sin embargo, esta técnica sólo se puede utilizar en el modo de ejecución asíncrono. Sólo 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 sólo 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 la cola múltiples 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 193.

Utilizar declaraciones SQL por separado y no cambiar la propiedad text de la declaración SQLStatement

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. Consulte [“Utilizar una instancia SQLStatement para cada declaración SQL”](#) en la página 195 para más información.

Utilizar parámetros de declaración

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 sólo 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 [“Utilización de parámetros en declaraciones”](#) en la página 175 para más información.

Utilizar constantes para los nombres de columnas y parámetros

Cuando no especifica una `itemClass` para una `SQLStatement`, para evitar errores ortográficos, defina las constantes `String` que contienen los nombres de columnas de una tabla. Use dichas constantes en el texto de la declaración y para los nombres de las propiedades cuando recupera valores de los objetos de resultados. Asimismo utilice constantes para los nombres de parámetros.

Capítulo 19: Almacenamiento de datos cifrados

El motor de ejecución de Adobe® AIR™ ofrece un almacén local cifrado persistente para cada aplicación de AIR instalada en el ordenador del usuario. Esto permite que usted pueda guardar y recuperar datos guardados en el disco duro local del usuario en un formato cifrado que no puedan descifrar fácilmente otros usuarios o aplicaciones. Se utiliza un almacén local cifrado e independiente para cada aplicación de AIR, y cada aplicación de AIR usa un almacén local cifrado e independiente para cada usuario.

El almacén local cifrado resulta práctico para guardar información que debe estar protegida, como los datos de inicio de sesión para servicios web.

AIR utiliza DPAPI en Windows y KeyChain en Mac OS para asociar el almacén local cifrado con cada aplicación y cada usuario. El almacén local cifrado utiliza cifrado AES-CBC de 128 bits.

La información en el almacén local cifrado sólo está disponible para contenido de aplicaciones de AIR en el entorno limitado de seguridad de la aplicación.

Utilice los métodos estáticos `setItem()` y `removeItem()` de la clase `EncryptedLocalStore` para guardar y recuperar datos del 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);

var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

El tercer parámetro del método `setItem()`, el parámetro `stronglyBound`, es opcional. Si este parámetro está definido en `true`, el almacén local cifrado brinda un nivel mayor de seguridad, vinculando el elemento guardado con la firma digital y los bits de la aplicación de AIR, así como con el ID del editor de la aplicación, cuando:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, true);
```

Para un elemento guardado con `stronglyBound` definido en `true`, las llamadas posteriores a `getItem()` sólo 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, ésta no podrá leer datos fuertemente vinculados que se habían escrito en el almacén local cifrado.

Como opción predeterminada, la aplicación de AIR no puede leer el almacén local cifrado de otra aplicación. La opción `stronglyBound` proporciona una vinculación adicional (a los datos de los bits de la aplicación) que impide que una aplicación agresora intente leer datos en el almacén local cifrado de la aplicación mediante un intento de apropiarse del ID del editor de la aplicación.

Si actualiza una aplicación para utilizar otro certificado de firma (utilizando una firma de migración), la versión actualizada no tendrá acceso a ninguno de los elementos en el almacén anterior, aun si el parámetro `stronglyBound` está definido en "false". Para obtener más información, consulte ["Cambio de certificado"](#) en la página 305.

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.reset()`, como en el siguiente ejemplo:

```
EncryptedLocalStore.reset();
```

Al depurar una aplicación en AIR Debug Launcher (ADL), la aplicación utiliza un almacén local cifrado distinto del que se utiliza en la versión instalada de la aplicación.

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.

Los datos del almacén local cifrado se ponen en un subdirectorio del directorio de datos de la aplicación del usuario; la ruta del subdirectorio es Adobe/AIR/ELS seguido del ID de la aplicación.

Capítulo 20: Entorno HTML

Adobe® AIR™ utiliza [WebKit](http://www.webkit.org) (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).

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 sólo 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.

AIR utiliza [WebKit](http://www.webkit.org) (www.webkit.org), también utilizado por el navegador Web Safari, para analizar, maquetar y representar contenido en formato HTML y JavaScript. Las clases y los objetos host integrados en AIR proporcionan una API para funciones que normalmente asociamos con las aplicaciones de escritorio. Estas funciones incluyen la lectura y escritura de archivos y la gestión de ventanas. Adobe AIR también hereda las API de Adobe® Flash® Player, que incluyen funciones como sockets de sonido y binarios.

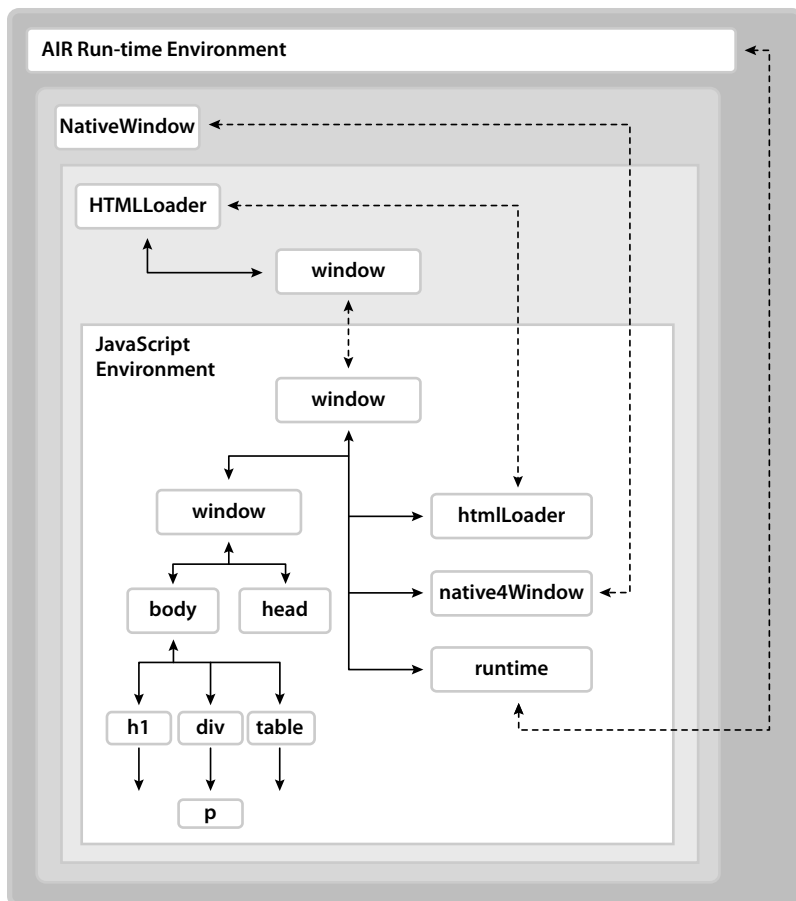
El contenido HTML en AIR no muestra contenido SWF o PDF si se aplican opciones de alfa, escala o transparencia. Para obtener más información, consulte [Consideraciones al cargar el contenido SWF o PDF en una página HTML](#) y “[Transparencia de la ventana](#)” en la página 61.

Información general sobre el entorno HTML

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 ActionScript™ de la clase se describen en “[Utilización de scripts en el contenedor HTML](#)” en la página 242 y también en *Flex 3 ActionScript Language Reference* (*Referencia del lenguaje de ActionScript para Flex 3*).

Entorno JavaScript y su relación con AIR

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: Sólo las páginas instaladas como parte de una aplicación tienen las propiedades `htmlLoader`, `nativeWindow` o `runtime` y sólo 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

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 sólo 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 *punto de acceso limitado*. Un punto de acceso 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 punto de acceso 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 puntos de acceso limitado se establecen definiendo las propiedades `parentSandboxBridge` y `childSandboxBridge` del objeto `window`. Para obtener más información, consulte “Seguridad en HTML” en la página 30 y “Elementos `frame` e `iframe` de HTML” en la página 211.

Plug-ins y objetos incorporados

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 `HTMLoader` 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.

Véase también

“Seguridad en HTML” en la página 30

“Entornos limitados en HTML” en la página 204

“Elementos `frame` e `iframe` de HTML” en la página 211

“Objeto `Window` en JavaScript” en la página 210

“Objeto `XMLHttpRequest`” en la página 205

“Cómo añadir contenido PDF” en la página 256

Extensiones de AIR y WebKit

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 sólo 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. 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).

Nota: AIR no es compatible con las siguientes funciones estándar y ampliadas de Webkit: el método `print()` del objeto `Window` de JavaScript; `plug-ins`, con la excepción de Acrobat o Adobe Reader 8.1+; gráficos vectoriales escalables (SVG), la propiedad `opacity` de CSS.

JavaScript en AIR

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 218.

Entornos limitados en HTML

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 navegador 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](#)” en la página 27.

El acceso al entorno del motor de ejecución y las API de AIR sólo 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 sólo 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 se 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. Sólo 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

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 32 se describen los usos admitidos de la función `eval()`.

Constructores de función

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

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

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, sólo se carga localmente el contenido del subdirectorio `air`.

Para obtener más información, consulte “Elementos `frame` e `iframe` de HTML” en la página 211 y “Seguridad en HTML” en la página 30.

Objeto Canvas

El objeto Canvas define una API para dibujar formas geométricas como líneas, arcos, elipses y polígonos. Para utilizar la API de lienzo, primero se añade un elemento canvas al documento y después se dibuja en el mismo utilizando la API Canvas de JavaScript. Por lo demás, el objeto Canvas se comporta mayormente como una imagen.

El siguiente ejemplo dibuja un triángulo utilizando un objeto Canvas:

```

<html>
<body>
<canvas id="triangleCanvas" style="width:40px; height:40px;"></canvas>
<script>
    var canvas = document.getElementById("triangleCanvas");
    var context = canvas.getContext("2d");
    context.lineWidth = 3;
    context.strokeStyle = "#457232";
    context.beginPath();
        context.moveTo(5,5);
        context.lineTo(35,5);
        context.lineTo(20,35);
        context.lineTo(5,5);
        context.lineTo(6,5);
    context.stroke();
</script>
</body>
</html>

```

Para obtener más documentación sobre la API Canvas, consulte [Safari JavaScript Reference \(Referencia de JavaScript para Safari\)](#) de Apple. Téngase en cuenta que hace poco el proyecto Webkit empezó a cambiar la API Canvas para normalizarla con el [HTML 5 Working Draft \(Borrador de HTML 5\)](#) propuesto por el Web Hypertext Application Technology Working Group (WHATWG) y W3C. Por ello es posible que parte de la documentación de la referencia de JavaScript para Safari no guarde una coherencia con la versión del lienzo que está presente en AIR.

Cookies

En las aplicaciones de AIR sólo 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 las API de AIR proporcionan otros medios de guardar datos persistentes (como las clases `EncryptedLocalStorage` y `FileStream`).

Objeto Clipboard

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
<code>clearData(mimeType)</code>	Borra los datos del portapapeles. Defina el parámetro <code>mimeType</code> en el tipo MIME de los datos a borrar.
<code>getData(mimeType)</code>	Obtiene los datos del portapapeles. Este método sólo se puede llamar en un controlador del evento <code>paste</code> . Defina el parámetro <code>mimeType</code> en el tipo MIME de los datos a devolver.
<code>setData(mimeType, data)</code>	Copia datos en el portapapeles. Defina el parámetro <code>mimeType</code> en el tipo MIME de los datos.

El código JavaScript que esté fuera del entorno limitado de la aplicación sólo 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: Sólo 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 143 y [Using the Pasteboard from JavaScript \(Utilización del Pasteboard de JavaScript\)](#) (Centro de desarrollo de Apple).

Arrastrar y colocar

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
<code>clearData(mimeType)</code>	Borra los datos. Defina el parámetro <code>mimeType</code> en el tipo MIME de la representación de datos a borrar.
<code>getData(mimeType)</code>	Obtiene los datos arrastrados. Este método sólo se puede llamar en un controlador del evento <code>drop</code> . Defina el parámetro <code>mimeType</code> en el tipo MIME de los datos a obtener.
<code>setData(mimeType, data)</code>	Defina los datos a arrastrar. Defina el parámetro <code>mimeType</code> en el tipo MIME de los datos.
<code>types</code>	Un conjunto de cadenas que contiene los tipos MIME de todas las representaciones de datos que están disponibles actualmente en el objeto <code>dataTransfer</code> .
<code>effectsAllowed</code>	Especifica si los datos arrastrados se pueden copiar, mover o vincular, o una combinación de estas operaciones. Defina la propiedad <code>effectsAllowed</code> en el controlador del evento <code>dragstart</code> .
<code>dropEffect</code>	Especifica qué efectos de arrastrar admitidos se pueden utilizar con un destino de arrastre. Defina la propiedad <code>dropEffect</code> en el controlador del evento <code>dragEnter</code> . 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 <code>dropEffect</code> , se elige un efecto para la propiedad <code>effectsAllowed</code> . 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 [“Arrastrar y colocar”](#) en la página 127 y [Using the Drag-and-Drop from JavaScript \(Utilización de arrastrar y colocar de JavaScript\)](#) (Centro de desarrollo de Apple).

Propiedades innerHTML y outerHTML

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 sólo 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 en HTML](#)” en la página 30.

Métodos `Document.write()` y `Document.writeln()`

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`

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 `contentEditable` de HTML](#)” en la página 214.

Eventos `unload` (para objetos `body` y `frameset`)

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

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` sólo se define para contenido que se encuentra en el entorno limitado de la aplicación, y sólo para el documento principal de una página con fotogramas o iframes.

Consulte el apartado “[Utilización del archivo AIRAliases.js](#)” en la página 223.

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` sólo se define para contenido que se encuentra en el entorno limitado de la aplicación, y sólo 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` sólo se define para contenido que se encuentra en el entorno limitado de la aplicación, y sólo 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 “[Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad](#)” en la página 232.

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 221.

Función `Window.open()` Cuando lo llama un código que se ejecuta en un entorno limitado ajeno a la aplicación, el método `open()` sólo 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 35.

Objeto `air.NativeApplication`

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;
```

Sólo se tiene acceso al objeto `NativeApplication` desde el entorno limitado de la aplicación. El objeto `NativeApplication` se describe en detalle en “[Trabajo con información sobre el motor de ejecución y el sistema operativo](#)” en la página 280.

Esquema URL de JavaScript

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.

Extensiones de HTML

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 211

“[Elemento `Canvas` en HTML](#)” en la página 213

“[Controladores de eventos de elementos de HTML](#)” en la página 214

Elementos `frame` e `iframe` de HTML

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 XMLHttpRequest a cualquier dominio remoto, incluya el atributo `allowCrossDomainXHR="allowCrossDomainXHR"` en la etiqueta "frame" inicial. De forma predeterminada, el contenido ajeno a la aplicación sólo 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 sólo 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 ondominialize Especifica un controlador de eventos para el evento `dominialize` 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 `dominialize` 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 `dominialize`. 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 XMLHttpRequest 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` ilustra la forma en que el contenido secundario puede tener acceso al puente de entorno limitado principal:

```
<html>
<head>
<script>
document.write(window.parentSandboxBridge.testProperty);
</script>
</head>
<body></body>
</html>
```

Para obtener más información, consulte [“Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad”](#) en la página 232 y [“Seguridad en HTML”](#) en la página 30.

Elemento Canvas en HTML

Define un área de dibujo para uso con la API Canvas de WebKit. No se pueden especificar comandos de gráficos en la propia etiqueta. Para dibujar en el lienzo, llame a los métodos de dibujo en el lienzo a través de JavaScript.

```
<canvas id="drawingAtrium" style="width:300px; height:300px;"></canvas>
```

Para obtener más información, consulte el apartado “Objeto Canvas” en la página 206.

Controladores de eventos de elementos de HTML

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.
onsearch	Se llama al copiar un elemento. Apple docs correct ?)
onselectstart	Se llama al iniciarse una selección.

Atributo contentEditable de HTML

El atributo `contentEditable` puede añadirse a cualquier elemento de 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 209.

Extensiones de CSS

WebKit admite varias propiedades de CSS ampliadas. En la tabla siguiente se relacionan las propiedades ampliadas admitidas. WebKit contiene más propiedades no estándar, pero no son totalmente compatibles con AIR, sea porque aún están en vías de desarrollo en WebKit, sea porque son funciones experimentales que quizá se acaben por eliminar.

Nombre de la propiedad de CSS	Valores	Descripción
<code>-webkit-border-horizontal-spacing</code>	Unidad de longitud que no sea negativa	Especifica el componente horizontal del espaciado de bordes.
<code>-webkit-border-vertical-spacing</code>	Unidad de longitud que no sea negativa	Especifica el componente vertical del espaciado de bordes.
<code>-webkit-line-break</code>	<code>after-white-space</code> (después de espacio blanco), <code>normal</code>	Especifica la regla de salto de línea a utilizar para texto en chino, japonés o coreano.
<code>-webkit-margin-bottom-collapse</code>	<code>collapse</code> (contraer), <code>discard</code> (desechar), <code>separate</code> (separar)	Define cómo se contrae el margen inferior de una celda de tabla.
<code>-webkit-margin-collapse</code>	<code>collapse</code> (contraer), <code>discard</code> (desechar), <code>separate</code> (separar)	Define cómo se contraen los márgenes superior e inferior de una celda de tabla.
<code>-webkit-margin-start</code>	Cualquier unidad de longitud	La anchura del margen inicial. Para texto leído de izquierda a derecha, esta propiedad tiene prioridad sobre el margen izquierdo. Para texto leído de derecha a izquierda, esta propiedad tiene prioridad sobre el margen derecho.
<code>-webkit-margin-top-collapse</code>	<code>collapse</code> (contraer), <code>discard</code> (desechar), <code>separate</code> (separar)	Define cómo se contrae el margen superior de una celda de tabla.
<code>-webkit-nspace-mode</code>	<code>normal</code> , <code>espacio</code>	Define el comportamiento de espacios duros en el contenido encerrado.
<code>-webkit-padding-start</code>	Cualquier unidad de longitud	Especifica la anchura del relleno inicial. Para texto leído de izquierda a derecha, esta propiedad tiene prioridad sobre el valor de relleno izquierdo. Para texto leído de derecha a izquierda, esta propiedad tiene prioridad sobre el valor de relleno derecho.
<code>-webkit-rtl-ordering</code>	<code>logical</code> (lógico), <code>visual</code>	Suprime la gestión predeterminada de texto mezclado de izquierda a derecha y derecha a izquierda.
<code>-webkit-text-fill-color</code>	Nombre o valor numérico de cualquier color	Especifica el color de relleno del texto.
<code>-webkit-text-security</code>	<code>circle</code> (círculo), <code>disc</code> (disco), <code>none</code> (ninguno), <code>square</code> (cuadrado)	Especifica la forma que sustituirá los caracteres en el campo de introducción de la contraseña.

Nombre de la propiedad de CSS	Valores	Descripción
-webkit-user-drag	<ul style="list-style-type: none">• auto: comportamiento predeterminado• element: se arrastra el elemento entero• none: no se puede arrastrar el elemento	Suprime el comportamiento automático de arrastrar.
-webkit-user-modify	read-only (sólo lectura), read-write (lectura y escritura), read-write-plaintext-only (lectura y escritura, sólo texto sin formato)	Especifica si se puede editar el contenido de un elemento.
-webkit-user-select	<ul style="list-style-type: none">• auto: comportamiento predeterminado• none: no se puede seleccionar el elemento• text: sólo se puede seleccionar texto en el elemento	Especifica si el usuario puede seleccionar el contenido de un elemento.

Para obtener más documentación, consulte la referencia sobre CSS de Apple Safari (<http://developer.apple.com/documentation/AppleApplications/Reference/SafariCSSRef/>).

Capítulo 21: Programación en HTML y JavaScript

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

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

El siguiente código carga una URL en un objeto HTMLLoader y define el objeto como uno objeto secundario de un objeto Sprite:

```
var container:Sprite;
var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
container.addChild(html);
```

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 asociados con HTML”](#) en la página 236.

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

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);
```

El contenido cargado a través del método `loadString()` se coloca en el entorno limitado de seguridad de la aplicación, y de este modo, proporciona un acceso completo a las API de AIR.

Reglas importantes de seguridad cuando se utiliza HTML en aplicaciones de AIR

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 proveniente de diferentes fuentes se coloca en clasificaciones de seguridad por separado, llamados entornos limitados (consulte [“Entornos limitados”](#) en la página 27 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 [“Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad”](#) en la página 232.

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 218.

Para más información, consulte [“Seguridad en HTML”](#) en la página 30.

Cómo evitar errores de JavaScript relacionados con la seguridad

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

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 de función](#)

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 32.*

Adobe mantiene 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

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 reescribir 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 232. 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 233.

Función eval()

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

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

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

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()

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')", 10);
```

con:

```
setTimeout(alert('Timeout'), 10);
```

O bien cuando la función requiere que el originador 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 de función

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

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`

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

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 232

document.write() y document.writeln()

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

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

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

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
- Utilización de las bases de datos SQL locales
- Control de los menús de aplicación y de ventana
- Acceso a los sockets para red
- Utilización 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 *Referencia del lenguaje de Adobe AIR para desarrolladores de HTML*. 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.

Nota: para más información, consulte *Clases, paquetes y espacios de nombres de ActionScript 3.0* y *Aspectos básicos de ActionScript para desarrolladores de JavaScript*.

Utilización del archivo AIRAliases.js

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 *Referencia del lenguaje de Adobe AIR para desarrolladores de HTML*.

Además de las clases runtime comúnmente utilizadas, el AIRAliases.js incluye alias 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

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 [“Utilización de esquemas de URL de AIR en direcciones URL”](#) en la página 288.

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 [“Peticiones de URL y redes”](#) en la página 286.

Disponibilidad de objetos ActionScript en JavaScript

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

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`, sólo 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

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.

Cómo quitar detectores de eventos

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 240.

Acceso a objetos JavaScript y DOM HTML desde ActionScript

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

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 sí 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.

En el siguiente ejemplo se muestra el uso de la etiqueta HTML `object` para mostrar un archivo SWF dentro del contenido HTML. 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>
</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);
    displayContainer.appendChild(createSWFObject(urlString, 650, 650));
}
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);
    return SWFObject;
}
</script>
```

Utilización de las bibliotecas de ActionScript en una página HTML

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.

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 3 SDK y se describe en la documentación [Flex 3 SDK](#).

Acceso a objetos JavaScript y DOM HTML desde un archivo importado ActionScript

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

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

Una vez que el objeto `HTMLLoader` ha distribuido el evento `complete`, puede examinar y manipular los estilos CSS en una página.

Por ejemplo, considere 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>
```

Después de que un objeto HTMLLoader carga este contenido, puede manipular los estilos CSS en la página a través del conjunto `cssRules` del conjunto `window.document.styleSheets`, 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`.

Utilización de scripts entre contenidos en diferentes entornos limitados de seguridad

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

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 *punto de conexión 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 211, “Seguridad en HTML” en la página 30 y “Objeto XMLHttpRequest” en la página 205.

Carga de contenido de aplicación en un entorno limitado que no pertenece a la aplicación

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

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

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

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 intrínsecamente 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

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 sólo 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 22: Gestión de eventos asociados con HTML

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 sólo 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

Un objeto HTMLLoader distribuye los siguientes eventos de ActionScript™:

Evento	Descripción
<code>htmlDOMInitialize</code>	Se distribuye al crearse el documento HTML, pero antes de que se analicen scripts o se añadan a la página nodos del DOM.
<code>complete</code>	Se distribuye cuando se ha creado el DOM de HTML como respuesta a una operación de cargar, inmediatamente después del evento <code>onload</code> en la página HTML.
<code>htmlBoundsChanged</code>	Se distribuye cuando ha cambiado al menos una de las propiedades <code>contentWidth</code> o <code>contentHeight</code> .
<code>locationChange</code>	Se distribuye cuando ha cambiado la propiedad de ubicación de HTMLLoader.
<code>scroll</code>	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 <code>window.scrollTo()</code> . Escribir texto en una zona de entrada de texto también puede dar lugar a un evento de desplazamiento.
<code>uncaughtScriptException</code>	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 236.

Gestión de eventos DOM con ActionScript

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():void {
    trace("You clicked it!");
}

```

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

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

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

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

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

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 ésta 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);
}
```

Otra forma de impedir que se produzca este error es eliminar un detector de eventos en cuanto se ejecute. 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

El método `hasEventListener()` permite verificar si existe un detector de eventos en un objeto.

Capítulo 23: Utilización de scripts en el contenedor HTML

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 las 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 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`.

Propiedades de visualización de objetos HTMLLoader

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

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
<code>x, y</code>	Coloca el objeto dentro del contenedor principal.
<code>width, height</code>	Cambia las dimensiones del área de visualización.
<code>visible</code>	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

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

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

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.

Nota: el motor de ejecución no puede mostrar el contenido SWF o PDF en ventanas transparentes.

Para más información sobre la carga de estos tipos de medios en un `HTMLLoader`, consulte Carga de contenido SWF en una página HTML y “Cómo añadir contenido PDF” en la página 256.

Propiedades de visualización avanzadas

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.
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

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. 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 254).

Acceso a la lista del historial HTML

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
<code>historyLength</code>	La longitud total de la lista del historial, incluyendo las entradas hacia atrás y hacia adelante.
<code>historyPosition</code>	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”.
<code>historyAt()</code>	Devuelve el objeto <code>URLRequest</code> que corresponde a la entrada del historial en la posición específica en la lista del historial.
<code>historyBack()</code>	Navega hacia atrás en la lista del historial, si es posible.
<code>historyForward()</code>	Navega hacia adelante en la lista del historial, si es posible.
<code>historyGo()</code>	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 en la lista del historial se almacenan como objetos de tipo `HistoryListItem`. La clase `HistoryListItem` tiene las siguientes propiedades:

Propiedad	Descripción
<code>isPost</code>	Está definida en <code>true</code> si la página HTML incluye datos POST.
<code>originalUrl</code>	La URL original de la página HTML, antes de cualquier redirección.
<code>title</code>	El título de la página HTML.
<code>url</code>	La URL de la página HTML.

Configuración del agente de usuario que se utiliza al cargar contenido HTML

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 `URLRequests` utilizados con el método `load()` de los 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

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

JavaScript proporciona varias API para controlar la ventana que muestra el contenido HTML. En AIR, estas API se pueden sustituir implementando una clase `HTMLHost` personalizada.

Ampliación de la clase `HTMLHost`

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

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`:

- 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

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()

Anule el método `setWindowRect()` para gestionar cambios en los límites del contenido HTML. Se llama al método `setWindowRect()` 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 setWindowRect(value:Rectangle):void
{
  htmlLoader.stage.nativeWindow.bounds = value;
}

```

Gestión de llamadas JavaScript a window.open()

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 pasado como un 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 <code>HTMLWindowCreateOptions</code>	Parámetro correspondiente en la cadena de funciones en la llamada JavaScript a <code>window.open()</code>
<code>fullscreen</code>	<code>fullscreen</code>
<code>height</code>	<code>height</code>
<code>locationBarVisible</code>	<code>location</code>
<code>menuBarVisible</code>	<code>menubar</code>
<code>resizeable</code>	<code>resizable</code>
<code>scrollBarsVisible</code>	<code>scrollbars</code>
<code>statusBarVisible</code>	<code>status</code>
<code>toolBarVisible</code>	<code>toolbar</code>
<code>width</code>	<code>width</code>
<code>x</code>	<code>left</code> o <code>screenX</code>
<code>y</code>	<code>top</code> o <code>screenY</code>

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()`

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`

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`

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()`

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

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
<code>visible</code>	Un valor Boolean que especifica si la ventana está inicialmente visible (<code>true</code>) o no (<code>false</code>).
<code>windowInitOptions</code>	Un objeto <code>NativeWindowInitOptions</code> . La clase <code>NativeWindowInitOptions</code> define las opciones de inicialización de un objeto <code>NativeWindow</code> , 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.
<code>scrollBarsVisible</code>	Si hay barras de desplazamiento (<code>true</code>) o no (<code>false</code>).
<code>bounds</code>	Un objeto <code>Rectangle</code> 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: 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

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 247.

Capítulo 24: Cómo añadir contenido PDF

Las aplicaciones que se ejecutan en Adobe® AIR™ pueden representar no sólo 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 sólo una parte de la interfaz. El plug-in de Adobe Reader controla la visualización de archivos PDF en una aplicación de AIR, de modo que se mantienen las modificaciones de la interfaz de barra de herramientas de Reader (como las de posición, anclaje y visibilidad) 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

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
<code>HTMLPDFCapability.STATUS_OK</code>	Se detecta una versión suficiente (8.1 o posterior) de Adobe Reader y se puede cargar contenido PDF en un objeto <code>HTMLLoader</code> .
<code>HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND</code>	No se detecta ninguna versión de Adobe Reader. Un objeto <code>HTMLLoader</code> no puede mostrar contenido PDF.
<code>HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD</code>	Se ha detectado una versión de Adobe Reader pero es demasiado antigua. Un objeto <code>HTMLControl</code> no puede mostrar contenido PDF.
<code>HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD</code>	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 <code>HTMLControl</code> no puede visualizar contenido PDF. Un objeto <code>HTMLControl</code> no puede mostrar contenido PDF.

Nota: en Windows, si está funcionando Adobe Acrobat o Adobe Reader versión 7.x o posterior en el sistema del usuario, ésta 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 `pdfCapability` 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 ésta 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.

El siguiente código detecta si un usuario puede visualizar contenido PDF en una aplicación de AIR, y en caso negativo, 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

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 subdirectorío de PDF del directorío de la aplicación:

```
app:/js_api_reference.pdf
```

Para obtener más información sobre los esquemas de URL de AIR, consulte “[Utilización de esquemas de URL de AIR en direcciones URL](#)” en la página 288.

Uso de scripts con el contenido PDF

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>.

Aspectos básicos de la comunicación HTML-PDF

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 sólo 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

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 ver más información, consulte “[Programación en HTML y JavaScript](#)” en la página 217.

Limitaciones conocidas del contenido PDF en AIR

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.
- El contenido PDF no se visualiza en una ventana en modo de pantalla completa (cuando la propiedad `displayState` de la clase Stage está definida en `StageDisplayState.FULL_SCREEN` o `StageDisplayState.FULL_SCREEN_INTERACTIVE`).

- No se pueden cambiar las propiedades visuales de un objeto HTMLLoader que contiene un archivo PDF. La modificación de las propiedades `filters`, `alpha`, `rotation` o `scaling` del objeto HTMLLoader hacen invisible el archivo PDF hasta que se restauren las propiedades. También es el caso si se modifican estas propiedades para un contenedor de objetos de visualización que contenga el objeto HTMLLoader.
- La propiedad `scaleMode` del objeto Stage del objeto NativeWindow que contiene el contenido PDF debe estar definido en `StageScaleMode.NO_SCALE`.
- 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 1.0.

Capítulo 25: Utilización de la administración de derechos digitales

El servidor Adobe® Flash® Media Rights Management Server (FMRMS) proporciona a los editores de medios la capacidad de distribuir contenido, más concretamente archivos FLV y MP4, y recuperar los costes de producción a través de una compensación directa (pagada por los usuarios) o indirecta (pagada por la publicidad) por parte de los consumidores. Los editores distribuyen medios en forma de archivos FLV cifrados que se pueden descargar y reproducir en Adobe® Media Player™, o en cualquier aplicación de AIR que haga uso de la API de administración de derechos digitales (DRM).

Con FMRMS los proveedores de contenido pueden utilizar licencias basadas en la identidad para proteger el contenido mediante el uso de credenciales de usuario. Tomemos el caso de un consumidor que desea ver un programa de televisión sin la publicidad que lo acompaña. Para evitar ver los avisos publicitarios, el consumidor se abona, pagando una suma al editor del contenido. El usuario podrá entonces utilizar su credencial de autenticación para obtener acceso y reproducir el programa sin que aparezca la publicidad. Otro consumidor quizá desee ver el contenido fuera de línea mientras está de viaje, sin acceso a Internet. Tras abonarse y pagar al editor del contenido por un servicio superior, la credencial de autenticación del usuario le permite tener acceso al programa y descargarlo del sitio Web del editor. El usuario puede ver el contenido fuera de línea durante el período admitido. El contenido también está protegido por las credenciales del usuario y no se puede compartir con otros usuarios.

Si un usuario intenta reproducir un archivo con cifrado DRM, la aplicación se comunica con el servidor FMRMS, que a su vez lo hace con el sistema del editor de contenido a través de su interfaz de proveedor de servicios para autenticar al usuario y recuperar la licencia, una licencia que determina si el usuario tiene permiso para acceder al contenido y, en caso afirmativo, por cuánto tiempo. La licencia determina también si el usuario tiene acceso al contenido estando fuera de línea y, en caso afirmativo, durante cuánto tiempo. Se necesitan las credenciales de usuario para determinar el acceso al contenido cifrado.

El uso de licencias basado en la identidad también admite el acceso anónimo. Por ejemplo: el proveedor puede utilizar el acceso anónimo para distribuir contenido con publicidad o permitir el acceso libre al contenido actual durante un número especificado de días. El material histórico puede considerarse contenido premium que debe pagarse y para el que se exige el uso de las credenciales de usuario. El proveedor de contenido también puede especificar y restringir el tipo y la versión de reproductor que se necesita para su contenido.

Aquí se describe cómo activar la aplicación de AIR para que reproduzca contenido protegido mediante el cifrado para la administración de derechos digitales. No hace falta comprender cómo cifrar contenido con DRM, pero se da por sentado que usted tiene acceso a contenido con cifrado DRM y se comunica con el servidor FMRMS para autenticar al usuario y recuperar la licencia.

Para información general sobre FMRMS, incluida la creación de normas, consulte la documentación que se incluye con FMRMS.

Para obtener información sobre Adobe Media Player, consulte la ayuda de Adobe Media Player que está disponible en Adobe Media Player.

Información suplementaria en línea sobre la administración de derechos digitales

Encontrará más información sobre la administración de derechos digitales en las fuentes siguientes:

Referencia del lenguaje

- [DRMAuthenticateEvent](#)
- [DRMErrorEvent](#)
- [DRMStatusEvent](#)
- [NetStream](#)

Artículos y muestras del Centro de desarrollo de Adobe

- [Centro de desarrollo de Adobe AIR para Flash](#) (en inglés); busque "digital rights management" o "drm"

Aspectos básicos del flujo de trabajo con FLV cifrado

Existen cuatro tipos de eventos: `StatusEvent`, `DRMAuthenticateEvent`, `DRMErrorEvent` y `DRMStatusEvent`, que pueden distribuirse cuando una aplicación de AIR no logra reproducir un archivo con cifrado DRM. Para poder utilizar estos archivos la aplicación debe añadir detectores de eventos para controlar los eventos DRM.

A continuación se reseña el flujo de trabajo para que una aplicación de AIR recupere y reproduzca el contenido protegido con cifrado DRM:

- 1 La aplicación utiliza un objeto `NetStream` para intentar reproducir un archivo FLV o MP4. Si el contenido está cifrado, se distribuye un evento `events.StatusEvent` con el código, `DRM.encryptedFLV`, que indica que el archivo FLV está cifrado.

***Nota:** si una aplicación no reproduce el archivo con cifrado DRM, puede detectar el evento `Status` distribuido cuando encuentra un contenido cifrado y después avisar al usuario de que el archivo no es compatible y cerrar la conexión.*

- 2 Si se trata de un archivo con cifrado anónimo, lo cual quiere decir que cualquier usuario puede ver el contenido sin necesidad de introducir las credenciales de autenticación, la aplicación de AIR continúa con el último paso de este flujo de trabajo. Si el archivo requiere una licencia basada en la identidad -para lo cual se necesita la credencial del usuario- el objeto `NetStream` genera un objeto de evento `DRMAuthenticateEvent`. El usuario debe facilitar sus credenciales de autenticación antes de que pueda empezar la reproducción.
- 3 La aplicación de AIR debe proveer un mecanismo para recoger las credenciales de autenticación necesarias. Las propiedades `usernamePrompt`, `passwordPrompt` y `urlPrompt` de la clase `DRMAuthenticationEvent`, proporcionadas por el servidor de contenido, sirven para indicar al usuario información sobre los datos que debe introducir. Estas propiedades se pueden utilizar para crear una interfaz de usuario para recuperar las credenciales de usuario necesarias. Por ejemplo: la cadena de valor `usernamePrompt` puede indicar que el nombre de usuario debe ser una dirección de correo electrónico.

***Nota:** AIR no proporciona una interfaz de usuario predeterminada para recoger las credenciales de autenticación. La interfaz de usuario la debe crear el desarrollador de la aplicación, quien deberá también controlar los eventos `DRMAuthenticateEvent`. Si la aplicación no proporciona un detector de eventos para objetos `DRMAuthenticateEvent`, el objeto con cifrado DRM permanece en estado de "espera de las credenciales", por lo que el contenido no estará disponible.*

- 4 Una vez que la aplicación obtiene las credenciales de usuario, las pasa al objeto NetStream con el método `setDRMAuthenticationCredentials()`. Esto indica al objeto NetStream que debe intentar autenticar al usuario en la próxima oportunidad. AIR pasa entonces la credencial al servidor FMRMS para que se autentique. Si se autentica el usuario, la aplicación continúa con el paso siguiente.

Si se produce un error de autenticación, se distribuye un nuevo evento `DRMAuthenticateEvent` y la aplicación vuelve al paso 3. Este proceso se repite de forma indefinida. La aplicación debe proporcionar un mecanismo para controlar y limitar los intentos reiterados de autenticación. Por ejemplo: la aplicación podría permitir al usuario cancelar el intento que podría cerrar la conexión NetStream.

- 5 Una vez autenticado el usuario, o si se utiliza el cifrado anónimo, el subsistema DRM recupera la licencia. Éste se utiliza para comprobar si el usuario está autorizado a ver el contenido. La información de la licencia puede aplicarse tanto a los usuarios autenticados como a los anónimos. Por ejemplo: tanto los usuarios autenticados como los anónimos pueden tener acceso al contenido durante un tiempo especificado antes de que el contenido caduque, o pueden no tener acceso al contenido porque el proveedor del contenido no ofrece compatibilidad con versión de la aplicación de visualización.

Si no se produce un error y se autoriza al usuario ver el contenido, se distribuye el objeto de evento `DRMStatusEvent` y la aplicación de AIR inicia la reproducción. El objeto `DRMStatusEvent` contiene la información de la licencia asociada, que identifica la póliza del usuario y sus permisos. Por ejemplo: contiene información sobre si el contenido puede estar disponible fuera de línea o cuándo vence la licencia y deja de poder verse el contenido. 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 acceso fuera de línea, la licencia se pone en caché y el contenido cifrado se descarga al equipo del usuario, quedando accesible durante el período de validez fuera de línea. 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.

Todos los errores relacionados con la DRM acaban en la distribución del objeto de evento `DRMErrorEvent` por la aplicación. AIR trata el error de autenticación de DRM volviendo a activar el objeto de evento `DRMAuthenticationEvent`. Los demás eventos de error los debe controlar explícitamente la aplicación. Esto incluye 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 puede aún no tener acceso al contenido porque no se han pagado los derechos. Esto también puede suceder si hay dos usuarios, ambos miembros registrados con el mismo editor de medios, que intentan compartir contenido que sólo ha pagado uno de ellos. La aplicación debe notificar al usuario el error, como las restricciones del contenido, además de ofrecer una alternativa, como instrucciones sobre cómo registrarse y pagar por los derechos de ver el contenido.

Cambios en la clase NetStream

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. Como parte de AIR, la clase NetStream incluye cuatro nuevos eventos relacionados con DRM:

Evento	Descripción
drmAuthenticate	<p>Definido en la clase DRMAuthenticateEvent, este evento se distribuye cuando un objeto NetStream intenta reproducir contenido con cifrado de administración de derechos digitales (DRM) que requiere una credencial de usuario para la autenticación para poder reproducirse.</p> <p>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.</p>
drmError	<p>Definido en la clase DRMErrorEvent, este evento se distribuye cuando un objeto NetStream, al intentar reproducir un archivo con cifrado de administración de derechos digitales (DRM), encuentra un error relacionado con DRM. Por ejemplo: si falla la autorización del usuario, se distribuye un objeto de evento de error de DRM. Esto puede ser porque el usuario no ha adquirido los derechos para ver el contenido o porque el proveedor del contenido no ofrece compatibilidad con la aplicación que se utiliza para visualizarlo.</p>
drmStatus	<p>Definido en la clase DRMStatusEvent, este evento se distribuye cuando el contenido con cifrado de administración de derechos digitales (DRM) empieza a reproducirse (si el usuario está autenticado y autorizado a reproducir el contenido). El objeto DRMStatusEvent contiene información relacionada con la licencia, por ejemplo sobre si el contenido puede estar disponible fuera de línea o cuándo vence la licencia y deja de poder verse el contenido.</p>
status	<p>Definido en events.StatusEvent, este evento se distribuye solamente cuando la aplicación intenta reproducir contenido con cifrado de administración de derechos digitales (DRM) invocando al método NetStream.play(). El valor de la propiedad de código de estado es "DRM.encryptedFLV".</p>

La clase NetStream incluye los siguientes métodos específicos para DRM:

Método	Descripción
resetDRMVouchers()	<p>Elimina de la memoria caché local todos los datos de la licencia de administración de derechos digitales (DRM) para el contenido actual. Para que el usuario pueda acceder al contenido cifrado, la aplicación debe volver a descargar la licencia.</p> <p>En el siguiente ejemplo, el código elimina las licencias para un objeto NetStream:</p> <pre>NetStream.resetDRMVouchers(); air.NetStream.resetDRMVouchers();</pre>
setDRMAuthenticationCredentials()	<p>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", las credenciales proporcionadas se autentican con el servidor FMRMS. 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: la opción "proxy" permite a la aplicación realizar la autenticación ante un servidor proxy si una empresa exige que se lleve a cabo este paso 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 el servidor FMRMS para poder obtener la licencia y reproducir el contenido. Se puede utilizar setDRMAuthenticationcredentials() una segunda vez, con la opción "drm", para realizar la autenticación ante el servidor FMRMS.</p>

En el siguiente código se configuran el nombre de usuario ("administrator"), la contraseña ("password") y el tipo de autenticación "drm" para autenticar al usuario. El método setDRMAuthenticationCredentials() debe facilitar credenciales que coincidan con las credenciales conocidas y aceptadas por el proveedor de contenido (las mismas con las que se obtuvo permiso para 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");
}

var connection:NetConnection = new air.NetConnection();
connection.connect(null);

var videoStream= new air.NetStream();

videoStream.addEventListener(air.DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

function drmAuthenticateEventHandler(event)
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Utilización de la clase DRMStatusEvent

Un objeto NetStream distribuye un objeto DRMStatusEvent cuando el contenido protegido por administración de derechos digitales (DRM) empieza a reproducirse satisfactoriamente (si se ha verificado la licencia y si el usuario está autenticado y autorizado a 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 varios motivos por los cuales se puede denegar el acceso a los usuarios anónimos. Por ejemplo: un usuario anónimo puede no tener acceso al contenido porque éste ha caducado.

El objeto DRMStatusEvent contiene información relacionada con la licencia, por ejemplo sobre si el contenido puede estar disponible fuera de línea o cuándo vence la licencia 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

La clase DRMStatusEvent incluye las siguientes propiedades:

Propiedad	Descripción
detail	Una cadena que explica el contexto del evento del estado. En DRM 1.0, el único valor válido es DRM.voucherObtained.
isAnonymous	Indica si el contenido, protegido por cifrado DRM, 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	Indica si el contenido, protegido por cifrado DRM, 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.
offlineLeasePeriod	La cantidad de días restantes durante los cuales se puede ver el contenido fuera de línea.
normativas	Un objeto personalizado que puede contener propiedades DRM personalizadas.
voucherEndDate	La fecha en que vence indefectiblemente la licencia y el contenido ya no se podrá ver.

Creación de un controlador para el evento DRMStatusEvent

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 con cifrado DRM.

```
private function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event.toString());
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event.toString());
}
```

Utilización de la clase DRMAuthenticateEvent

El objeto DRMAuthenticateEvent se distribuye cuando un objeto NetStream intenta reproducir contenido con cifrado de administración de derechos digitales (DRM) que requiere una credencial de usuario para la autenticación para poder reproducirse.

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 podría proveer al usuario una interfaz de usuario sencilla para introducir los valores para nombre de usuario y contraseña, y opcionalmente el valor para tipo también. La aplicación de AIR debe proporcionar, además, un mecanismo para controlar y limitar los intentos reiterados de autenticación.

Propiedades de DRMAuthenticateEvent

La clase DRMAuthenticateEvent incluye las siguientes propiedades:

Propiedad	Descripción
authenticationType	Indica si las credenciales facilitadas son para autenticación frente al servidor FMRS ("drm") o un servidor proxy ("proxy"). Por ejemplo: la opción "proxy" permite a la aplicación realizar la autenticación ante un servidor proxy si una empresa exige que se lleve a cabo este paso 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 el servidor FMRS para poder obtener la licencia y reproducir el contenido. Se puede utilizar setDRMAuthenticationcredentials() una segunda vez, con la opción "drm", para realizar la autenticación ante el servidor FMRS.
header	El encabezado del archivo de contenido cifrado suministrado por el servidor. Contiene información acerca del contexto del contenido cifrado.
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.

Creación de un controlador para el evento DRMAuthenticateEvent

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");
}

var connection:NetConnection = new air.NetConnection();
connection.connect(null);

var videoStream= new air.NetStream();

videoStream.addEventListener(air.DRMAuthenticateEvent.DRM_AUTHENTICATE,
    drmAuthenticateEventHandler)

function drmAuthenticateEventHandler(event)
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

Creación de una interfaz para recuperar las credenciales de usuario

Si el contenido protegido por DRM 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.

Utilización de la clase DRMErrorEvent

AIR distribuye un objeto DRMErrorEvent cuando un objeto NetStream, al intentar reproducir un archivo con cifrado de administración de derechos digitales (DRM), encuentra un error relacionado con DRM. En el caso de credenciales de usuario no válidas, el objeto DRMAuthenticateEvent se ocupa del error distribuyéndose reiteradas veces hasta que el usuario introduzca las credenciales válidas o la aplicación de AIR prohíba más intentos. La aplicación debe detectar si hay otros eventos de error de DRM para detectar, identificar y controlar los errores relacionados con DRM.

Si un usuario introduce credenciales válidas, aun así es posible que no se le permita ver el contenido cifrado, dependiendo de las condiciones de la licencia de DRM. Por ejemplo: un usuario podría intentar ver el contenido en una aplicación no autorizada, es decir, una aplicación que no ha sido 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

La clase DRMErrorEvent incluye la siguiente propiedad:

subErrorID	Indica el ID de error menor con más información sobre el problema subyacente.
------------	---

En la tabla siguiente se relacionan los errores que notifica el objeto DRMErrorEvent:

Código de error grave	Código de error menor	Detalles del error	Descripción
1001	0		Error de autenticación del usuario.
1002	0		Flash Media Rights Management Server (FMRMS) no admite Secure Sockets Layer (SSL).
1003	0		El contenido ha caducado y ya no está disponible para que se vea.
1004	0		Error de autorización del usuario. Esto puede suceder, por ejemplo, si el usuario no ha adquirido el contenido y por lo tanto no goza de los derechos para verlo.
1005	0	<i>Server URL</i>	No hay conexión con el servidor.
1006	0		Se necesita actualizar el cliente; es decir, Flash Media Rights Management Server (FMRMS) requiere un nuevo motor de administración de derechos digitales (DRM).
1007	0		Error interno genérico.
1008	<i>Código de error de descifrado detallado</i>		Clave de licencia incorrecta.
1009	0		El contenido FLV está dañado.

Código de error grave	Código de error menor	Detalles del error	Descripción
1010	0	<i>publisherID: applicationID</i>	El ID de la aplicación de visualización no coincide con un ID válido admitido por el editor del contenido.
1011	0		La versión de la aplicación no coincide con lo especificado en la póliza.
1012	0		Error de verificación de la licencia asociada con el contenido cifrado; indica que el contenido puede estar dañado.
1013	0		No se pudo guardar la licencia asociada con el contenido cifrado.
1014	0		Error de verificación de la integridad del encabezado de FLV; indica que el contenido puede estar dañado.

Código de error grave	ID de error menor	Detalles del error	Descripción
3300	Código de error de Adobe Policy Server		La aplicación detectó una licencia no válida asociada con el contenido.
3301	0		Error de autenticación del usuario.
3302	0		Flash Media Rights Management Server (FMRMS) no admite Secure Sockets Layer (SSL).
3303	0		El contenido ha caducado y ya no está disponible para que se vea.
3304	0		Error de autorización del usuario. Esto puede suceder aunque el usuario esté autenticado; por ejemplo, puede que no haya adquirido los derechos de ver el contenido.
3305	0	<i>Server URL</i>	No hay conexión con el servidor.
3306	0		Se necesita actualizar el cliente; es decir, Flash Media Rights Management Server (FMRMS) requiere un nuevo motor cliente para la administración de derechos digitales.
3307	0		Error interno genérico de la administración de derechos digitales.
3308	<i>Código de error de descifrado detallado</i>		Clave de licencia incorrecta.
3309	0		El contenido de vídeo Flash está dañado.
3310	0	<i>publisherID: applicationID</i>	El ID de la aplicación de visualización no coincide con un ID válido admitido por el editor del contenido. Es decir, el proveedor del contenido no admite la aplicación de visualización.
3311	0	<i>min=x:max=y</i>	La versión de la aplicación no coincide con lo especificado en la licencia.
3312	0		Error de verificación de la licencia asociada con el contenido cifrado; indica que el contenido puede estar dañado.

Código de error grave	ID de error menor	Detalles del error	Descripción
3313	0		No se pudo guardar en Microsafe la licencia asociada con el contenido cifrado.
3314	0		Error de verificación de la integridad del encabezado de FLV; indica que el contenido puede estar dañado.
3315			No se admite la reproducción remota del contenido protegido por DRM.

Creación de un controlador para el evento `DRMErrorEvent`

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 con cifrado DRM. Cuando una aplicación encuentra un error, normalmente lleva a cabo diversas tareas de limpieza, informa del error al usuario y ofrece opciones para resolver el problema.

```
private function drmErrorHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
function drmErrorHandler(event)
{
    air.trace(event.toString());
}
```

Capítulo 26: Opciones de inicio y cierre de aplicaciones

En esta sección se tratan las opciones y consideraciones para iniciar una aplicación de Adobe® AIR™ instalada, así como las opciones y consideraciones para cerrar una aplicación que está en curso.

Invocación de aplicaciones

Se invoca una aplicación de 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);
- visita una página web que contiene un logotipo o una aplicación de Flash que llama al método `com.adobe.air.AIR.launchApplication()` 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>`). Consulte [“Inicio desde el navegador de una aplicación de AIR instalada”](#) en la página 300.

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` sólo distribuye un evento `invoke` si la aplicación no está ya ejecutándose. Consulte [“Inicio desde el navegador de una aplicación de AIR instalada”](#) en la página 300.

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.

Sólo 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.

La aplicación puede controlar eventos `invoke` registrando un detector con su objeto `NativeApplication`:

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
air.NativeApplication.nativeApplication.addEventListener(air.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

Los argumentos de la línea de comandos asociados con la invocación de una aplicación de AIR se transmiten en el evento `invoke` distribuido por el objeto `NativeApplication`. La propiedad `InvokeEvent.arguments` 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 `InvokeEvent.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

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 recibidos y se muestran el directorio y los argumentos de la línea de comandos actuales.

Nota: para crear el siguiente ejemplo con Adobe® Flash® CS3, primero debe crear un archivo de Flash (Adobe AIR). En el panel de configuración de ActionScript 3.0 (Archivo > Publicación de configuración... > botón Configuración), escriba el nombre `InvokeEventLogExample` en el campo de la clase `Document`. Guarde el archivo FLA con el nombre `InvokeEventLogExample.fla`. A continuación, cree un archivo de ActionScript en la misma carpeta. Escriba el siguiente código en el archivo de ActionScript y guarde el archivo con el nombre `InvokeEventLogExample.as`.

```
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().toString();
```



```
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);
}
}
}
```

Inicio de aplicaciones al iniciar sesión

Para configurar una aplicación de AIR de modo que se inicie automáticamente cuando el usuario actual inicia sesión, configure `NativeApplication.nativeApplication.startAtLogin=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.

Nota: la aplicación no se inicia cuando arranca el ordenador, sino que lo hace al iniciar sesión el usuario. Esta configuración sólo se aplica al usuario actual. Además, 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.

Invocación desde el navegador

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 sólo se admite si el archivo descriptor de la aplicación define `allowBrowserInvocation` en `true`:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Para obtener más información sobre el archivo descriptor de la aplicación, consulte “[Configuración de las propiedades de una aplicación de AIR](#)” en la página 44.

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
<code>arguments</code>	Un conjunto de argumentos (cadenas) que se pasan a la aplicación.
<code>isHTTPS</code>	Indica si el contenido en el navegador utiliza el esquema <code>https</code> de la URL (<code>true</code>) o no (<code>false</code>).
<code>isUserEvent</code>	Indica si la invocación desde el navegador produjo un evento de usuario (hacer clic, por ejemplo). En AIR 1.0 siempre está definido en <code>true</code> ; AIR requiere un evento de usuario para la función de invocación desde el navegador.
<code>sandboxType</code>	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 <code>Security.sandboxType</code> y pueden ser uno de los siguientes: <ul style="list-style-type: none"> <code>Security.APPLICATION</code>: el contenido se encuentra en el entorno limitado de seguridad de la aplicación. <code>Security.LOCAL_TRUSTED</code>: el contenido se encuentra en el entorno limitado de seguridad local de confianza. <code>Security.LOCAL_WITH_FILE</code>: el contenido se encuentra en el entorno limitado de seguridad local con sistema de archivos. <code>Security.LOCAL_WITH_NETWORK</code>: el contenido se encuentra en el entorno limitado de seguridad local de red. <code>Security.REMOTE</code>: el contenido se encuentra en un dominio remoto (en la red).
<code>securityDomain</code>	El dominio de seguridad para el contenido del navegador, por ejemplo <code>"www.adobe.com"</code> o <code>"www.example.org"</code> . Esta propiedad sólo 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, ésta 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 ésta 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.

Para obtener más información sobre la invocación de una aplicación desde el navegador, consulte [“Inicio desde el navegador de una aplicación de AIR instalada”](#) en la página 300.

Cierre de una aplicación

La forma más rápida de cerrar una aplicación es llamar a `NativeApplication.nativeApplication.exit()`; esto funciona perfectamente cuando la aplicación no tiene datos que guardar ni recursos 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 sólo sucede en Mac OS, pues Windows no ofrece un comando de salir 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. Éste podría ser el caso, por ejemplo, si hay una ventana de documento con paletas de herramientas. Podría ser inconveniente (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).

Véase también

[“Configuración de las propiedades de una aplicación de AIR”](#) en la página 44

[“Presentación de interfaz de usuario de actualización personalizada de la aplicación”](#) en la página 310

Capítulo 27: Lectura de la configuración de una aplicación

En tiempo de ejecución se pueden obtener las propiedades del archivo descriptor de la aplicación y el ID del editor de una aplicación. Estos datos se definen en las propiedades `applicationDescriptor` y `publisherID` del objeto `NativeApplication`.

Lectura del archivo descriptor de la aplicación

Para leer como objeto XML el archivo descriptor de la aplicación que está en ejecución, obtenga la propiedad `applicationDescriptor` del objeto `NativeApplication`, como en el ejemplo siguiente:

```
var appXml:XML = NativeApplication.nativeApplication.applicationDescriptor;
```

A continuación se puede acceder a los datos del descriptor de la aplicación como objeto XML (E4X), como en el ejemplo siguiente:

```
var appXml:XML = NativeApplication.nativeApplication.applicationDescriptor;
var ns:Namespace = appXml.namespace();
var appId = appXml.ns::id[0];
var appVersion = appXml.ns::version[0];
var appName = appXml.ns::filename[0];
air.trace("appId:", appId);
air.trace("version:", appVersion);
air.trace("filename:", appName);
var xmlString = air.NativeApplication.nativeApplication.applicationDescriptor;
```

Para obtener más información, consulte [“Estructura del archivo descriptor de la aplicación”](#) en la página 44.

Obtención de los identificadores de la aplicación y del editor

Los ID de la aplicación y del editor identifican una aplicación de AIR de forma exclusiva. El ID de la aplicación se especifica en el elemento `<id>` del descriptor de la aplicación. El ID del editor se obtiene del certificado utilizado para firmar el paquete de instalación de AIR.

El ID de la aplicación puede leerse en la propiedad `id` del objeto `NativeApplication`, como se muestra en el siguiente código:

```
trace(NativeApplication.nativeApplication.applicationID);
```

El ID del editor puede leerse en la propiedad `publisherID` del objeto `NativeApplication`:

```
trace(NativeApplication.nativeApplication.publisherID);
```

Nota: cuando se ejecuta una aplicación de AIR con ADL, no tendrá ID de editor a menos que se le asigne uno temporalmente utilizando el indicador `-pubID` en la línea de comandos de ADL.

El ID del editor para una aplicación instalada también aparece en el archivo `META-INF/AIR/publisherid` del directorio de instalación de la aplicación.

Para obtener más información, consulte “[Identificador del editor de AIR](#)” en la página 302.

Capítulo 28: Trabajo con información sobre el motor de ejecución y el sistema operativo

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™.

Gestión de asociaciones con archivos

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 ésta 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
<code>isSetAsDefaultApplication()</code>	Devuelve un valor de "true" si la aplicación de AIR está asociada con el tipo de archivo especificado.
<code>setAsDefaultApplication()</code>	Crea la asociación entre la aplicación de AIR y la acción de abrir del tipo de archivo.
<code>removeAsDefaultApplication()</code>	Elimina la asociación entre la aplicación de AIR y el tipo de archivo.
<code>getDefaultApplication()</code>	Notifica la ruta de la aplicación que está asociada con el tipo de archivo.

AIR sólo 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, ésta emitirá una excepción del motor de ejecución.

Para obtener información sobre la declaración de tipos de archivos en el descriptor de la aplicación, consulte [“Declaración de asociaciones con tipos de archivos”](#) en la página 51.

Obtención de la versión y el nivel de revisión del motor de ejecución

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

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 23.

Seguimiento de la presencia de usuarios

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: sólo se distribuye un evento `userIdle` entre dos eventos `userPresent` cualesquiera.

Capítulo 29: Supervisión de la conectividad de la red

Adobe® AIR™ proporciona los medios para comprobar si hay cambios en la conectividad de la red del ordenador en el que hay instalada una aplicación de AIR. Esta información es de utilidad si una aplicación usa datos que se obtienen de la red. Asimismo, una aplicación puede comprobar la disponibilidad de un servicio de red.

Detección de cambios de conectividad de la red

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. El objeto `NativeApplication` de la aplicación distribuye el evento de cambio en la red. Para reaccionar a este evento, añada un detector:

```
NativeApplication.nativeApplication.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

Defina también una función de controlador de eventos:

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

El evento `Event.NETWORK_CHANGE` 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. Con esta finalidad los marcos de supervisión del servicio en el paquete `air.net` proporcionan aplicaciones de AIR con un medio basado en eventos de responder a los cambios en la conectividad de la red a un host en particular.

Nota: el marco de supervisión del servicio detecta si un servidor responde a una petición de forma aceptable. Esto no garantiza la máxima 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 sólo ofrecen un diagnóstico parcial de la conectividad de la red.

Aspectos básicos de la supervisión del servicio

El marco de supervisión del servicio, que es independiente de la arquitectura de AIR, reside en el archivo `servicemonitor.swc`. Para poder utilizar el marco hay que incluir el archivo `servicemonitor.swc` en el proceso de creación.

Importante: Para utilizar estas clases en ActionScript, 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.*;
```

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.

Detección de la conectividad de HTTP

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.adobe.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

Detección de la conectividad de sockets

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.

De forma similar al ejemplo para `URLMonitor`, el siguiente código incluye una instancia de la clase `SocketMonitor` para detectar los cambios en la conectividad de una conexión de socket en el puerto 6667, un puerto común para IRC:

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.adobe.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

Capítulo 30: Peticiones de URL y redes

La nueva funcionalidad de Adobe AIR que especifica peticiones de URL no está disponible en el contenido SWF que se ejecuta en el navegador. Esta funcionalidad sólo está disponible en el contenido del entorno limitado de seguridad de la aplicación. En esta sección se describen las características de URLRequest en tiempo de ejecución y se aborda el contenido de AIR y los cambios en la API de redes.

Para obtener más información sobre el uso de las redes de ActionScript™ 3.0 y las funciones de comunicación, consulte el manual *Programación con ActionScript 3.0* (suministrado con Adobe® Flash® CS3 y Adobe® Flex™ Builder™ 3).

Utilización de la clase URLRequest

La clase URLRequest permite definir la cadena URL de forma realmente sencilla. AIR añade algunas propiedades a la clase URLRequest que sólo están disponibles en el contenido de AIR que se ejecuta en el entorno limitado de seguridad de la aplicación. El contenido del motor de ejecución puede definir las direcciones URL mediante los nuevos esquemas de URL (además de los esquemas estándar como `file` y `http`).

Propiedades de URLRequest

La clase URLRequest incluye las siguientes propiedades, sólo disponibles en el contenido del entorno limitado de seguridad de la aplicación de AIR:

Propiedad	Descripción
<code>followRedirects</code>	Especifica si se siguen las redirecciones (<code>true</code> , valor predeterminado) o no (<code>false</code>). Sólo se admite en tiempo de ejecución.
<code>manageCookies</code>	Especifica si la cola del protocolo HTTP debe gestionar las cookies (<code>true</code> , valor predeterminado) o no (<code>false</code>) en esta petición. Sólo se admite en tiempo de ejecución.
<code>authenticate</code>	Especifica si deben manejarse peticiones de autenticación (<code>true</code>) en esta petición. Sólo se admite en tiempo de ejecución. El comportamiento predeterminado es autenticar peticiones. Esto puede provocar la aparición de un cuadro de diálogo si el servidor requiere credenciales. También puede definir el nombre de usuario y la contraseña. Consulte “ Configuración de valores predeterminados de URLRequest ” en la página 287.
<code>cacheResponse</code>	Especifica si los datos correctos de respuesta de esta petición se deben guardar en la memoria caché. Sólo se admite en tiempo de ejecución. El valor predeterminado es guardar la respuesta en la memoria caché (<code>true</code>).
<code>useCache</code>	Especifica si se debe consultar la memoria caché local antes de que el objeto URLRequest tome los datos. Sólo se admite en tiempo de ejecución. El comportamiento predeterminado (<code>true</code>) es utilizar la versión guardada en la memoria local, si está disponible.
<code>userAgent</code>	Especifica la cadena de agente de usuario que se utiliza en la petición HTTP.

Se pueden establecer las siguientes propiedades de un objeto URLRequest por su contenido en cualquier entorno limitado (no sólo en el entorno limitado de seguridad de la aplicación de AIR):

Propiedad	Descripción
contentType	El tipo de contenido MIME de todos los datos enviados con la petición de URL.
data	Un objeto que contiene datos que se van a transmitir con la petición de URL.
digest	"Compendio" seguro en un archivo guardado en la memoria caché para realizar un seguimiento de la caché de Adobe® Flash® Player.
method	Controla el método de petición HTTP, por ejemplo, operaciones 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 <code>method</code> . Se permite cualquier palabra clave HTTP, aunque "GET" es el método predeterminado. Consulte "Seguridad en AIR" en la página 23.)
requestHeaders	El conjunto de encabezados de petición HTTP que se añadirán a la petición HTTP.
url	Especifica la dirección URL que se va a solicitar.

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 217.

Configuración de valores predeterminados de `URLRequest`

La clase `URLRequestDefaults` permite definir la configuración predeterminada de los objetos `URLRequest`. Por ejemplo, el siguiente código establece los valores predeterminados de las propiedades `manageCookies` y `useCache`:

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
air.URLRequestDefaults.manageCookies = false;
air.URLRequestDefaults.useCache = false;
```

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 sólo se utilizan si las requiere el servidor. Si el usuario ya se ha autenticado (por ejemplo, mediante el cuadro de diálogo de autenticación), no es posible modificar el usuario autenticado llamando al método `setLoginCredentialsForHost()`.

Por ejemplo, el siguiente código establece el nombre de usuario y la contraseña predeterminados para `www.example.com`:

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
air.URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

Cada propiedad de la configuración de `URLRequestDefaults` se aplica únicamente al dominio de la aplicación del contenido que configura la propiedad. Sin embargo, el método `setLoginCredentialsForHost()` se aplica al contenido de todos los dominios de la aplicación dentro una aplicación de AIR. De este modo, una aplicación puede conectarse a un host y cargar *todo* el contenido de la aplicación con las credenciales especificadas.

Para obtener más información, consulte la clase `URLRequestDefaults` en el manual [Referencia del lenguaje y componentes ActionScript 3.0](#) (http://www.adobe.com/go/learn_air_aslr_es).

Utilización de esquemas de URL de AIR en direcciones URL

Al definir direcciones URL en cualquier entorno limitado de seguridad de AIR, se ponen a disposición del usuario los siguientes esquemas estándar de URL (entre otros):

http: y https:

Utilice estos esquemas del mismo modo que lo haría en un navegador Web.

file:

Utilice este esquema para especificar una ruta relativa a la raíz del sistema de archivos. Por ejemplo:

```
file:///c:/AIR Test/test.txt
```

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 este esquema para especificar una ruta relativa al directorio raíz de la aplicación instalada (el directorio que contiene el archivo descriptor 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 ejecuta en la aplicación de ADL, el directorio de recursos de la aplicación se establece como el directorio que contiene el archivo descriptor de la aplicación.

app-storage:

Utilice este esquema para especificar una ruta relativa al directorio de almacenamiento de la aplicación. En cada aplicación instalada, AIR define un directorio de almacenamiento exclusivo de la aplicación para cada usuario. Es una ubicación útil para guardar datos específicos de cada aplicación. Por ejemplo, la ruta siguiente apunta a un archivo `prefs.xml` en un subdirectorio de configuración del directorio de almacenamiento de la aplicación:

```
app-storage:/settings/prefs.xml
```

La ubicación del directorio de almacenamiento de la aplicación se basa en el nombre de usuario, el ID de aplicación y el ID de editor:

- Mac OS - En:

```
/Usuarios/nombre de usuario/Biblioteca/Preferences/applicationID.publisherID/Local Store/
```

Por ejemplo:

```
/Users/babbage/Library/Preferences/com.example.TestApp.02D88EEED35F84C264A183921344EEA353A629FD.1/Local Store
```

- Windows - En el directorio Documents and Settings, en:

```
nombre de usuario/Application Data/applicationID.publisherID/Local Store/
```

Por ejemplo:

```
C:\Documents and Settings\babbage\Application Data\com.example.TestApp.02D88EEED35F84C264A183921344EEA353A629FD.1\Local Store
```

La dirección URL (y la propiedad `url`) de un objeto `File` creado con `File.applicationStorageDirectory` utiliza el esquema de URL `app-storage`, tal como se indica a continuación:

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
var dir = air.File.applicationStorageDirectory;
dir = dir.resolvePath("prefs.xml");
air.trace(dir.url); // app-storage:/preferences
```

Utilización de esquemas de URL en AIR

Puede elegir un objeto URLRequest que utilice cualquiera de estos esquemas de URL 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, sólo se pueden utilizar estos esquemas de URL 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 23.

Esquemas de URL no permitidos

Algunas de las siguientes API permiten lanzar contenido en un navegador Web. Por motivos de seguridad, algunos esquemas de URL 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, consulte [“Apertura de una dirección URL en el navegador Web predeterminado del sistema”](#) en la página 289.

Cambios en la clase URLStream

La clase URLStream proporciona acceso de bajo nivel para descargar datos desde direcciones URL. En tiempo de ejecución, la clase URLStream incluye un nuevo evento: `httpResponseStatus`. Al contrario que `HttpStatus`, el evento `httpResponseStatus` se envía antes que cualquier dato de respuesta. El evento `httpResponseStatus` (definido en 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).

Apertura de una dirección URL en el navegador Web predeterminado del sistema

Puede utilizar la función `navigateToURL()` para abrir una dirección URL en el navegador Web predeterminado del sistema. Para el objeto `URLRequest` que transfiere como parámetro `request` de esta función, sólo se utiliza la propiedad `url`.

```
var url = "http://www.adobe.com";
var urlReq = new air.URLRequest(url);
air.navigateToURL(urlReq);
```

Cuando se utiliza la función `navigateToURL()`, se permiten esquemas de URL 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 URL 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 23.)

Entorno limitado de la aplicación

Se permiten los siguientes esquemas. Utilice estos esquemas del mismo modo que lo haría en un navegador Web.

- http:
- https:
- file:
- mailto: - AIR dirige estas peticiones a la aplicación de correo registrada en el sistema
- app:
- app-storage:

El resto de esquemas de URL no están permitidos.

Entorno limitado remoto

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 URL 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 URL 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 URL 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 URL no están permitidos.

Capítulo 31: Distribución, instalación y ejecución de aplicaciones de AIR

Las aplicaciones de AIR se distribuyen como un solo archivo de instalación de AIR que contiene el código de la aplicación y todos los componentes. Este archivo puede distribuirse por cualquiera de las vías tradicionales, por ejemplo descargándolo, por correo electrónico o en soportes físicos como CD-ROM. El usuario pueden instalar la aplicación haciendo doble clic en el archivo de AIR. Se puede utilizar la función de *instalación integrada*, que permite al usuario instalar la aplicación de AIR (y Adobe® AIR™, si es preciso) haciendo clic en un solo enlace en una página Web.

Antes de poder distribuirlo, el archivo de instalación de AIR se debe empaquetar y firmar con un certificado de firma de código y una clave privada. La firma digital del archivo de instalación ofrece la seguridad de que la aplicación no ha sido modificada desde que se firmó. Además, si una autoridad de certificación fidedigna como Verisign o Thawte emite un certificado digital, los usuarios de su aplicación podrán confirmar su identidad como editor y firmante. El archivo de AIR se firma cuando se empaqueta la aplicación junto con la herramienta AIR Developer Tool (ADT).

Para obtener información sobre cómo empaquetar una aplicación en un archivo de AIR con la actualización de AIR para Flash, consulte [“Creación de archivos de aplicación e instalador de AIR”](#) en la página 15.

Para obtener información sobre cómo empaquetar una aplicación en un archivo de AIR con el SDK de Adobe® AIR™, consulte [“Empaquetado de archivos de instalación de AIR con AIR Developer Tool \(ADT\)”](#) en la página 326.

Instalación y ejecución de una aplicación de AIR desde el escritorio

Se puede simplemente enviar el archivo de AIR al destinatario. Por ejemplo: podría enviar el archivo de AIR como adjunto a un correo electrónico o como vínculo en una página Web.

Una vez que descargó la aplicación de AIR, el usuario sigue estas instrucciones para instalarla:

- 1 Haga doble clic en el archivo de AIR.

Adobe AIR ya debe estar instalado en el ordenador.

- 2 En la ventana Instalación, deje seleccionada la configuración predeterminada y haga clic en Continuar.

En Windows AIR realiza automáticamente lo siguiente:

- Instala la aplicación en el directorio Archivos de programa
- Crea un acceso directo para la aplicación en el escritorio
- Crea un acceso directo en el menú Inicio
- Añade una entrada para la aplicación en Agregar o quitar programas, en el Panel de control

En Mac OS, la aplicación se añade de forma predeterminada al directorio Aplicaciones.

Si la aplicación ya está instalada, el instalador ofrece al usuario la opción de abrir la versión existente de la misma o actualizarla a la versión del archivo de AIR descargado. El instalador identifica la aplicación utilizando para ello el ID de la aplicación y el ID del editor que figuran en el archivo de AIR.

- 3 Una vez concluida la instalación, haga clic en Finalizar.

En Mac OS, para instalar una versión actualizada de una aplicación el usuario debe contar con privilegios del sistema adecuados para instalar programas en el directorio de aplicaciones. En Windows, el usuario debe contar con privilegios de administrador.

Una aplicación también puede instalar una nueva versión mediante ActionScript o JavaScript. Para obtener más información, consulte [“Actualización de aplicaciones de AIR”](#) en la página 308.

Una vez instalada la aplicación de AIR, basta con que el usuario haga doble clic en la aplicación para ejecutarla, al igual que con cualquier otra aplicación del escritorio.

- En Windows, haga doble clic en el icono de la aplicación (que está instalada en el escritorio o en una carpeta) o seleccione la aplicación en el menú Inicio.
- En Mac OS, haga doble clic en la aplicación en la carpeta en la que se instaló. El directorio de instalación predeterminado es /Aplicaciones.

La función de *instalación integrada* permite al usuario instalar una aplicación de AIR haciendo clic en un vínculo en una página Web. La función de *invocación desde el navegador* permite al usuario ejecutar una aplicación de AIR instalada haciendo clic en un vínculo en una página Web. Estas funciones se describen en la siguiente sección.

Instalación y ejecución de aplicaciones de AIR desde una página Web

La función de instalación integrada permite incorporar un archivo SWF en una página Web mediante la cual el usuario puede instalar una aplicación de AIR desde el navegador. Si no está instalado el motor de ejecución, lo instalará la función de instalación integrada. La función de instalación integrada permite al usuario instalar la aplicación de AIR sin necesidad de guardar el archivo de AIR en el ordenador. El SDK de AIR incluye un archivo `badge.swf` que permite utilizar fácilmente la función de instalación integrada. Para obtener más información, consulte [“Utilización del archivo badge.swf para instalar una aplicación de AIR”](#) en la página 294.

Personalización del archivo `badge.swf` de instalación integrada

Además de utilizar el archivo `badge.swf` suministrado con el SDK, puede crear su propio archivo SWF para usarlo en una página del navegador. El archivo SWF personalizado puede interactuar con el motor de ejecución de cualquiera de las formas siguientes:

- Puede instalar una aplicación de AIR. Consulte [“Instalación de una aplicación de AIR desde el navegador”](#) en la página 299.
- Puede comprobar si hay instalada una aplicación de AIR en particular. Consulte [“Cómo comprobar desde una página Web si una aplicación de AIR está instalada”](#) en la página 298.
- Puede comprobar si está instalado el motor de ejecución. Consulte [“Cómo comprobar si está instalado el motor de ejecución”](#) en la página 297.
- Puede iniciar una aplicación de AIR instalada en el sistema del usuario. Consulte [“Inicio desde el navegador de una aplicación de AIR instalada”](#) en la página 300.

Estas capacidades se proporcionan al llamar a las API de un archivo SWF alojado en `adobe.com`: `air.swf`. En esta sección se describe cómo utilizar y personalizar el archivo `badge.swf` y cómo llamar a las API de `air.swf` desde su propio archivo SWF.

Además, un archivo SWF que se ejecuta en el navegador puede comunicarse con una aplicación de AIR en curso utilizando la clase `LocalConnection`. Para obtener más información, consulte [Comunicación entre aplicaciones](#).

Importante: Las funciones que se describen en esta sección (y las API del archivo `air.swf`) requieren que el usuario tenga Adobe® Flash® Player 9 actualización 3 instalado en el navegador Web. Se puede escribir código para comprobar la versión instalada de Flash Player y proveer una interfaz alternativa para el usuario si no está instalada la versión de Flash Player que se requiere. Por ejemplo: si hay una versión anterior de Flash Player instalada, podría proporcionar un vínculo a la versión de descarga del archivo de AIR (en lugar de utilizar el archivo `badge.swf` o la API `air.swf` para instalar una aplicación).

Utilización del archivo `badge.swf` para instalar una aplicación de AIR

El SDK de AIR incluye un archivo `badge.swf` que permite utilizar fácilmente la función de instalación integrada. El archivo `badge.swf` puede instalar el motor de ejecución y una aplicación de AIR desde un vínculo en una página Web. El archivo `badge.swf` y su código fuente se le proporcionan para que los distribuya a través de su sitio Web.

Las instrucciones de esta sección facilitan información sobre la configuración de los parámetros del archivo `badge.swf` suministrado por Adobe. También proporcionamos el código fuente para el archivo `badge.swf`, que podrá usted personalizar.

Incorporación del archivo `badge.swf` en una página Web

- 1 Localice los siguientes archivos, suministrados en el directorio `samples/badge` del SDK de AIR, y añádalos a su servidor Web.
 - `badge.swf`
 - `default_badge.html`
 - `AC_RunActiveContent.js`
- 2 Abra la página `default_badge.html` en un editor de textos.
- 3 En la página `default_badge.html`, en la función de JavaScript `AC_FL_RunContent()`, ajuste las definiciones del parámetro `FlashVars` para lo siguiente:

Parámetro	Descripción
<code>appname</code>	El nombre de la aplicación que muestra el archivo SWF si no está instalado el motor de ejecución.
<code>appurl</code>	(Obligatorio). La URL del archivo de AIR a descargar. Hay que utilizar una URL absoluta (y no una relativa).
<code>airversion</code>	(Obligatorio). Para la versión 1.0 del motor de ejecución, defina esto en 1.0.
<code>imageurl</code>	La URL de la imagen (opcional) a mostrar como logotipo.
<code>buttoncolor</code>	El color del botón de descargar (especificado como valor hexadecimal; por ejemplo, <code>FFCC00</code>).
<code>messagecolor</code>	El color del mensaje de texto que aparece debajo del botón si no está instalado el motor de ejecución (especificado como valor hexadecimal; por ejemplo, <code>FFCC00</code>).

- 4 El tamaño mínimo del archivo `badge.swf` es de 217 píxeles de anchura por 180 píxeles de altura. Ajuste los valores de los parámetros `width` y `height` de la función `AC_FL_RunContent()` de acuerdo con sus necesidades.
- 5 Cambie el nombre del archivo `default_badge.html` y ajuste su código (o inclúyalo en otra página HTML) para adaptarlo a sus necesidades.

También se puede editar y recompilar el archivo `badge.swf`. Para obtener más información, consulte “[Modificación del archivo `badge.swf`](#)” en la página 295.

Instalación de la aplicación de AIR desde un vínculo de instalación integrada en una página Web

Una vez que haya añadido a una página el vínculo de instalación integrada, el usuario podrá instalar la aplicación de AIR con sólo hacer clic en el vínculo del archivo SWF.

- 1 Vaya a la página HTML en un navegador Web que tenga instalado Flash Player (versión 9 actualización 3 o posterior).
- 2 En la página Web, haga clic en el vínculo del archivo badge.swf.
 - Si ha instalado el motor de ejecución, vaya al paso siguiente.
 - Si no ha instalado el motor de ejecución, aparece un cuadro de diálogo que le pregunta si desea instalarlo. Instale el motor de ejecución (consulte “[Instalación de Adobe AIR](#)” en la página 1) y continúe con el siguiente paso.
- 3 En la ventana Instalación, deje seleccionada la configuración predeterminada y haga clic en Continuar.

En un ordenador con Windows, AIR realiza automáticamente lo siguiente:

- Instala la aplicación en c:\Archivos de programa\
- Crea un acceso directo para la aplicación en el escritorio
- Crea un acceso directo en el menú Inicio
- Añade una entrada para la aplicación en Agregar o quitar programas, en el Panel de control

En Mac OS el instalador añade la aplicación al directorio de aplicaciones (por ejemplo, en el directorio /Aplicaciones de Mac OS).

- 4 Seleccione las opciones que desee y haga clic en el botón Instalar.
- 5 Una vez concluida la instalación, haga clic en Finalizar.

Modificación del archivo badge.swf

El SDK de AIR proporciona los archivos de origen para el archivo badge.swf. Estos archivos están incluidos en la carpeta src del SDK:

Archivos de origen	Descripción
badge fla	El archivo de origen de Flash CS3 o CS4 se utiliza para compilar el archivo badge.swf. El archivo badge fla se compila en un archivo SWF 9 (que se puede cargar en Flash Player).
AIRBadge.as	Una clase de ActionScript 3.0 que define la clase de base que se utiliza en el archivo badge fla.

Puede utilizar Flash CS3 o CS4 para rediseñar la interfaz visual del archivo badge.swf.

La función constructora `AIRBadge()`, definida en la clase `AIRBadge`, carga el archivo `air.swf` alojado en <http://airdownload.adobe.com/air/browserapi/air.swf>. El archivo `air.swf` incluye código para utilizar la función de instalación integrada.

El método `onInit()` (en la clase `AIRBadge`) se invoca una vez satisfactoriamente cargado el archivo `air.swf`:

```

private function onInit(e:Event):void {
    _air = e.target.content;
    switch (_air.getStatus()) {
        case "installed" :
            root.statusMessage.text = "";
            break;
        case "available" :
            if (_appName && _appName.length > 0) {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                    + _messageColor + "'>In order to run " + _appName +
                    ", this installer will also set up Adobe® AIR™.</font></p>";
            } else {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                    + _messageColor + "'>In order to run this application, "
                    + "this installer will also set up Adobe® AIR™.</font></p>";
            }
            break;
        case "unavailable" :
            root.statusMessage.htmlText = "<p align='center'><font color='#"
                + _messageColor
                + "'>Adobe® AIR™ is not available for your system.</font></p>";
            root.buttonBg_mc.enabled = false;
            break;
    }
}
}

```

El código define la variable global `_air` en la clase principal del archivo `air.swf`. Esta clase incluye los siguientes métodos públicos, a los que tiene acceso el archivo `badge.swf` para llamar a la función de instalación integrada:

Método	Descripción
<code>getStatus()</code>	Determina si el motor de ejecución está instalado (o puede instalarse) en el ordenador. Para obtener más información, consulte "Cómo comprobar si está instalado el motor de ejecución" en la página 297.
<code>installApplication()</code>	<p>Instala la aplicación especificada en el equipo del usuario. Para obtener más información, consulte "Instalación de una aplicación de AIR desde el navegador" en la página 299.</p> <ul style="list-style-type: none"> <code>url</code>: una cadena que define la URL. Hay que utilizar una ruta de URL absoluta (y no una relativa). <code>runtimeVersion</code>: una cadena que indica la versión del motor de ejecución (por ejemplo: "1.0.M6") que requiere la aplicación a instalarse. <code>arguments</code>: argumentos a pasarse a la aplicación si se inicia en cuanto se haya instalado. La aplicación se inicia en el momento de instalarla si el elemento <code>allowBrowserInvocation</code> está definido en <code>true</code> en el archivo descriptor de la aplicación. (Para obtener más información sobre el archivo descriptor de la aplicación, consulte "Configuración de las propiedades de una aplicación de AIR" en la página 44). Si la aplicación se inicia a resultados de una instalación integrada desde el navegador (habiendo el usuario seleccionado iniciarla en cuanto se instalara), el objeto <code>NativeApplication</code> de la aplicación distribuye un objeto <code>BrowserInvokeEvent</code> sólo si se pasan argumentos. Tenga en cuenta las consecuencias para la seguridad que pueden tener los datos que pase a la aplicación. Para obtener más información, consulte "Inicio desde el navegador de una aplicación de AIR instalada" en la página 300.

Los valores para `url` y `runtimeVersion` se pasan al archivo SWF a través de las opciones de FlashVars en la página HTML contenedora.

Si la aplicación se inicia automáticamente al instalarla, se puede utilizar la comunicación por `LocalConnection` para que la aplicación instalada se ponga en contacto con el archivo `badge.swf` al invocarse. Para obtener más información, consulte [Comunicación entre aplicaciones](#).

También se puede llamar al método `getApplicationVersion()` del archivo `air.swf` para comprobar si una aplicación está instalada. Se puede llamar a este método antes del proceso de instalación de la aplicación o una vez iniciada la instalación. Para obtener más información, consulte “[Cómo comprobar desde una página Web si una aplicación de AIR está instalada](#)” en la página 298.

Carga del archivo `air.swf`

Puede crear su propio archivo SWF que utilice las API del archivo `air.swf` para interactuar con el motor de ejecución y las aplicaciones de AIR desde una página Web en un navegador. El archivo `air.swf` está alojado en <http://airdownload.adobe.com/air/browserapi/air.swf>. Para hacer referencia a las API de `air.swf` desde el archivo SWF, cargue el archivo `air.swf` en el mismo dominio de la aplicación que el archivo SWF. El código siguiente muestra un ejemplo de cargar el archivo `air.swf` en el dominio de la aplicación del archivo SWF de carga:

```
var airSWF:Object; // This is the reference to the main class of air.swf
var airSWFLoader:Loader = new Loader(); // Used to load the SWF
var loaderContext:LoaderContext = new LoaderContext();
    // Used to set the application domain

loaderContext.applicationDomain = ApplicationDomain.currentDomain;

airSWFLoader.contentLoaderInfo.addEventListener(Event.INIT, onInit);
airSWFLoader.load(new URLRequest("http://airdownload.adobe.com/air/browserapi/air.swf"),
    loaderContext);

function onInit(e:Event):void
{
    airSWF = e.target.content;
}
```

Una vez cargado el archivo `air.swf` (cuando el objeto `contentLoaderInfo` de `Loader` distribuye el evento `init`), se puede llamar a cualquiera de las API de `air.swf`. Estas API se describen en las siguientes secciones:

- “[Cómo comprobar si está instalado el motor de ejecución](#)” en la página 297
- “[Cómo comprobar desde una página Web si una aplicación de AIR está instalada](#)” en la página 298
- “[Instalación de una aplicación de AIR desde el navegador](#)” en la página 299
- “[Inicio desde el navegador de una aplicación de AIR instalada](#)” en la página 300

Nota: El archivo `badge.swf` suministrado con el SDK de AIR carga automáticamente el archivo `air.swf`. Consulte “[Utilización del archivo `badge.swf` para instalar una aplicación de AIR](#)” en la página 294. Las instrucciones que aparecen en esta sección son para crear su propio archivo SWF que cargue el archivo `air.swf`.

Cómo comprobar si está instalado el motor de ejecución

Un archivo SWF puede comprobar si el motor de ejecución está instalado, llamando al método `getStatus()` en el archivo `air.swf` cargado desde <http://airdownload.adobe.com/air/browserapi/air.swf>. Para obtener más información, consulte “[Carga del archivo `air.swf`](#)” en la página 297.

Una vez cargado el archivo `air.swf`, el archivo SWF puede llamar al método `getStatus()` del archivo `air.swf` como en el ejemplo siguiente:

```
var status:String = airSWF.getStatus();
```

El método `getStatus()` devuelve uno de los siguientes valores de cadena, basado en el estado del motor de ejecución en el ordenador:

Valor de la cadena	Descripción
"available"	El motor de ejecución puede instalarse en este ordenador pero ahora no está instalado.
"unavailable"	El motor de ejecución no puede instalarse en este ordenador.
"installed"	El motor de ejecución está instalado en este ordenador.

El método `getStatus()` emite un error si no está instalada la versión necesaria de Flash Player (versión 9 actualización 3) en el navegador.

Cómo comprobar desde una página Web si una aplicación de AIR está instalada

Un archivo SWF puede comprobar si una aplicación de AIR (con ID de la aplicación e ID del editor que coincidan) está instalada, llamando al método `getApplicationVersion()` en el archivo `air.swf` cargado desde <http://airdownload.adobe.com/air/browserapi/air.swf>. Para obtener más información, consulte “Carga del archivo `air.swf`” en la página 297.

Una vez cargado el archivo `air.swf`, el archivo SWF puede llamar al método `getApplicationVersion()` del archivo `air.swf` como en el ejemplo siguiente:

```
var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EED35F84C264A183921344EEA353A629FD.1";
airSWF.getApplicationVersion(appID, pubID, versionDetectCallback);

function versionDetectCallback(version:String):void
{
    if (version == null)
    {
        trace("Not installed.");
        // Take appropriate actions. For instance, present the user with
        // an option to install the application.
    }
    else
    {
        trace("Version", version, "installed.");
        // Take appropriate actions. For instance, enable the
        // user interface to launch the application.
    }
}
```

El método `getApplicationVersion()` utiliza los siguientes parámetros:

Parámetros	Descripción
appID	El ID de la aplicación. Para obtener más información, consulte “Definición de los datos básicos de la aplicación” en la página 46.
pubID	El ID del editor de la aplicación. Para obtener más información, consulte “Identificador del editor de AIR” en la página 302.
callback	Una función callback que cumple la función de controlador. El método <code>getApplicationVersion()</code> funciona de modo asíncrono, y al detectar la versión instalada (o la falta de una), se invoca este método callback. La definición del método callback debe incluir un parámetro, una cadena de caracteres, que se establece como la cadena de la versión de la aplicación instalada. Si la aplicación no está instalada, se pasa un valor nulo a la función, como se muestra en el ejemplo de código anterior.

El método `getApplicationVersion()` emite un error si no está instalada la versión necesaria de Flash Player (versión 9 actualización 3) en el navegador.

Instalación de una aplicación de AIR desde el navegador

Un archivo SWF puede instalar una aplicación de AIR, llamando al método `installApplication()` en el archivo `air.swf` cargado desde `http://airdownload.adobe.com/air/browserapi/air.swf`. Para obtener más información, consulte [“Carga del archivo air.swf”](#) en la página 297.

Una vez cargado el archivo `air.swf`, el archivo SWF puede llamar al método `installApplication()` del archivo `air.swf` como en el código siguiente:

```
var url:String = "http://www.example.com/myApplication.air";  
var runtimeVersion:String = "1.0";  
var arguments:Array = ["launchFromBrowser"]; // Optional  
airSWF.installApplication(url, runtimeVersion, arguments);
```

El método `installApplication()` instala la aplicación especificada en el equipo del usuario. Este método utiliza los siguientes parámetros:

Parámetro	Descripción
<code>url</code>	Una cadena de caracteres que define la URL del archivo de AIR a instalar. Hay que utilizar una ruta de URL absoluta (y no una relativa).
<code>runtimeVersion</code>	Una cadena que indica la versión del motor de ejecución (por ejemplo: "1.0") que requiere la aplicación a instalarse.
<code>arguments</code>	Un conjunto de argumentos a pasarse a la aplicación si se inicia en cuanto se haya instalado. La aplicación se inicia en el momento de instalarla si el elemento <code>allowBrowserInvocation</code> está definido en <code>true</code> en el archivo descriptor de la aplicación. (Para obtener más información sobre el archivo descriptor de la aplicación, consulte “Configuración de las propiedades de una aplicación de AIR” en la página 44). Si la aplicación se inicia a resultas de una instalación integrada desde el navegador (habiendo el usuario seleccionado iniciarla en cuanto se instalara), el objeto <code>NativeApplication</code> de la aplicación distribuye un objeto <code>BrowserInvokeEvent</code> sólo si se pasan argumentos. Para obtener más información, consulte “Inicio desde el navegador de una aplicación de AIR instalada” en la página 300.

El método `installApplication()` sólo funciona cuando se le llama en el controlador de eventos para un evento de usuario, por ejemplo al hacer clic con el ratón.

El método `installApplication()` emite un error si no está instalada la versión necesaria de Flash Player (versión 9 actualización 3) en el navegador.

En Mac OS, para instalar una versión actualizada de una aplicación el usuario debe contar con privilegios del sistema adecuados para instalar programas en el directorio de aplicaciones (y privilegios de administrador si la aplicación actualiza el motor de ejecución). En Windows, el usuario debe contar con privilegios de administrador.

También se puede llamar al método `getApplicationVersion()` del archivo `air.swf` para comprobar si una aplicación ya está instalada. Se puede llamar a este método antes de que empiece el proceso de instalación de la aplicación o una vez iniciada la instalación. Para obtener más información, consulte [“Cómo comprobar desde una página Web si una aplicación de AIR está instalada”](#) en la página 298. Una vez en ejecución la aplicación, ésta puede comunicarse con el contenido SWF en el navegador utilizando la clase `LocalConnection`. Para obtener más información, consulte [Comunicación entre aplicaciones](#).

Inicio desde el navegador de una aplicación de AIR instalada

Para utilizar la función de invocación desde el navegador (lo que permite iniciar la aplicación desde el navegador), el archivo descriptor de la aplicación en cuestión debe incluir la siguiente definición:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Para obtener más información sobre el archivo descriptor de la aplicación, consulte [“Configuración de las propiedades de una aplicación de AIR”](#) en la página 44.

Un archivo SWF en el navegador puede iniciar una aplicación de AIR, llamando al método `launchApplication()` en el archivo `air.swf` cargado desde <http://airdownload.adobe.com/air/browserapi/air.swf>. Para obtener más información, consulte [“Carga del archivo air.swf”](#) en la página 297.

Una vez cargado el archivo `air.swf`, el archivo SWF puede llamar al método `launchApplication()` del archivo `air.swf` como en el código siguiente:

```
var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EED35F84C264A183921344EEA353A629FD.1";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.launchApplication(appID, pubID, arguments);
```

El método `launchApplication()` se define en el nivel superior del archivo `air.swf` (que se carga en el dominio de aplicación de la interfaz de usuario del archivo SWF). Al llamar a este método, AIR inicia la aplicación especificada (si está instalada y se permite la invocación desde el navegador mediante la opción `allowBrowserInvocation` del archivo descriptor de la aplicación). El método utiliza los siguientes parámetros:

Parámetro	Descripción
appID	El ID de la aplicación que se va a iniciar. Para obtener más información, consulte “Definición de los datos básicos de la aplicación” en la página 46.
pubID	El ID del editor de la aplicación que se va a iniciar. Para obtener más información, consulte “Identificador del editor de AIR” en la página 302.
arguments	Un conjunto de argumentos que se pasan a la aplicación. El objeto <code>NativeApplication</code> de la aplicación distribuye un evento <code>BrowserInvokeEvent</code> que tiene una propiedad "arguments" definida en este conjunto..

El método `launchApplication()` sólo funciona cuando se le llama en el controlador de eventos para un evento de usuario, por ejemplo al hacer clic con el ratón.

El método `launchApplication()` emite un error si no está instalada la versión necesaria de Flash Player (versión 9 actualización 3) en el navegador.

Si el elemento `allowBrowserInvocation` está definido en `false` en el archivo descriptor de la aplicación, la llamada al método `launchApplication()` no surtirá efecto.

Antes de presentar la interfaz de usuario para iniciar la aplicación, puede ser conveniente llamar al método `getApplicationVersion()` en el archivo `air.swf`. Para obtener más información, consulte [“Cómo comprobar desde una página Web si una aplicación de AIR está instalada”](#) en la página 298.

Cuando se invoca la aplicación a través de la función de invocación desde el navegador, el objeto `NativeApplication` de la aplicación distribuye un objeto `BrowserInvokeEvent`. Para obtener más información, consulte [“Invocación desde el navegador”](#) en la página 274.

Si utiliza la función de invocación desde el navegador, asegúrese de tener en cuenta las implicaciones para la seguridad, descritas en el apartado [“Invocación desde el navegador”](#) en la página 274.

Una vez en ejecución la aplicación, ésta puede comunicarse con el contenido SWF en el navegador utilizando la clase `LocalConnection`. Para obtener más información, consulte [Comunicación entre aplicaciones](#).

Implementación en la empresa

Los administradores de TI pueden instalar el motor de ejecución de Adobe AIR y aplicaciones de AIR de forma silenciosa con herramientas de implementación estándar. Los administradores de TI pueden realizar las siguientes tareas:

- Realizar una instalación silenciosa del motor de ejecución de Adobe AIR empleando herramientas como Microsoft SMS, IBM Tivoli o cualquier herramienta de implementación que permita las instalaciones silenciosas que utilizan un arrancador
- Realizar una instalación silenciosa de la aplicación de AIR con las mismas herramientas que se utilizan para implementar el motor de ejecución

Para obtener más información, consulte la [Guía del administrador de Adobe AIR](http://www.adobe.com/go/learn_air_admin_guide_es) (http://www.adobe.com/go/learn_air_admin_guide_es).

Firma digital de archivos de AIR

La firma digital de sus archivos de instalación de AIR con un certificado emitido por una autoridad de certificación (AC) reconocida ofrece seguridad a los usuarios de que la aplicación que están instalando no ha sido alterada ni de modo accidental ni malintencionado e identifica a usted como el firmante (editor). Si la aplicación de AIR está firmada con un certificado de confianza, o está *encadenada* con un certificado de confianza en el ordenador de instalación, AIR presenta el nombre del editor durante la instalación. De lo contrario, el nombre del editor aparece como “Unknown” (desconocido).

Importante: Una entidad malentencionada podría falsificar un archivo de AIR con su identidad si lograra obtener su archivo de almacén de claves para la firma descubre si clave privada.

Información sobre certificados de firma de código

Las garantías de seguridad, limitaciones y obligaciones legales respecto del uso de certificados de firma de código se reseñan en la declaración de prácticas de certificación (DPC) y los acuerdos de suscriptor publicados por la autoridad de certificación emisora. Para obtener más información sobre los acuerdos para dos de las autoridades de certificación más importantes, consulte:

[CPS de Verisign](http://www.verisign.es/repository/cps20-test-ca.html) (<http://www.verisign.es/repository/cps20-test-ca.html>)

[Acuerdo de suscripción de Verisign](https://www.verisign.es/repository/subscriber/SUBAGR.html) (<https://www.verisign.es/repository/subscriber/SUBAGR.html>)

[DPC de Thawte](http://www.thawte.com/cps/index.html) (<http://www.thawte.com/cps/index.html>)

[Thawte Code Signing Developer's Agreement \(Acuerdo de desarrollador para firma de códigos de Thawte\)](http://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/develcertsign.pdf) (<http://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/develcertsign.pdf>)

Firma de códigos de AIR

Cuando se firma un archivo de AIR se incluye una firma digital en el archivo de instalación. La firma incluye un resumen del paquete, el cual se utiliza para verificar que el archivo de AIR no se haya alterado desde que se firmó, e incluye información acerca del certificado de firma, que sirve para verificar la identidad del editor.

AIR utiliza la infraestructura de claves públicas (PKI), compatible a través del almacén de certificados del sistema operativo, para establecer si un certificado es de confianza. El ordenador en el que se instala una aplicación de AIR debe o bien confiar directamente en el certificado que se utiliza para firmar la aplicación de AIR, o bien confiar en una serie de certificados que vinculan el certificado con una autoridad de certificación de confianza para que se pueda verificar la información del editor.

Si se firma un archivo de AIR con un certificado que no está encadenado con uno de los certificados raíz de confianza (que normalmente incluye a todos los certificados autofirmados), no se puede verificar la información del editor. Si bien AIR puede determinar que un paquete de AIR no ha sido alterado desde que se firmó, no hay manera de saber quién creó y firmó el archivo.

***Nota:** un usuario puede optar por confiar en un certificado autofirmado y en adelante las aplicaciones de AIR firmadas con el certificado presentarán el valor del campo de nombre común en el certificado como nombre del editor. AIR no proporciona ningún mecanismo para que un usuario designe un certificado como “de confianza”. El certificado (sin incluir la clave privada) debe proporcionarse por separado al usuario, quien deberá emplear uno de los mecanismos suministrados con el sistema operativo o una herramienta adecuada para importar el certificado en el lugar correcto en el almacén de certificados del sistema.*

Identificador del editor de AIR

Como parte del proceso de crear un archivo de AIR, el AIR Developer Tool (ADT) genera un ID del editor. Se trata de un identificador exclusivo del certificado que se utiliza para crear el archivo de AIR. Si se vuelve a utilizar el mismo certificado para varias aplicaciones de AIR, todas tendrán el mismo ID de editor. El ID de editor sirve para identificar la aplicación de AIR en la comunicación LocalConnection (consulte Comunicación entre aplicaciones). Para identificar el ID del editor de una aplicación instalada, lea la propiedad `NativeApplication.nativeApplication.publisherID`.

Para generar el ID del editor se utilizan los campos siguientes: Name, CommonName, Surname, GivenName, Initials, GenerationQualifier, DNQualifier, CountryName, localityName, StateOrProvinceName, OrganizationName, OrganizationalUnitName, Title, Email, SerialNumber, DomainComponent, Pseudonym, BusinessCategory, StreetAddress, PostalCode, PostalAddress, DateOfBirth, PlaceOfBirth, Gender, CountryOfCitizenship, CountryOfResidence y NameAtBirth. Si se renueva un certificado emitido por una autoridad de certificación, o se vuelve a generar un certificado autofirmado, para que el ID del editor no cambie estos campos deben ser iguales que la vez anterior. También deben ser iguales el certificado raíz de un certificado emitido por una AC y la clave pública de un certificado autofirmado.

Formatos de certificados

Las herramientas de firma de AIR aceptan cualquier almacén de claves accesibles a través de la arquitectura de criptografía de Java (Java Cryptography Architecture o JCA). En esto se incluyen los almacenes de claves basados en archivos, como archivos de formato PKCS12 (que suelen tener la extensión de archivo .pfx o .p12), archivos .keystore de Java, almacenes de claves de hardware PKCS11 y los almacenes de claves del sistema. Los formatos de almacenes de claves a los que tiene acceso ADT depende de la versión y configuración del motor de ejecución de Java que se utilice para ejecutar ADT. El acceso a algunos tipos de almacén de claves, como los tokens de hardware PKCS11, pueden necesitar que se instalen y configuren plug-ins de JCA y controladores de software adicionales.

Para firmar archivos de AIR se puede utilizar un certificado para firma de código existente de clase 3 y alta seguridad o se puede obtener uno nuevo. Se pueden utilizar, por ejemplo, cualquiera de los siguientes tipos de certificados de Verisign o Thawte:

- Verisign:
 - Microsoft Authenticode Digital ID

- Sun Java Signing Digital ID
- Thawte:
 - Certificado de AIR Developer
 - Certificado de Apple Developer
 - Certificado de JavaSoft Developer
 - Certificado de Microsoft Authenticode

Nota: el certificado debe estar marcado para realizar la firma de código. Normalmente no se puede utilizar un certificado SSL para firmar un archivo de AIR.

Marcas de hora

Cuando se firma un archivo de AIR, la herramienta de empaquetado consulta al servidor de una autoridad de marcas de hora para obtener una fecha y hora de la firma que sean verificables independientemente. La marca de hora que se obtiene se incorpora en el archivo de AIR. Siempre y cuando el certificado de firma sea válido en el momento de firmarlo, se podrá instalar el archivo de AIR, incluso después de caducado el certificado. Por otro lado, si no se obtiene la marca de hora, el archivo de AIR dejará de poder instalarse cuando caduque o se revoque el certificado.

La opción predeterminada es que las herramientas de empaquetado de AIR obtienen una marca de hora. No obstante, para que las aplicaciones puedan empaquetarse cuando no se dispone del servicio de marcas de hora, el marcado de hora se puede desactivar. Adobe recomienda que todo archivo de AIR que se distribuya al público incluya una marca de hora.

La autoridad de marcas de hora predeterminada que utilizan las herramientas de empaquetado de AIR es Geotrust.

Obtención de un certificado

Para obtener un certificado, normalmente visitaría el sitio Web de la autoridad de certificación y llevaría a cabo el proceso de adquisición de la empresa en cuestión. Las herramientas que se utilizan para producir el archivo del almacén de claves que necesitan las herramientas de AIR dependen del tipo de certificado que se compre, cómo se guarda el certificado en el ordenador receptor y, en algunos casos, el navegador que se utilizó para obtener el certificado. Por ejemplo: para obtener y exportar un certificado Authenticode de Microsoft, Verisign o Thawte exigen que se utilice Microsoft Internet Explorer. El certificado podrá entonces exportarse como archivo con prefijo .pfx directamente desde la interfaz de usuario de Internet Explorer.

Se puede generar un certificado autofirmado con Air Development Tool (ADT) que se utiliza para empaquetar los archivos de instalación de AIR. También pueden utilizarse algunas herramientas de terceros.

Para ver las instrucciones sobre cómo generar un certificado autofirmado, así como instrucciones para firmar un archivo de AIR, consulte “[Empaquetado de archivos de instalación de AIR con AIR Developer Tool \(ADT\)](#)” en la página 326. También se pueden exportar y firmar archivos de AIR con Flex Builder, Dreamweaver y la actualización de AIR para Flash

El siguiente ejemplo describe cómo obtener un certificado de desarrollador de AIR de la autoridad de certificación Thawte y prepararlo para el uso con ADT. Este ejemplo muestra una sola de las diversas formas de obtener y preparar un certificado de firma de código para su uso.

Ejemplo: Cómo obtener un certificado de desarrollador de AIR con Thawte

Para adquirir un certificado de desarrollador de AIR, el sitio Web de Thawte exige el uso del navegador Mozilla Firefox. La clave privada para el certificado se guarda en el almacén de claves del navegador. Asegúrese de que el almacén de claves de Firefox esté protegido con una contraseña maestra y que el ordenador mismo sea seguro desde el punto de vista físico. (Podrá exportar y eliminar el certificado y la clave privada del almacén de claves del navegador una vez finalizado el proceso de adquisición).

Como parte del proceso de inscripción para el certificado se genera un par de claves: pública y privada. La clave privada se guarda automáticamente en el almacén de claves de Firefox. Deberá utilizar el mismo ordenador y navegador para solicitar y recuperar el certificado del sitio Web de Thawte.

- 1 Visite el sitio Web de Thawte y vaya a la [página de productos para Certificados para firma de códigos](#).
- 2 En la lista de certificados para firma de códigos, seleccione el certificado de Adobe AIR Developer.
- 3 Complete el proceso de inscripción de tres pasos. Necesitará facilitar información sobre su organización, así como datos de contacto. A continuación, Thawte lleva a cabo su proceso de verificación de identidad y es posible que solicite información suplementaria. Una vez finalizada la verificación, Thawte le enviará un correo electrónico con instrucciones sobre cómo recuperar el certificado.

Nota: para obtener más información (en inglés) sobre el tipo de documentación que se requiere, haga clic en: https://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/enroll_codesign_eng.pdf.

- 4 Recupere el certificado emitido del sitio de Thawte. El certificado se guarda automáticamente en el almacén de claves de Firefox.
- 5 Exporte del almacén de claves un archivo que contenga la clave privada y el certificado siguiendo los pasos que se indican a continuación:

Nota: la clave privada y el certificado de Firefox se exportan en un formato .p12 (pfx) compatible con ADT, Flex, Flash y Dreamweaver.

- a Abra el cuadro de diálogo *Administrador de certificados* de Firefox:
 - b En Windows: abra Herramientas -> Opciones -> Opciones avanzadas -> Certificados -> Administrar certificados
 - c En Mac OS: abra Firefox -> Preferencias -> Avanzado -> Cifrado -> Ver certificados -> Sus certificados
 - d Seleccione el certificado para firma de códigos de Adobe AIR en la lista de certificados y haga clic en el botón **Hacer copia**.
 - e Escriba un nombre de archivo y el lugar al que se debe exportar el archivo del almacén de claves, y haga clic en **Guardar**.
 - f Si utilizó la contraseña maestra de Firefox, se le solicitará escribir la contraseña para el dispositivo de seguridad del software a fin de poder exportar el archivo. (Esta contraseña es de uso exclusivo en Firefox).
 - g En el cuadro de diálogo para la *contraseña para la copia de seguridad del certificado*, cree una contraseña para el archivo del almacén de claves.
Importante: Esta contraseña protege el archivo del almacén de claves y se necesita cuando se utiliza el archivo para firmar aplicaciones de AIR. Conviene elegir una contraseña segura.
 - h Haga clic en Aceptar. Debería recibir un mensaje confirmando la creación de la contraseña para copias de seguridad. El archivo del almacén contiene la clave privada y el certificado se guarda con una extensión .p12 (en formato PKCS12).
- 6 Utilice el archivo de almacén de claves exportado con ADT, Flex Builder, Flash o Dreamweaver. Hay que utilizar la contraseña creada para el archivo cada vez que se firme una aplicación de AIR.

Importante: La clave privada y el certificado se siguen guardando en el almacén de claves de Firefox. Mientras que esto permite exportar otra copia del archivo del certificado, también facilita otro punto de acceso que debe protegerse para mantener la seguridad de su certificado y su clave privada.

Cambio de certificado

En algunas circunstancias, quizá necesite cambiar el certificado que utiliza para firmar su aplicación de AIR. Entre estas circunstancias se encuentran:

- la actualización de un certificado autofirmado a un certificado emitido por una autoridad de certificación;
- el cambio de un certificado autofirmado a otro que está por caducar;
- el cambio de un certificado comercial a otro (por ejemplo, cuando cambia de identidad la empresa).

Al ser el certificado de firma uno de los elementos que determinan la identidad de una aplicación de AIR, no se puede simplemente firmar una actualización de la aplicación con otro certificado. Para que AIR reconozca un archivo de AIR como actualización, hay que firmar tanto el original como todo archivo de AIR actualizado con el mismo certificado. De lo contrario, AIR instalará el nuevo archivo de AIR como aplicación independiente en lugar de actualizar la existente.

A partir de AIR 1.1 se puede cambiar el certificado de firma de una aplicación utilizando una firma de migración. Una firma de migración es una segunda firma que se aplica al archivo de AIR de actualización. La firma de migración utiliza el certificado original, que establece que el firmante es el editor original de la aplicación.

Importante: hay que cambiar el certificado antes de que el certificado original caduque o se revoque. Si no se crea una actualización firmada con firma de migración antes de que caduque el certificado, los usuarios tendrán que desinstalar la versión existente de la aplicación antes de instalar la actualización. Los certificados emitidos por entidades comerciales suelen ser renovables para evitar que caduquen. Los certificados autofirmados no se pueden renovar.

Para cambiar el certificado:

- 1 Cree una actualización de la aplicación.
- 2 Empaquete y firme el archivo de AIR de actualización con el certificado **nuevo**.
- 3 Vuelva a firmar el archivo de AIR con el certificado **original** (con el comando `-migrate` de ADT).

El procedimiento para aplicar una firma de migración se describe en “[Firma de un archivo AIR para cambiar el certificado de la aplicación](#)” en la página 332.

La instalación del archivo de AIR actualizado cambia la identidad de la aplicación. Este cambio de identidad tiene las consecuencias siguientes:

- El ID del editor de la aplicación cambia para que se corresponda con el nuevo certificado.
- La nueva versión de la aplicación no tiene acceso a los datos que están en el almacén local cifrado existente.
- Cambia la ubicación del directorio de almacenamiento de la aplicación. Los datos de la ubicación anterior no se copian en el nuevo directorio. (Pero la nueva aplicación puede localizar el directorio original con base en el ID del editor anterior).
- La aplicación ya no puede abrir conexiones locales con el ID del editor anterior.
- Si un usuario reinstala un archivo de AIR anterior a la migración, AIR lo instala como aplicación independiente con el ID del editor anterior.

Es responsabilidad de la aplicación migrar los datos de la versión original a la nueva versión de la aplicación. Para migrar datos que se encuentran en el almacén local cifrado hay que exportar los datos antes de cambiar el certificado. La nueva versión de la aplicación no puede leer el almacén local cifrado de la versión anteriores. (A menudo resulta más fácil volver a crear los datos que migrarlos).

Debe seguir aplicando la firma de migración a todas las actualizaciones posteriores que sea posible. De lo contrario, los usuarios que aún no hayan actualizado el original tendrán que instalar una versión intermedia de migración o desinstalar la versión actual antes de poder instalar la última actualización. Llegará el momento, por supuesto, en que el certificado original habrá caducado y ya no se podrá aplicar una firma de migración. (No obstante, a menos que se desactive la marca de hora, los archivos de AIR que ya habían firmado con firma de migración seguirán siendo válidos. La firma de migración lleva una marca de hora para que AIR pueda aceptar la firma incluso después de caducado el certificado).

Un archivo de AIR con firma de migración es, en otros aspectos, un archivo de AIR normal. Si se instala la aplicación en un sistema que no tiene la versión original, AIR la instala de la forma habitual.

Nota: si se renueva un certificado de emisión comercial, no suele ser necesario migrar el certificado. Un certificado renovado conserva la misma identidad del editor que el original, a menos que haya cambiado el nombre distinguido. Para obtener una lista completa de los atributos del certificado que se utilizan para determinar el nombre distinguido, consulte “Identificador del editor de AIR” en la página 302.

Terminología

Esta sección contiene un glosario de algunos de los principales términos que necesita conocer al tomar decisiones sobre cómo firmar la aplicación para su distribución pública.

Término	Descripción
Autoridad de certificación (AC)	Entidad de una red de infraestructura de claves públicas que interviene como tercero de confianza y que, en última instancia, certifica la identidad del propietario de una clave pública. Una AC emite certificados digitales firmados por su propia clave privada para constatar que ha verificado la identidad del titular del certificado.
Declaración de prácticas de certificación (DPC)	Plantea las prácticas y políticas de la autoridad de certificación respecto de la emisión y verificación de certificados. La DPC forma parte del contrato entre la AC y sus suscriptores y partes confiantes. Reseña también las políticas de verificación de identidad y el nivel de garantía que ofrecen los certificados emitidos.
Lista de revocación de certificados (LRC)	Lista de certificados emitidos que han sido revocados y en los cuales ya no se debe confiar. AIR comprueba la LRC en el momento de firmarse una aplicación de AIR y, si no hay marca de hora, nuevamente cuando se instala la aplicación.
Cadena de certificación	Secuencia de certificados en la que cada certificado de la cadena ha sido firmado por el certificado siguiente.
Certificado digital	Documento digital que contiene información acerca de la identidad del propietario, la clave pública del propietario y la identidad del propio certificado. Un certificado emitido por una autoridad de certificación está firmado a su vez por un certificado de propiedad de la AC emisora.
Firma digital	Mensaje o resumen cifrado que sólo puede descifrarse con la clave pública de un par clave pública-clave privada. En una PKI la firma digital contiene un certificado digital (o más) que en última instancia llevan hasta la autoridad de certificación. Una firma digital sirve para validar que un mensaje (o un archivo informático) no ha sido alterado desde que se firmó (dentro de los límites de garantía que ofrezca el algoritmo criptográfico que se utilizó) y, suponiendo que se confía en la autoridad de certificación emisora, para validar la identidad del firmante.
Almacén de claves	Base de datos que contiene certificados digitales y, en algunos casos, las claves privadas asociadas.

Término	Descripción
Java Cryptography Architecture (JCA)	Arquitectura de criptografía de Java: arquitectura ampliable para administrar y acceder a los almacenes de claves. Para obtener más información, consulte la guía de consulta de Java Cryptography Architecture (en inglés) .
PKCS #11	Cryptographic Token Interface Standard (norma de interfaces de tokens criptográficos) de RSA Laboratories. Un almacén de claves basado en tokens de hardware.
PKCS #12	Personal Information Exchange Syntax Standard (norma para la sintaxis de intercambio de información personal) de RSA Laboratories. Un almacén de claves basado en archivo que suele contener una clave privada y su certificado digital asociado.
Clave privada	La mitad privada de un sistema criptográfico asimétrico que consta de clave pública y clave privada. La clave privada es confidencial y no debe nunca transmitirse a través de una red. Los mensajes con firma digital son cifrados con la clave privada por parte del firmante.
Clave pública	La mitad pública de un sistema criptográfico asimétrico que consta de clave pública y clave privada. La clave pública se distribuye libremente y se utiliza para descifrar los mensajes cifrados con la clave privada.
Public Key Infrastructure (PKI)	Infraestructura de claves públicas: sistema de confianza en el que las autoridades de certificación autentican la identidad de los propietarios de claves públicas. Los clientes de la red confían en los certificados digitales emitidos por una AC de confianza para verificar la identidad del firmante de un mensaje o archivo digital.
Marca de hora	Dato firmando digitalmente que contiene la fecha y hora en que se produjo un evento. ADT puede incluir en un paquete de AIR una marca de hora provista por un servidor de hora en conformidad con la norma RFC 3161 . Cuando la hay, AIR utiliza la marca de hora para establecer la validez de un certificado en el momento de firmar. Esto permite instalar una aplicación de AIR una vez caducado su certificado de firma.
Autoridad de marcas de hora	Autoridad que emite marcas de hora. Para que AIR la reconozca, la marca de hora debe estar en conformidad con la norma RFC 3161 y la firma de la marca de hora debe encadenarse con un certificado raíz de confianza en la máquina en que se instale la aplicación.

Capítulo 32: Actualización de aplicaciones de AIR

Los usuarios pueden instalar o actualizar cualquier aplicación de AIR haciendo doble clic en el archivo de AIR de su equipo o desde un navegador (mediante la perfeccionada función de instalación). El instalador de Adobe® AIR™ gestiona la instalación y avisa al usuario si está actualizando una aplicación previa existente. (Consulte [“Distribución, instalación y ejecución de aplicaciones de AIR”](#) en la página 292.)

Sin embargo, también es posible permitir que las propias aplicaciones se actualicen solas mediante la clase Updater. (Una aplicación instalada puede detectar nuevas versiones disponibles para su descarga e instalación.) La clase Updater incluye un método `update()` que permite al usuario apuntar a un archivo de AIR de un equipo y actualizar a dicha versión.

Tanto el ID de aplicación como el ID de editor de un archivo de actualización de AIR deben coincidir para que la aplicación se actualice. El ID de editor proviene del certificado de firma. Por ello, tanto la actualización como la aplicación que va a actualizarse deben estar firmadas con el mismo certificado.

En AIR 1.1, es posible migrar una aplicación para utilizar un nuevo certificado de firma para el código. Para migrar una aplicación y utilizar una nueva firma, es preciso firmar el archivo de actualización de AIR con el certificado nuevo y con el original. La migración de certificados es un proceso que no se puede invertir. Una vez concluida la migración, sólo se reconocerán como actualizaciones de la instalación existente aquellos archivos de AIR firmados con el nuevo certificado (o con ambos certificados).

Puede utilizar la migración de certificados para pasar de un certificado firmado automáticamente a un certificado comercial de firma de código, o de uno firmado automáticamente a otro del mismo tipo. Si no migra el certificado, los usuarios existentes deberán desinstalar su versión actual de la aplicación para poder instalar la nueva versión. Para obtener más información, consulte [“Cambio de certificado”](#) en la página 305.

Actualización de aplicaciones

La clase Updater (del paquete `flash.desktop`) incluye un método, `update()`, que se puede utilizar para actualizar la aplicación actualmente en ejecución a una versión distinta. Por ejemplo, si el usuario tiene una versión del archivo de AIR ("Sample_App_v2.air") en el escritorio, el siguiente código actualizaría la aplicación:

```
var updater:Updater = new Updater();
var airFile:File = File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version:String = "2.01";
updater.update(airFile, version);
```

Antes de utilizar la clase Updater, el usuario o la aplicación deben descargar la versión actualizada del archivo de AIR en el equipo. Para obtener más información, consulte [“Descarga de un archivo de AIR en el equipo del usuario”](#) en la página 310.

Resultados de la llamada al método

Cuando una aplicación del motor de ejecución llama al método `update()`, éste cierra la aplicación y, seguidamente, intenta instalar la nueva versión del archivo de AIR. Se comprueba que el ID de aplicación y el ID de editor especificados en el archivo de AIR coinciden con el ID de aplicación y de editor de la aplicación que llama al método `update()`. (Para obtener más información sobre el ID de aplicación y el ID de editor, consulte “[Configuración de las propiedades de una aplicación de AIR](#)” en la página 44.) También comprueba si la cadena de versión coincide con la cadena `version` transferida al método `update()`. Si la instalación concluye correctamente, el motor de ejecución abre la nueva versión de la aplicación. En caso contrario (si la instalación no concluye correctamente), vuelve a abrir la versión existente de la aplicación (previa a la instalación).

En Mac OS, para instalar una versión actualizada de una aplicación, el usuario debe disponer de los privilegios necesarios para instalar aplicaciones en el directorio de aplicaciones. En Windows, el usuario debe disponer de privilegios de administrador.

Si la versión actualizada de la aplicación requiere una versión actualizada del motor de ejecución, se instala la versión más reciente del motor de ejecución. Para actualizar el motor de ejecución, el usuario debe disponer de privilegios de administrador en el equipo.

Al verificar una aplicación con ADL, llamar al método `update()` produce una excepción de tiempo de ejecución.

Cadena de versión

Para que el archivo de AIR se pueda instalar, la cadena que se especifica como parámetro `version` del método `update()` debe coincidir con la cadena del atributo `version` del elemento principal `application` del archivo descriptor de la aplicación. Es preciso especificar el parámetro `version` por motivos de seguridad. Al solicitar a la aplicación que verifique el número de versión del archivo de AIR, la aplicación no instala de forma accidental una versión anterior que aún pueda contener vulnerabilidades de seguridad solucionadas en la versión ya instalada. La aplicación también comprueba la cadena de versión en el archivo de AIR y la compara con la de la aplicación instalada para evitar desactualizaciones.

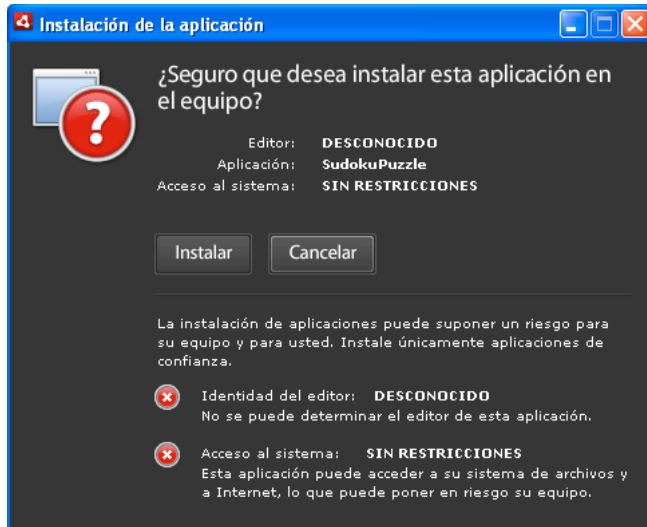
La cadena de versión puede tener cualquier formato. Por ejemplo, "2.01" o "versión 2". Es el usuario (el desarrollador de la aplicación) quien debe decidir qué formato darle a esta cadena. El motor de ejecución no valida la cadena de versión; el código de la aplicación debe hacerlo antes de actualizar la aplicación.

Si una aplicación de Adobe AIR descarga un archivo de AIR por Internet, se recomienda disponer de un mecanismo que permite al servicio Web notificar a la aplicación de Adobe AIR sobre la descarga actual de la versión. La aplicación puede utilizar esta cadena como el parámetro `version` del método `update()`. Si el archivo de AIR se obtiene por otros medios que impiden conocer la versión del archivo de AIR, la aplicación de AIR puede examinar el archivo de AIR para extraer la información sobre su versión. (Un archivo de AIR es un archivo ZIP comprimido y el archivo descriptor de la aplicación es el segundo registro del archivo.)

Para obtener más información sobre el archivo descriptor de aplicación, consulte “[Configuración de las propiedades de una aplicación de AIR](#)” en la página 44.

Presentación de interfaz de usuario de actualización personalizada de la aplicación

AIR incluye una interfaz de actualización predeterminada:



Esta interfaz siempre se utiliza la primera vez que el usuario instala la versión de una aplicación en un ordenador. Sin embargo, es posible definir una interfaz propia para utilizarla en el futuro. Para ello, especifique un elemento `customUpdateUI` en el archivo descriptor de la aplicación de la aplicación instalada actualmente:

```
<customUpdateUI>true</customUpdateUI>
```

Cuando la aplicación se instale y el usuario abra un archivo de AIR con un ID de aplicación e ID de editor que coincidan con los de la aplicación instalada, será el motor de ejecución el encargado de abrir la aplicación, no el archivo de instalación predeterminado de la aplicación de AIR. Para obtener más información, consulte [“Interfaz de usuario personalizada para actualizaciones”](#) en la página 51.

Al invocar la aplicación (cuando el objeto `NativeApplication.nativeApplication` distribuye un evento `invoke`), ésta puede decidir si actualiza o no la aplicación (con la clase `Updater`). Si decide actualizarse, puede presentar su propia interfaz de instalación al usuario (interfaz que no es igual que la estándar).

Descarga de un archivo de AIR en el equipo del usuario

Para poder utilizar la clase `Updater`, el usuario o la aplicación deben guardar primero localmente un archivo de AIR en el equipo del usuario. Por ejemplo, el siguiente código lee un archivo de AIR desde una dirección URL (http://example.com/air/updates/Sample_App_v2.air) y lo guarda en el directorio de almacenamiento de la aplicación:

```
var urlString:String = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq:URLRequest = new URLRequest(urlString);
var urlStream:URLStream = new URLStream();
var fileData:ByteArray = new ByteArray();
urlStream.addEventListener(Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event:Event):void {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile():void {
    var file:File = File.applicationStorageDirectory.resolvePath("My App v2.air");
    var fileStream:FileStream = new FileStream();
    fileStream.open(file, FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

Para obtener más información, consulte [“Flujo de trabajo de lectura y escritura de archivos”](#) en la página 116.

Ver si una aplicación se está ejecutando por primera vez

Una vez actualizada la aplicación, puede ofrecer al usuario un mensaje de bienvenida o de primeros pasos. Al iniciarse, la aplicación comprueba si se está ejecutando por primera vez para saber si debe mostrar o no el mensaje.

Una forma de hacerlo es guardar un archivo en el directorio de almacenamiento de la aplicación al inicializarla. Cada vez que se inicie la aplicación, comprobará si existe este archivo. Si el archivo no existe, significa que la aplicación se está ejecutando por primera vez para el usuario. Si el archivo existe, significa que el usuario ya ha ejecutado la aplicación al menos una vez. Si el archivo existe y su número de versión es anterior a la versión actual, significa que el usuario está ejecutando la aplicación por primera vez.

Si la aplicación guarda datos localmente (por ejemplo, en el directorio de almacenamiento de la aplicación), puede buscar datos guardados previamente (de versiones anteriores) durante la primera ejecución de la aplicación.

Capítulo 33: Localización de aplicaciones de AIR

Adobe® AIR™ 1.1 está preparado para el uso en diversos idiomas.

Introducción a la localización

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.

El SDK de AIR proporciona una arquitectura de localización de HTML (que se encuentra en el archivo `AIRLocalizer.js` file). Esta arquitectura incluye las API que ayudan a trabajar con varias configuraciones regionales. Para obtener más información, consulte "[Localización de contenido HTML](#)" en la página 314.

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.

Localización del nombre y la descripción en el instalador de la aplicación

Se pueden especificar varios idiomas para los elementos `name` y `description` en el archivo descriptor de la aplicación. En el ejemplo siguiente se especifica el nombre de la aplicación en tres idiomas (inglés, francés y alemán):

```
<name>
  <text xml:lang="en">Sample 1.0</text>
  <text xml:lang="fr">Échantillon 1.0</text>
  <text xml:lang="de">Stichprobe 1.0</text>
</name>
```

El atributo `xml:lang` de cada elemento de texto especifica un código de idioma, tal como se define en RFC4646 (<http://www.ietf.org/rfc/rfc4646.txt>).

El elemento "name" sólo define el nombre de la aplicación que presenta el instalador de aplicaciones de AIR. El instalador de aplicaciones de AIR utiliza el valor localizado que mejor se corresponde con el idioma de la interfaz de usuario definido en la configuración del sistema operativo.

Se pueden especificar varias versiones de idiomas del elemento `description` en el archivo descriptor de la aplicación. Este elemento define el texto descriptivo que presenta el instalador de aplicaciones de AIR.

Estas opciones sólo se aplican a los idiomas que se ofrecen en el instalador de aplicaciones de AIR. No definen las configuraciones regionales disponibles para la aplicación instalada y en funcionamiento. Las aplicaciones de AIR pueden ofrecer interfaces de usuario compatibles con varios idiomas, incluidos los del instalador de aplicaciones de AIR y otros adicionales.

Para obtener más información, consulte "[Definición de propiedades en el archivo descriptor de la aplicación](#)" en la página 45.

Elección de una configuración regional

Para determinar qué configuración regional utiliza la aplicación se puede utilizar uno de los métodos siguientes:

- **Solicitud al usuario:** se puede iniciar la aplicación con una configuración regional predeterminada y después pedir al usuario que seleccione la que prefiera.
- **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 guión (por ejemplo: "es-ES" o "ja-JP"). La primera entrada del conjunto devuelto tiene el mismo ID de idioma primario 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` sería 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 sólo 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 Flash

Flash CS3 y CS4 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 del lenguaje y componentes ActionScript 3.0](#) para Flash.

Localización de contenido HTML

El SDK de AIR 1.1 incluye una arquitectura de localización de HTML. El archivo JavaScript AIRLocalizer.js define la arquitectura. El archivo AIRLocalizer.js se encuentra en el directorio frameworks del SDK de AIR. Este archivo incluye la clase air.Localizer, que ofrece funciones de utilidad para la creación de aplicaciones compatibles con varias versiones localizadas.

Carga del código de la arquitectura de localización de HTML de AIR

Para utilizar la arquitectura de localización, copie el archivo AIRLocalizer.js en su proyecto. Inclúyalo en el archivo HTML principal de la aplicación con una etiqueta de script:

```
<script src="AIRLocalizer.js" type="text/javascript" charset="utf-8"></script>
```

El código JavaScript que le sigue llamar al objeto air.Localizer.localizer:

```
<script>
    var localizer = air.Localizer.localizer;
</script>
```

El objeto air.Localizer.localizer es un objeto de instancia única que define métodos y propiedades para utilizar y gestionar los recursos localizados. La clase Localizer incluye los métodos siguientes:

Método	Descripción
getFile()	Obtiene el texto de un paquete de recursos especificado para una configuración regional especificada. Consulte "Obtención de recursos para una configuración regional específica" en la página 320.
getLocaleChain()	Devuelve los idiomas de la cadena de configuraciones regionales. Consulte "Definición de la cadena de configuraciones regionales" en la página 319.
getString()	Obtiene la cadena de caracteres definida para un recurso. Consulte "Obtención de recursos para una configuración regional específica" en la página 320.
setBundlesDirectory()	Configura la ubicación del directorio de paquetes (bundles). Consulte "Personalización de las opciones de AIR HTML Localizer" en la página 318.
setLocalAttributePrefix()	Define el prefijo para los atributos de localización que se utilizan en los elementos DOM de HTML. Consulte "Personalización de las opciones de AIR HTML Localizer" en la página 318.
setLocaleChain()	Define el orden de los idiomas en la cadena de configuraciones regionales. Consulte "Definición de la cadena de configuraciones regionales" en la página 319.
sortLanguagesByPreference()	Ordena las configuraciones regionales en la cadena de configuraciones regionales en función del orden en que se encuentran en la configuración del sistema operativo. Consulte "Definición de la cadena de configuraciones regionales" en la página 319.
update()	Actualiza el DOM de HTML (o un elemento DOM) con cadenas de caracteres localizadas procedentes de la actual cadena de configuraciones regionales. Para ver una discusión sobre las cadenas de configuraciones regionales, consulte "Gestión de cadenas de configuraciones regionales" en la página 316. Para obtener más información sobre el método update(), consulte "Actualización de elementos DOM para utilizar la configuración regional actual" en la página 317.

La clase Localizer incluye las siguientes propiedades estáticas:

Propiedad	Descripción
localizer	Devuelve una referencia al objeto Localizer de instancia única para la aplicación.
ultimateFallbackLocale	La configuración regional que se utiliza cuando la aplicación no admite ninguna de las preferencias del usuario. Consulte “Definición de la cadena de configuraciones regionales” en la página 319.

Definición de paquetes de recursos

La arquitectura de localización de HTML lee las versiones localizadas de cadenas de caracteres en los archivos de *localización*. Un archivo de localización es una colección de valores basados en clave y serializados en un archivo de texto. A veces se le conoce por la palabra *paquete*.

Cree un subdirectorio del directorio del proyecto de la aplicación llamado “configregional”. (Puede usar otro nombre; consulte [“Personalización de las opciones de AIR HTML Localizer”](#) en la página 318.) Este directorio incluirá los archivos de localización. Este directorio se conoce como el *directorio de paquetes*.

Para cada configuración regional que admita la aplicación, cree un subdirectorio del directorio de paquetes. Dé a cada subdirectorio un nombre que corresponda al código de la configuración regional. Por ejemplo: llame al directorio francés “fr” y al directorio inglés “en”. Para definir una configuración regional con códigos de idioma y de país se puede utilizar guión bajo (_). Por ejemplo, el directorio de inglés estadounidense se llamaría “en_us”. (También se puede utilizar un guión normal en lugar de un guión bajo: “en-us”. La arquitectura de localización de HTML admite ambas formas).

No hay límite de la cantidad de archivos de recursos que se pueden añadir a un subdirectorio de configuraciones regionales. Se suele crear un archivo de localización para cada idioma (y colocar el archivo en el directorio de ese idioma). La arquitectura de localización de HTML incluye un método `getFile()` que permite leer el contenido de un archivo (consulte [“Obtención de recursos para una configuración regional específica”](#) en la página 320).

Los archivos que tienen la extensión de archivo `.properties` se denominan “archivos de propiedades de localización”. Se pueden utilizar para definir pares clave-valor para una configuración regional. Un archivo de propiedades define un valor de cadena en cada línea. En el siguiente ejemplo se define el valor de cadena “Hello in English.” para una clave denominada `greeting`:

```
greeting=Hello in English.
```

Un archivo de propiedades que contiene el texto siguiente define seis pares clave-valor:

```
title=Sample Application
greeting=Hello in English.
exitMessage=Thank you for using the application.
color1=Red
color2=Green
color3=Blue
```

Este ejemplo muestra una versión en inglés del archivo de propiedades, que se guarda en el directorio `en`.

La versión en francés de este archivo de propiedades se coloca en el directorio `fr`:

```
title=Programme d'échantillon
greeting=Bonjour en français.
exitMessage=Merci pour l'usage du programme.
color1=Rouge
color2=Vert
color3=Bleu
```

Se pueden definir varios archivos de recursos para distintos tipos de información. Por ejemplo: un archivo `legal.properties` podría contener un texto jurídico estándar (como un aviso de copyright). Quizá se desee utilizar ese recurso en varias aplicaciones. Asimismo, se pueden definir archivos separados que definen el contenido localizado para distintas partes de la interfaz de usuario.

Utilice para estos archivos la codificación UTF-8 para mayor compatibilidad con distintos idiomas.

Gestión de cadenas de configuraciones regionales

Cuando la aplicación carga el archivo `AIRLocalizer.js`, la misma examina las configuraciones regionales que tiene definidas. Estas configuraciones regionales corresponden a los subdirectorios del directorio de paquetes (consulte “[Definición de paquetes de recursos](#)” en la página 315). Esta lista de configuraciones regionales disponibles se denomina la *cadena de configuraciones regionales*. El archivo `AIRLocalizer.js` ordena automáticamente la cadena de configuraciones regionales en función del orden definido en la configuración del sistema operativo. (La propiedad `Capabilities.languages` enumera los idiomas de la interfaz de usuario del sistema operativo en orden de preferencia).

De este modo, si una aplicación define recursos para las configuraciones regionales “es”, “es_ES” y “es_UY”, la arquitectura AIR HTML Localizer ordena en consecuencia la cadena de configuraciones regionales. Cuando se inicia una aplicación en un sistema que da aviso de “es” como configuración regional primaria, la cadena de configuraciones regionales se ordena en la secuencia [“es”, “es_ES”, “es_UY”]. En este caso la aplicación busca recursos primero en el paquete “es” y después en el paquete “es_ES”.

Sin embargo, si el sistema da aviso de “es_ES” como configuración regional primaria, la clasificación es [“es_ES”, “es”, “es_UY”]. En este caso la aplicación busca recursos primero en el paquete “es_ES” y después en el paquete “es”.

La aplicación define automáticamente la primera configuración regional de la cadena como la configuración regional predeterminada a utilizarse. Puede pedir al usuario que seleccione una configuración regional la primera vez que ejecuta la aplicación. Puede optar por guardar la selección en un archivo de preferencias y en adelante utilizar esa configuración regional cada vez que se inicie la aplicación.

La aplicación puede utilizar cadenas de caracteres de recurso en cualquier configuración regional de la cadena de configuraciones regionales. Si una configuración regional específica no define una cadena de caracteres de recurso, la aplicación utiliza la siguiente cadena de caracteres de recurso que coincida para otras configuraciones regionales definidas en la cadena de dichas configuraciones.

Se puede personalizar la cadena de configuraciones regionales llamando al método `setLocaleChain()` del objeto `Localizer`. Consulte “[Definición de la cadena de configuraciones regionales](#)” en la página 319.

Actualización de los elementos DOM con contenido localizado

Un elemento de la aplicación puede hacer referencia a un valor de clave de un archivo de propiedades de localización. En el ejemplo siguiente, el elemento `title` especifica un atributo `local_innerHTML`. La arquitectura de localización utiliza este atributo para buscar un valor localizado. De forma predeterminada, la arquitectura busca nombres de atributo que empiezan con “local_”. La arquitectura actualiza los atributos cuyos nombres coinciden con el texto que sigue a “local_”. En este caso, la arquitectura define el atributo `innerHTML` del elemento `title`. El atributo `innerHTML` utiliza el valor definido para la clave `mainWindowTitle` en el archivo de propiedades predeterminadas (`default.properties`):

```
<title local_innerHTML="default.mainWindowTitle"/>
```

Si la configuración regional actual no define ningún valor que coincida, la arquitectura de localización busca en el resto de la cadena de configuraciones regionales. Utiliza la siguiente configuración regional de la cadena que tenga definido un valor.

En el siguiente ejemplo el texto (el atributo `innerHTML`) del elemento `p` utiliza el valor de la clave `greeting` definido en el archivo de propiedades predeterminadas:

```
<p local_innerHTML="default.greeting" />
```

En el siguiente ejemplo el atributo del valor (y el texto mostrado) del elemento `input` utiliza el valor de la clave `btnBlue` definido en el archivo de propiedades predeterminadas:

```
<input type="button" local_value="default.btnBlue" />
```

Para actualizar el DOM de HTML para que utilice las cadenas de caracteres definidas en la cadena de configuraciones regionales actual, llame al método `update()` del objeto `Localizer`. Al llamar al método `update()` el objeto `Localizer` analiza el DOM y aplica manipulaciones donde encuentre atributos de localización ("`local_...`");

```
air.Localizer.localizer.update();
```

Se pueden definir valores tanto para un atributo ("`innerHTML`", por ejemplo) como para su correspondiente atributo de localización ("`local_innerHTML`", por ejemplo). En este caso, la arquitectura de localización sólo sobrescribe el valor del atributo si encuentra un valor coincidente en la cadena de localización. Por ejemplo, el siguiente elemento define ambos atributos, `value` y `local_value`:

```
<input type="text" value="Blue" local_value="default.btnBlue"/>
```

También puede actualizarse un solo elemento DOM en particular. Consulte el apartado siguiente, "[Actualización de elementos DOM para utilizar la configuración regional actual](#)" en la página 317.

De forma predeterminada, AIR HTML Localizer utiliza "`local_`" como prefijo para los atributos que definen las opciones de localización para un elemento. Por ejemplo: de forma predeterminada, un atributo `local_innerHTML` define el nombre del paquete y recurso que se utiliza para el valor `innerHTML` de un elemento. También de forma predeterminada, un atributo `local_value` define el nombre del paquete y recurso que se utiliza para el atributo `value` de un elemento. Se puede configurar AIR HTML Localizer para que utilice otro prefijo de atributo en lugar de "`local_`". Consulte "[Personalización de las opciones de AIR HTML Localizer](#)" en la página 318.

Actualización de elementos DOM para utilizar la configuración regional actual

Cuando el objeto `Localizer` actualiza del DOM de HTML, hace que los elementos marcados utilicen valores de atributo basados en las cadenas de caracteres definidas en la actual cadena de configuraciones regionales. Para que el localizador de HTML actualice el DOM de HTML, llame al método `update()` del objeto `Localizer`:

```
air.Localizer.localizer.update();
```

Para actualizar un solo elemento DOM especificado, páselo como parámetro al método `update()`. El método `update()` tiene un solo parámetro, `parentNode`, que es optativo. Cuando está especificado, el parámetro `parentNode` define el elemento DOM que se debe localizar. Llamar al método `update()` y especificar un parámetro `parentNode` define valores localizados para todos los elementos secundarios que especifican atributos de localización.

Por ejemplo, tomemos el siguiente elemento `div`:

```
<div id="colorsDiv">
  <h1 local_innerHTML="default.lblColors" ></h1>
  <p><input type="button" local_value="default.btnBlue" /></p>
  <p><input type="button" local_value="default.btnRed" /></p>
  <p><input type="button" local_value="default.btnGreen" /></p>
</div>
```

Para actualizar este elemento de modo que utilice cadenas de caracteres localizadas en la cadena de configuraciones regionales actual, use el código JavaScript siguiente:

```
var divElement = window.document.getElementById("colorsDiv");
air.Localizer.localizer.update(divElement);
```

Si no se encuentra un valor de clave en la cadena de configuraciones regionales, la arquitectura de localización define como valor de atributo el valor del atributo "local_". Por ejemplo: supongamos que en el ejemplo anterior la arquitectura de localización no encuentra ningún valor para la clave `lblColors` (en ninguno de los archivos `default.properties` de la cadena de configuraciones regionales). En este caso, utiliza "default.lblColors" como el valor de `innerHTML`. El uso de este valor indica (al desarrollador) que faltan recursos.

El método `update()` distribuye un evento `resourceNotFound` cuando no encuentra un recurso en la cadena de configuraciones regionales. La constante `air.Localizer.RESOURCE_NOT_FOUND` define la cadena "resourceNotFound". El evento tiene tres propiedades: `bundleName`, `resourceName` y `locale`. La propiedad `bundleName` es el nombre del paquete en el que no se encuentra el recurso. La propiedad `resourceName` es el nombre del recurso no disponible. La propiedad `locale` es el nombre de la configuración regional en la que no se encuentra el recurso.

El método `update()` distribuye un evento `bundleNotFound` cuando no encuentra el paquete especificado. La constante `air.Localizer.BUNDLE_NOT_FOUND` define la cadena "bundleNotFound". El evento tiene tres propiedades: `bundleName` y `locale`. La propiedad `bundleName` es el nombre del paquete en el que no se encuentra el recurso. La propiedad `locale` es el nombre de la configuración regional en la que no se encuentra el recurso.

El método `update()` funciona de modo asíncrono (y distribuye los eventos `resourceNotFound` y `bundleNotFound` de forma asíncrona). El siguiente código define detectores para los eventos `resourceNotFound` y `bundleNotFound`:

```
air.Localizer.localizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, bnfHandler);
air.Localizer.localizer.update();
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + " :." + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + " :." + event.locale);
}
```

Personalización de las opciones de AIR HTML Localizer

El método `setBundlesDirectory()` del objeto `Localizer` permite personalizar la ruta del directorio de paquetes. El método `setLocalAttributePrefix()` del objeto `Localizer` permite personalizar la ruta del directorio de paquetes y el valor de atributo que utiliza el `Localizer`.

El directorio de paquetes predeterminado se define como el subdirectorio de configuraciones regionales del directorio de la aplicación. Para especificar otro directorio, llame al método `setBundlesDirectory()` del objeto `Localizer`. Este método utiliza un parámetro, `path`, que es la ruta al directorio de paquetes deseado, en forma de cadena de caracteres. El parámetro `path` puede tener cualquiera de los valores siguientes:

- Una cadena que define una ruta relativa al directorio de la aplicación, como "configregionales"
- Una cadena que define una URL válida que utiliza los esquemas de URL `app`, `app-storage` o `file`, por ejemplo "app://languages" (*no* utilice el esquema de URL `http`)
- Un objeto `File`

Para obtener información sobre las URL y rutas de directorio, consulte "[Rutas a objetos File](#)" en la página 102.

En el siguiente ejemplo, el código define como directorio de paquetes un subdirectorio "languages" del directorio de almacenamiento de la aplicación (y no el directorio de la aplicación):

```
air.Localizer.localizer.setBundlesDirectory("languages");
```

Pase una ruta válida como parámetro `path`. De lo contrario, el método emite una excepción `BundlePathNotFoundError`. Este error tiene a `BundlePathNotFoundError` como su propiedad `name` y su propiedad `message` especifica la ruta no válida.

De forma predeterminada, AIR HTML Localizer utiliza "local_" como prefijo para los atributos que definen las opciones de localización para un elemento. Por ejemplo, el atributo `local_innerHTML` define el nombre del paquete y recurso que se utiliza para el valor `innerHTML` del siguiente elemento `input`:

```
<p local_innerHTML="default.greeting" />
```

El método `setLocalAttributePrefix()` del objeto `Localizer` permite utilizar otro prefijo de atributo en lugar de "local_". Este método estático utiliza un parámetro, que es la cadena de caracteres que se desea utilizar como prefijo de atributo. En el siguiente ejemplo, el código define la arquitectura de localización para que utilice "loc_" como prefijo de atributo:

```
air.Localizer.localizer.setLocalAttributePrefix("loc_");
```

Se puede personalizar el prefijo de atributo que utiliza la arquitectura de localización. Puede ser conveniente personalizar el prefijo si el valor predeterminado ("local_") está en conflicto con el nombre de otro atributo que se utilice en el código. Cuando llame a este método, asegúrese de utilizar caracteres válidos para los atributos de HTML. (Por ejemplo, el valor no puede contener un carácter de espacio en blanco).

Para obtener más información sobre el uso de atributos de localización en elementos de HTML, consulte ["Actualización de los elementos DOM con contenido localizado"](#) en la página 316.

Las opciones de directorio de paquetes y prefijo de atributo no persisten entre distintas sesiones de la aplicación. Si utiliza opciones personalizadas para el directorio de paquetes o el prefijo de atributo, asegúrese de configurarlas cada vez que inicie la aplicación.

Definición de la cadena de configuraciones regionales

Cuando se carga el código de `AIRLocalizer.js`, define automáticamente la cadena de configuraciones regionales predeterminada. Las configuraciones regionales disponibles en el directorio de paquetes y la configuración de idiomas del sistema operativo definen esta cadena de configuraciones regionales. (Para obtener más información, consulte ["Gestión de cadenas de configuraciones regionales"](#) en la página 316).

Se puede modificar la cadena de configuraciones regionales llamando al método estático `setLocaleChain()` del objeto `Localizer`. Por ejemplo, puede ser conveniente llamar a este método si el usuario indica una preferencia para un idioma concreto. El método `setLocaleChain()` utiliza un solo parámetro, `chain`, que es un conjunto de configuraciones regionales, por ejemplo `["fr_FR", "fr", "fr_CA"]`. El orden de las configuraciones locales en el conjunto define el orden en que la arquitectura busca recursos (en operaciones posteriores). Si no se encuentra un recurso para la primera configuración regional de la cadena, sigue buscando en los recursos de la otra configuración regional. Si falta el argumento `chain`, o si no es un conjunto o es un conjunto vacío, la función falla y emite una excepción `IllegalArgumentsError`.

El método estático `getLocaleChain()` del objeto `Localizer` devuelve un conjunto que enumera las configuraciones regionales de la cadena de configuraciones regionales actual.

El siguiente código lee la cadena de configuraciones regionales actual y añade dos configuraciones regionales francesas a la cabeza de la cadena:

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
```

El método `setLocaleChain()` distribuye un evento "change" cuando actualiza la cadena de configuraciones regionales. La constante `air.Localizer.LOCALE_CHANGE` define la cadena "change". El evento tiene una propiedad, `localeChain`, que es un conjunto de códigos de configuración regional en la nueva cadena de configuraciones regionales. El siguiente código define un detector para este evento:

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
localizer.addEventListener(air.Localizer.LOCALE_CHANGE, changeHandler);
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
function changeHandler(event)
{
    alert(event.localeChain);
}
```

La propiedad estática `air.Localizer.ultimateFallbackLocale` representa la configuración regional que se utiliza cuando la aplicación no admite ninguna de las preferencias del usuario. El valor predeterminado es "en". Se lo puede cambiar a otra configuración regional, como en el código siguiente:

```
air.Localizer.ultimateFallbackLocale = "fr";
```

Obtención de recursos para una configuración regional específica

El método `getString()` del objeto `Localizer` devuelve la cadena definida para un recurso en una configuración regional específica. No hace falta especificar un valor `locale` cuando se llama al método. En este caso el método busca en toda la cadena de configuraciones regionales y devuelve la cadena de caracteres de la primera configuración regional que proporciona el nombre del recurso especificado. El método utiliza los siguientes parámetros:

Parámetro	Descripción
<code>bundleName</code>	El paquete que contiene el recurso. Es el nombre del archivo de propiedades sin la extensión <code>.properties</code> . Por ejemplo: si este parámetro está definido en "alerts", el código del Localizer busca en archivos de localización que tengan el nombre <code>alerts.properties</code> .
<code>resourceName</code>	El nombre del recurso.
<code>templateArgs</code>	Opcional. Un conjunto de cadenas para sustituir las etiquetas numeradas en la cadena de sustitución. Tomemos como ejemplo una llamada a la función en que el parámetro <code>templateArgs</code> es ["Raúl", "4"] y la cadena del recurso coincidente es "Hello, {0}. You have {1} new messages.". En este caso, la función devuelve "Hello, Raúl. You have 4 new messages.". Para pasar por alto esta opción, pase un valor <code>null</code> .
<code>locale</code>	Opcional. El código de la configuración regional (por ejemplo: "en", "en_us" o "fr") que se debe utilizar. Si se facilita una configuración regional y no se encuentra ningún valor coincidente, el método no seguirá buscando valores en otras configuraciones regionales de la cadena. Si no se especifica ningún código de configuración regional, la función devuelve la cadena de caracteres que está en la primera configuración regional que proporciona un valor para el nombre del recurso especificado.

La arquitectura de localización puede actualizar los atributos marcados del DOM de HTML. Hay también otras formas de utilizar cadenas localizadas. Por ejemplo, se puede utilizar una cadena de caracteres en HTML generado de forma dinámica o como valor de parámetro en una llamada a una función. En el siguiente ejemplo, el código llama a la función `alert()` con la cadena de caracteres definida en el recurso `error114` del archivo de propiedades predeterminadas de la configuración regional `fr_FR`:

```
alert(air.Localizer.localizer.getString("default", "error114", null, "fr_FR"));
```

El método `getString()` distribuye un evento `resourceNotFound` cuando no encuentra el recurso en el paquete especificado. La constante `air.Localizer.RESOURCE_NOT_FOUND` define la cadena `"resourceNotFound"`. El evento tiene tres propiedades: `bundleName`, `resourceName` y `locale`. La propiedad `bundleName` es el nombre del paquete en el que no se encuentra el recurso. La propiedad `resourceName` es el nombre del paquete en el que no se encuentra el recurso. La propiedad `locale` es el nombre de la configuración regional en la que no se encuentra el recurso.

El método `getString()` distribuye un evento `bundleNotFound` cuando no encuentra el paquete especificado. La constante `air.Localizer.BUNDLE_NOT_FOUND` define la cadena `"bundleNotFound"`. El evento tiene tres propiedades: `bundleName` y `locale`. La propiedad `bundleName` es el nombre del paquete en el que no se encuentra el recurso. La propiedad `locale` es el nombre de la configuración regional en la que no se encuentra el recurso.

El método `getString()` funciona de modo asíncrono (y distribuye los eventos `resourceNotFound` y `bundleNotFound` de forma asíncrona). El siguiente código define detectores para los eventos `resourceNotFound` y `bundleNotFound`:

```
air.Localizerlocalizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizerlocalizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, bnfHandler);
var str = air.Localizer.localizer.getString("default", "error114", null, "fr_FR");
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":@" + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":@" + event.locale);
}
```

El método `getFile()` del objeto `Localizer` devuelve el contenido de un paquete, en forma de cadena, para una configuración regional determinada. El archivo del paquete se lee como archivo UTF-8. El método incluye los siguientes parámetros:

Parámetro	Descripción
resourceFileName	El nombre del archivo del recurso (por ejemplo, "about.html").
templateArgs	Opcional. Un conjunto de cadenas para sustituir las etiquetas numeradas en la cadena de sustitución. Tomemos como ejemplo una llamada a la función en que el parámetro <code>templateArgs</code> es ["Raúl", "4"] y el archivo del recurso coincidente contiene dos líneas: <pre><html> <body>Hello, {0}. You have {1} new messages.</body> </html></pre> En este caso, la función devuelve una cadena de dos líneas: <pre><html> <body>Hello, Raúl. You have 4 new messages. </body> </html></pre>
locale	El código de la configuración regional (por ejemplo: "es_ES") a utilizarse. Si se facilita una configuración regional y no se encuentra ningún archivo coincidente, el método no seguirá buscando en otras configuraciones regionales de la cadena. Si no se especifica <i>ningún</i> código de configuración regional, la función devuelve el texto de la primera configuración regional de la cadena que tenga un archivo que coincida con el nombre de archivo del recurso, <code>resourceFileName</code> .

En el siguiente ejemplo, el código llama al método `document.write()` utilizando el contenido del archivo `about.html` file de la configuración regional `fr`:

```
var aboutWin = window.open();
var aboutHtml = localizer.getFile("about.html", null, "fr");
aboutWin.document.close();
aboutWin.document.write(aboutHtml);
```

El método `getFile()` distribuye un evento `fileNotFound` cuando no encuentra un recurso en la cadena de configuraciones regionales. La constante `air.Localizer.FILE_NOT_FOUND` define la cadena "resourceNotFound". El método `getFile()` funciona de modo asíncrono (y distribuye el evento `fileNotFound` de forma asíncrona). El evento tiene tres propiedades: `bundleName` y `locale`. La propiedad `fileName` es el nombre del archivo que no se encuentra. La propiedad `locale` es el nombre de la configuración regional en la que no se encuentra el recurso. El siguiente código define un detector para este evento:

```
air.Localizer.localizer.addEventListener(air.Localizer.FILE_NOT_FOUND, fnfHandler);
air.Localizer.localizer.getFile("missing.html", null, "fr");
function fnfHandler(event)
{
    alert(event.fileName + ": " + event.locale);
}
```

Localización de fechas, horas y monedas

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 "en_US", 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();
    }
}
```

Capítulo 34: Creación de aplicaciones de AIR con las herramientas de la línea de comandos

Las herramientas de la línea de comandos de Adobe® AIR™ permiten probar y empaquetar las aplicaciones de Adobe AIR. Estas herramientas pueden utilizarse también en los procesos de construcción automatizados. Las herramientas de la línea de comandos vienen incluidas en el [kit de desarrollo de software para AIR](http://www.adobe.com/go/learn_air_download_AIRSDK_es) (http://www.adobe.com/go/learn_air_download_AIRSDK_es).

Utilización de AIR Debug Launcher (ADL)

Utilice AIR Debug Launcher (ADL) para ejecutar tanto aplicaciones basadas en SWF como las basadas en HTML durante la fase de desarrollo. Con ADL se puede ejecutar una aplicación sin primero tener que empaquetarla e instalarla. De forma predeterminada, ADL utiliza un motor de ejecución incluido con el SDK, con lo cual no se necesita instalar el motor de ejecución por separado para utilizar ADL.

ADL imprime sentencias trace y errores en tiempo de ejecución a la salida estándar, pero no admite puntos de corte u otras funciones de depuración. Si desarrolla una aplicación basada en SWF, utilice Flash Debugger (o Flash CS3 o CS4) para resolver problemas complejos de depuración.

Inicio de una aplicación con ADL

Utilice la siguiente sintaxis:

```
adl [-runtime runtime-directory] [-pubid publisher-id] [-nodebug] application.xml [root-directory] [-- arguments]
```

-runtime runtime-directory Especifica el directorio que contiene el motor de ejecución a utilizar. Si no se especifica, se utiliza el directorio del motor de ejecución del mismo SDK que el programa ADL. Si se traslada ADL de su carpeta en SDK, habrá que especificar el directorio del motor de ejecución. En Windows, especifique el directorio que contiene el directorio `Adobe AIR`. En Mac OS X, especifique el directorio que contiene `Adobe AIR.framework`.

-pubid publisher-id Asigna el valor especificado como ID del editor de la aplicación de AIR para esta ejecución. La especificación de un ID de editor temporal permite ensayar las funciones de una aplicación de AIR, como la comunicación a través de una conexión local, que utilizan el ID del editor para ayudar a identificar una aplicación con exclusividad. El ID del editor final lo determina del certificado digital utilizado para firmar el archivo de instalación de AIR.

-nodebug Desactiva la compatibilidad con la depuración. Si se utiliza, el proceso de la aplicación no podrá conectar con el depurador de Flash y se suprimen los cuadros de diálogo para excepciones no controladas. Las sentencias trace continúan imprimiéndose en la ventana de la consola. Si desactivamos la depuración, el funcionamiento de la aplicación se agilizará y el modo de ejecución será más similar al de una aplicación instalada.

application.xml El archivo descriptor de la aplicación. Consulte “[Configuración de las propiedades de una aplicación de AIR](#)” en la página 44.

root-directory Especifica el directorio raíz de la aplicación a ejecutar. Si no se especifica, se utilizará el directorio que contiene el archivo descriptor de la aplicación.

-- **arguments** Las cadenas de caracteres que aparezcan después de "--" se pasan a la aplicación como argumentos de la línea de comandos.

***Nota:** cuando se intenta iniciar una aplicación de AIR que ya está ejecutándose, no se inicia una nueva instancia de la aplicación, sino que se distribuye un evento `invoke` a la instancia que está en ejecución.*

Impresión de sentencias trace

Para imprimir sentencias trace en la consola que se utiliza para ejecutar ADL, añada sentencias trace al código con la función `trace()`:

```
trace("debug message");  
air.trace("debug message");
```

Ejemplos con ADL

Ejecute una aplicación del directorio actual:

```
adl myApp-app.xml
```

Ejecute una aplicación de un subdirectorio del directorio actual:

```
adl source/myApp-app.xml release
```

Ejecute una aplicación y pase dos argumentos de la línea de comandos, "tick" y "tock":

```
adl myApp-app.xml -- tick tock
```

Ejecute una aplicación con un motor de ejecución específico:

```
adl -runtime /AIRSDK/runtime myApp-app.xml
```

Conexión a Flash Debugger (FDB)

Para depurar una aplicación de AIR basada en SWF con Flash Debugger, inicie una sesión de FDB y, acto seguido, inicie una versión de depuración de la aplicación. La versión de depuración de un archivo SWF se conecta automáticamente con una sesión FDB de detección.

1 Inicie FDB. El programa FDB se encuentra en el directorio `bin` de la carpeta del SDK de Flex.

La consola presenta el indicador de FDB: `<fdb>`

2 Ejecute el comando de ejecutar: `<fdb>run [Intro]`

3 En otra consola de comandos o de shell, inicie una versión de depuración de la aplicación:

```
adl myApp-debug.xml
```

4 Utilice comandos de FDB para definir los puntos de corte según proceda.

5 Escriba: `continue [Intro]`

Códigos de error y de salida de ADL

En el cuadro siguiente se describen los códigos de error o de salida que imprime ADL:

Código de salida	Descripción
0	Inicio satisfactorio. ADL se cierra después de cerrarse la aplicación de AIR.
1	Invocación satisfactoria de una aplicación de AIR que ya está en ejecución. ADL se cierra inmediatamente.
2	Error de uso. Los argumentos proporcionados a ADL son incorrectos.
3	No se puede encontrar el motor de ejecución.
4	No se puede iniciar el motor de ejecución. Esto se debe con frecuencia a que la versión o el nivel de revisión que se especifica en la aplicación no coincide con la versión o el nivel de revisión del motor de ejecución.
5	Se ha producido un error de causa desconocida.
6	No se puede encontrar el archivo descriptor de la aplicación.
7	El contenido del descriptor de la aplicación no es válido. Este error suele indicar que el XML no está bien conformado.
8	No se puede encontrar el archivo de contenido principal de la aplicación (especificado en el elemento <content> del archivo descriptor de la aplicación).
9	El archivo de contenido principal de la aplicación no es un archivo SWF o HTML válido.

Empaquetado de archivos de instalación de AIR con AIR Developer Tool (ADT)

El archivo de instalación de AIR, tanto para las aplicaciones de AIR basadas en SWF como para las basadas en HTML, se crean con AIR Developer Tool (ADT). (Si utiliza Adobe Flash CS3 para crear la aplicación, también puede utilizar el comando Crear archivo de AIR del menú Comandos para crear el paquete de AIR. Para obtener más información, consulte “[Actualización de Adobe AIR para Flash CS3 Professional](#)” en la página 13. Si utiliza Flash CS4, seleccione Archivo > Configuración de AIR. Para obtener más información, consulte [Publicación para Adobe AIR](#) en Utilización de Flash.

ADT es un programa en Java que se puede ejecutar desde la línea de comandos o una herramienta de construcción como Ant. Los kits de desarrollo de software de AIR y Flex incluyen scripts de la línea de comandos que le ejecutan el programa Java.

Empaquetado de archivos de instalación de AIR

Cada aplicación de AIR debe incluir, como mínimo, un archivo descriptor de la aplicación y un archivo SWF o HTML principal. Los demás componentes de la aplicación que se instalen también se deben empaquetar en el archivo de AIR.

Todos los archivos instaladores de AIR deben firmarse con un certificado digital. El instalador de AIR utiliza la firma para verificar que el archivo de la aplicación no ha sido modificado desde que se firmó. Se puede utilizar un certificado de firma de código emitido por una autoridad de certificación, como VeriSign o Thawte, o un certificado autofirmado. Un certificado emitido por una autoridad de certificación de confianza ofrece a los usuarios de la aplicación cierta seguridad respecto de su identidad como editor de la aplicación. No se puede utilizar un certificado autofirmado para verificar su identidad como firmante. (Esta desventaja también reduce la seguridad de que no se ha modificado el paquete, dado que se podría sustituir un archivo de instalación legítimo por uno falsificado antes de que lo reciba el usuario).

Se puede empaquetar y firmar un archivo de AIR en un solo paso con el comando `-package` de ADT. También se puede crear un paquete intermedio sin firmar con el comando `-prepare`, firmando después el paquete intermedio con el comando `-sign` en un paso separado.

Al firmar el paquete de instalación, ADT se pone en contacto automáticamente con el servidor de una autoridad de marcas de hora para verificar la hora. La marca de hora se incluye en el archivo de AIR. Un archivo de AIR que incluya una marca de hora verificada podrá instalarse en el futuro en cualquier momento. Si ADT no puede conectarse al servidor de marcas de hora, se cancela el empaquetado. La opción de marca de hora puede pasarse por alto, pero sin marca de hora la aplicación de AIR ya no podrá instalarse una vez caducado el certificado que se utilizó para firmar el archivo de instalación.

Si crea un paquete para actualizar una aplicación de AIR existente, el paquete debe firmarse con el mismo certificado que la aplicación original o con un certificado que tenga la misma identidad. Para tener la misma identidad, ambos certificados deben tener el mismo nombre distinguido (todos los campos informativos coinciden) y la misma cadena de certificación hasta el certificado raíz. Ello permite utilizar un certificado renovado emitido por una autoridad de certificación siempre y cuando no se cambie ninguno de los datos identificativos.

A partir de AIR 1.1, el comando `-migrate` permite migrar una aplicación y utilizar un certificado nuevo. Para la migración del certificado hay que firmar el archivo de AIR tanto con el certificado nuevo como con el anterior. La migración del certificado permite cambiar de un certificado de firma de código autofirmado a un certificado comercial, o bien de un certificado autofirmado o comercial a otro del mismo tipo. Cuando se migra un certificado, los usuarios existentes no necesitan desinstalar la aplicación existente antes de instalar la nueva versión. Las firmas de migración llevan marca de hora de forma predeterminada.

Nota: los valores definidos en el archivo descriptor de la aplicación determinan la identidad de la aplicación de AIR y su ruta de instalación predeterminada. Consulte “[Estructura del archivo descriptor de la aplicación](#)” en la página 44.

Empaquetado y firma de un archivo de AIR en un solo paso

❖ Utilice el comando `-package` con la siguiente sintaxis (en una sola línea de comandos):

```
adt -package SIGNING_OPTIONSair_fileapp_xml [file_or_dir | -C dirfile_or_dir | -e file dir ...] ...
```

SIGNING_OPTIONS Las opciones de firma identifican al almacén de claves que contiene la clave privada y el certificado que se utilizaron para firmar el archivo de AIR. Para firmar una aplicación de AIR con un certificado autofirmado generado por ADT, las opciones a utilizar son:

```
-storetype pkcs12 -keystore certificate.p12
```

En este ejemplo, `certificate.p12` es el nombre del archivo del almacén de claves. (ADT le solicita la contraseña, dado que no se indica en la línea de comandos). Las opciones de firma se describen con detenimiento en “[Opciones de firma en la línea de comandos de ADT](#)” en la página 330.

air_file El nombre del archivo de AIR que se va a crear.

app_xml La ruta al archivo descriptor de la aplicación. La ruta se puede indicar en relación con el directorio actual o como ruta absoluta. (En el archivo de AIR el archivo descriptor de la aplicación cambia de nombre a “application.xml”).

file_or_dir Los archivos y directorios a empaquetar en el archivo de AIR. Se puede especificar la cantidad de archivos y directorios que se desee, delimitados por un espacio en blanco. Si se incluye un directorio en la lista, se añaden al paquete todos los archivos y subdirectorios que contenga, excepto los archivos ocultos. (Si se especifica el archivo descriptor de la aplicación, sea directamente o mediante el uso de un comodín o por expansión de un directorio, éste se pasa por alto y no se añade al paquete por segunda vez). Los archivos y directorios especificados deben estar incluidos en el directorio actual o en uno de sus subdirectorios. Para cambiar el directorio actual, utilice la opción `-c`.

Importante: No se pueden utilizar comodines en los argumentos `file_or_dir` después de la opción `-c`. (Los shells de comandos expanden los comodines antes de pasar los argumentos a ADT, por lo que ADT buscaría los archivos en el lugar equivocado). Sí se puede utilizar el punto (".") para representar el directorio actual. Por ejemplo: `"-c assets ."` copia todo el contenido del directorio `assets`, incluidos los subdirectorios, hasta el nivel raíz del paquete de la aplicación.

`-c dir` Cambia el directorio de trabajo al valor `dir` antes de procesar los archivos y directorios posteriores que se añadan al paquete de la aplicación. Los archivos o directorios se añaden a la raíz del paquete de la aplicación. La opción `-c` se puede utilizar todas las veces que se desee para incluir archivos desde varios lugares del sistema de archivos. Si se especifica una ruta relativa para `dir`, la ruta siempre se resuelve desde el directorio de trabajo original.

A medida que ADT procesa los archivos y directorios incluidos en el paquete, se van guardando las rutas relativas entre el directorio actual y los archivos de destino. Estas rutas se expanden en la estructura de directorios de la aplicación cuando se instala el paquete. Por lo tanto, si se especifica `-c release/bin lib/feature.swf`, el archivo `release/bin/lib/feature.swf` se coloca en el subdirectorio `lib` de la carpeta raíz de la aplicación.

`-e file dir` Coloca el archivo especificado en el directorio especificado del paquete.

Nota: el elemento `<content>` del archivo descriptor de la aplicación debe especificar la ubicación final del archivo principal de la aplicación dentro del árbol de directorios del paquete de la aplicación.

Ejemplos con ADT

Empaquete determinados archivos de la aplicación del directorio actual:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf components.swc
```

Empaquete todos los archivos y subdirectorios del directorio de trabajo actual:

```
adt -package -storetype pkcs12 -keystore ../cert.p12 myApp.air myApp.xml .
```

Nota: el archivo del almacén de claves contiene la clave privada que se utilizó para firmar la aplicación. No incluya nunca el certificado de firma en el paquete de AIR. Si utiliza comodines en el comando ADT, coloque el archivo del almacén de claves en otra carpeta para que no se incluya en el paquete. En este ejemplo el archivo del almacén de claves, `cert.p12`, reside en el directorio principal.

Empaquete solamente los principales archivos y un subdirectorio de imágenes:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf images
```

Empaquete el archivo `application.xml` y el archivo SWF principal que se encuentran en un directorio de trabajo (`release/bin`):

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml -C release/bin myApp.swf
```

Empaquete componentes procedentes de más de un lugar en el sistema de archivos de construcción. En este ejemplo, los componentes de la aplicación se encuentran en las siguientes carpetas antes de que se empaquetan:

```
/devRoot
  /myApp
    /release
      /bin
        myApp.xml
        myApp.swf
    /artwork
      /myApp
        /images
          image-1.png
          ...
          image-n.png
    /libraries
      /release
        /libs
          lib-1.swf
          ...
          lib-n.swf
          AIRAliases.js
```

La ejecución del siguiente comando ADT desde el directorio /devRoot/myApp:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml
  -C release/bin myApp.swf
  -C ../artwork/myApp images
  -C ../libraries/release libs
```

produce un paquete con la siguiente estructura:

```
/myAppRoot
  /META-INF
    /AIR
      application.xml
      hash
  myApp.swf
  mimetype
  /images
    image-1.png
    ...
    image-n.png
  /libs
    lib-1.swf
    ...
    lib-n.swf
    AIRAliases.js
```

Ejecute ADT como programa de Java (sin definir la ruta de clase):

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air
myApp.xml myApp.swf
```

Ejecute ADT como programa de Java (con la ruta de clase de Java definida para incluir el paquete ADT.jar):

```
java com.adobe.air.ADT -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml
myApp.swf
```


Opciones de firma en la línea de comandos de ADT

ADT utiliza Java Cryptography Architecture (JCA) para acceder a las claves privadas y los certificados para firmar aplicaciones de AIR. Las opciones de firma identifican el almacén de claves, así como la clave privada y el certificado que en él se encuentran.

El almacén de claves debe incluir tanto la clave privada como la cadena de certificación asociada. La cadena de certificación sirve para establecer el ID del editor de la aplicación. Si el certificado de firma está encadenado con un certificado de confianza en un ordenador, el nombre común del certificado se presenta como el nombre del editor en el cuadro de diálogo de la instalación de AIR.

ADT exige que el certificado cumpla con la norma x509v3 (RFC3280) e incluya la extensión de uso mejorado de claves con los valores correspondientes para la firma de códigos. Se respetan las limitaciones propias del certificado y éstas podrían descartar el uso de algunos certificados para firmar aplicaciones de AIR.

***Nota:** ADT utiliza la configuración de proxy para el entorno de ejecución de Java (JRE), si procede, para conectarse a recursos de Internet con la finalidad de comprobar listas de revocación de certificados y obtener marcas de hora. Si tiene algún problema con la conexión a recursos de Internet cuando utiliza ADT y la red requiere una configuración de proxy concreta, es posible que necesite ajustar la configuración de proxy para JRE.*

Especificación de las opciones de firma de AIR

- ❖ Para especificar las opciones de firma de ADT para los comandos `-package` y `-prepare`, utilice la sintaxis siguiente:

```
[-alias aliasName] [-storetype type] [-keystore path] [-storepass password1] [-keypass password2] [-providerName className] [-tsa url]
```

-alias *aliasName*: el alias de una clave en el almacén de claves. No es necesario especificar un alias si el almacén de claves contiene un solo certificado. Si no se especifica ningún alias, ADT utiliza la primera clave del almacén de claves.

No todas las aplicaciones con administración del almacén de claves permiten que se asigne un alias a los certificados. Si hace uso del almacén de claves del sistema en Windows, por ejemplo, utilice como alias el nombre distinguido del certificado. Se puede utilizar la utilidad Java Keytool para enumerar los certificados disponibles para poder determinar el alias. Por ejemplo, si se ejecuta el comando:

```
keytool -list -storetype Windows-MY
```

se produce una salida para un certificado como ésta:

```
CN=TestingCert,OU=QE,O=Adobe,C=US, PrivateKeyEntry,  
Certificate fingerprint (MD5): 73:D5:21:E9:8A:28:0A:AB:FD:1D:11:EA:BB:A7:55:88
```

Para hacer referencia a este certificado en la línea de comandos de ADT, defina el alias en:

```
CN=TestingCert,OU=QE,O=Adobe,C=US
```

En Mac OS X, el alias de un certificado en la Llave es el nombre que se muestra en la aplicación Acceso a Llaves.

-storetype *type*: el tipo de almacén de claves, determinado por la implementación del almacén de claves. La implementación del almacén de claves predeterminada incluida con la mayoría de las instalaciones de Java es compatible con los tipos `JKS` y `PKCS12`. Java 5.0 ofrece compatibilidad con el tipo `PKCS11`, para acceder a almacenes de claves en tokens de hardware, y al tipo `Keychain`, para acceder a la Llave de Mac OS X. Java 6.0 ofrece compatibilidad con el tipo `MSCAPI` (en Windows). Si se han instalado y configurado otros proveedores de JCA, puede que se disponga además de otros tipos de almacenes de claves. Si no se especifica ningún tipo de almacén de claves, se utiliza el tipo predeterminado para el proveedor de JCA predeterminado.

Tipo de almacén	Formato del almacén de claves	Versión mínima de Java
JKS	Archivo de almacén de claves de Java (.keystore)	1.2
PKCS12	Archivo PKCS12 (.p12 o .pfx)	1.4
PKCS11	Token de hardware	1.5
KeychainStore	Llave de Mac OS X	1.5
Windows-MY o Windows-ROOT	MSCAPI	1.6

-keystore path : la ruta al archivo del almacén de claves para tipos de almacén basados en archivo.

-storepass password1 : la contraseña para acceder al almacén de claves. Si no se especifica, ADT la solicita.

-keypass password2 : la contraseña para acceder a la clave privada que se utiliza para firmar la aplicación de AIR. Si no se especifica, ADT la solicita.

-providerName className : el proveedor de JCA para el tipo de almacén de claves especificado. Si no se especifica, ADT utiliza el proveedor predeterminado para ese tipo de almacén de claves.

-tsa url : especifica la URL de un [RFC3161](#) para marcar la hora en la firma digital. Si no se especifica una URL, se utiliza un servidor de marcas de hora predeterminado suministrado por Geotrust. Cuando la firma de una aplicación de AIR lleva una marca de hora, la aplicación puede instalarse después de caducado el certificado de firma porque la marca de hora verifica que el certificado era válido al momento de firmar.

Si ADT no puede conectarse al servidor de marcas de hora, se cancela la firma y no se produce ningún paquete. La marca de hora se puede desactivar especificando `-tsa none`. Sin embargo, una aplicación de AIR empaquetada sin marca de hora no se puede instalar una vez caducado el certificado de firma.

***Nota:** las opciones de firma son como las opciones equivalentes de la utilidad Keytool de Java. La utilidad Keytool sirve para examinar y administrar almacenes de claves en Windows. La utilidad de seguridad de Apple® sirve para lo mismo en Mac OS X.*

Ejemplos de opciones de firma

Firma con un archivo .p12:

```
-storetype pkcs12 -keystore cert.p12
```

Firma con el almacén de claves de Java predeterminado:

```
-alias AIRcert -storetype jks
```

Firma con un almacén de claves de Java específico:

```
-alias AIRcert -storetype jks -keystore certStore.keystore
```

Firma con la Llave de Mac OS X:

```
-alias AIRcert -storetype KeychainStore -providerName Apple
```

Firma con el almacén de claves del sistema de Windows:

```
-alias cn=AIRCert -storetype Windows-MY
```

Firma con un token de hardware (consulte las instrucciones del fabricante del token para ver cómo se configura Java para utilizar el token y para obtener el valor correcto de `providerName`):

```
-alias AIRCert -storetype pkcs11 -providerName tokenProviderName
```

Firma sin incorporar una marca de hora:

```
-storetype pkcs12 -keystore cert.p12 -tsa none
```

Creación de archivos intermedios sin firmar de AIR con ADT

Utilice el comando `-prepare` para crear un archivo intermedio de AIR sin firmar. Para producir un archivo de instalación de AIR válido, el archivo intermedio de AIR debe firmarse con el comando `-sign` de ADT.

El comando `-prepare` acepta los mismos indicadores y parámetros que el comando `-package` (a excepción de las opciones de firma). La única diferencia es que no se firma el archivo de salida. El archivo intermedio se genera con la extensión del nombre de archivo `airi`.

Para firmar un archivo intermedio de AIR, utilice el comando `-sign` de ADT. (Consulte [Firma de un archivo intermedio de AIR con ADT](#)).

Ejemplo con ADT

```
adt -prepare unsignedMyApp.airi myApp.xml myApp.swf components.swc
```

Firma de un archivo intermedio de AIR con ADT

Para firmar un archivo intermedio de AIR con ADT, utilice el comando `-sign`. El comando "sign" sólo funciona con archivos intermedios de AIR (con la extensión `airi`). Un archivo de AIR no se puede firmar por segunda vez.

Para crear un archivo intermedio de AIR, utilice el comando de ADT `-sign`. (Consulte "Creación de archivos intermedios sin firmar de AIR con ADT" en la página 332)

Firma de archivos AIRI

❖ Utilice el comando `-sign` de ADR con la sintaxis siguiente:

```
adt -sign SIGNING_OPTIONS airi_file air_file
```

SIGNING_OPTIONS Las opciones de firma identifican la clave privada y el certificado con que se firma el archivo de AIR. Estas opciones se describen en "Opciones de firma en la línea de comandos de ADT" en la página 330.

airi_file La ruta al archivo intermedio sin firmar de AIR que se va a firmar.

air_file El nombre del archivo de AIR que se va a crear.

Ejemplo con ADT

```
adt -sign -storetype pkcs12 -keystore cert.p12 unsignedMyApp.airi myApp.air
```

Para obtener más información, consulte "Firma digital de archivos de AIR" en la página 301.

Firma de un archivo AIR para cambiar el certificado de la aplicación

Para actualizar una aplicación de AIR existente de modo que pueda utilizar un nuevo certificado de firma, utilice el comando `-migrate` de ADT.

La migración de certificados puede resultar de utilidad en las siguientes situaciones:

- Actualización de un certificado autofirmado a un certificado emitido por una autoridad de certificación
- Cambio de un certificado autofirmado que está por caducar a un nuevo certificado autofirmado
- el cambio de un certificado comercial a otro (por ejemplo, cuando cambia de identidad la empresa).

Para poder aplicar una firma de migración, el certificado original debe aún tener validez. Una vez caducado el certificado, no se puede aplicar una firma de migración. Los usuarios de la aplicación tendrán que desinstalar la versión existente antes de poder instalar la versión actualizada. Obsérvese que la firma de migración lleva una marca de hora, de forma predeterminada, de modo que las actualizaciones de AIR firmadas con firma de migración seguirán siendo válidas incluso una vez caducado el certificado.

***Nota:** si se renueva un certificado de emisión comercial, no suele ser necesario migrar el certificado. Un certificado renovado conserva la misma identidad del editor que el original, a menos que haya cambiado el nombre distinguido. Para obtener una lista completa de los atributos del certificado que se utilizan para determinar el nombre distinguido, consulte “Identificador del editor de AIR” en la página 302.*

Para migrar la aplicación y utilizar un nuevo certificado:

- 1 Cree una actualización de la aplicación.
- 2 Empaquete y firme el archivo de AIR de actualización con el certificado **nuevo**.
- 3 Vuelva a firmar el archivo de AIR con el certificado **original** utilizando el comando `-migrate`.

Un archivo de AIR firmado con el comando `-migrate` puede utilizarse tanto para instalar una nueva versión de la aplicación como para actualizar las versiones anteriores, incluidas las que fueron firmadas con el certificado anterior.

Migración de una aplicación de AIR para utilizar un nuevo certificado

❖ Utilice el comando `-migrate` de ADT con la sintaxis siguiente:

```
adt -migrate SIGNING_OPTIONSair_file_inair_file_out
```

SIGNING_OPTIONS Las opciones de firma identifican la clave privada y el certificado con que se firma el archivo de AIR. Estas opciones deben identificar el certificado de firma **original** y se describen en “[Opciones de firma en la línea de comandos de ADT](#)” en la página 330

air_file_in El archivo de AIR para la actualización, firmado con el certificado **nuevo**.

air_file_out El archivo de AIR que se va a crear.

Ejemplo con ADT

```
adt -migrate -storetype pkcs12 -keystore cert.p12 myApp.air myApp.air
```

Para obtener más información, consulte “[Firma digital de archivos de AIR](#)” en la página 301.

***Nota:** el comando `-migrate` se añadió a ADT en la versión 1.1 de AIR.*

Creación de certificados autofirmados con ADT

Los certificados autofirmados permiten producir un archivo de instalación de AIR válido, pero sólo ofrecen una garantía de seguridad limitada al usuario dado que no se puede verificar la autenticidad de los certificados autofirmados. Cuando se instala un archivo de AIR autofirmado, los datos del editor se presentan al usuario como “Desconocidos”. Un certificado generado por ADT tiene una validez de cinco años.

Si se crea una actualización para una aplicación de AIR que se firmó con un certificado autogenerado, habrá que utilizar el mismo certificado para firmar tanto los archivos de AIR originales como los de la actualización. Los certificados que produce ADT son siempre únicos, aunque se utilicen los mismos parámetros. Por lo tanto, si desea autofirmar las actualizaciones con un certificado generado por ADT, conserve el certificado original en un lugar seguro. Por otra parte, no podrá producir un archivo de AIR actualizado después de haber caducado el certificado generado por ADT. (Sí podrá publicar nuevas aplicaciones con otro certificado, pero no las nuevas versiones de la misma aplicación).

Importante: Dadas las limitaciones de los certificados autofirmados, Adobe recomienda enfáticamente utilizar un certificado comercial emitido por una autoridad de certificación de renombre, como VeriSign o Thawte, para firmar aplicaciones de AIR que se distribuyen de forma pública.

El certificado y la clave privada asociada generados por ADT se guardan en un archivo de almacén de claves del tipo PKCS12. La contraseña especificada se establece en la propia clave y no en el almacén de claves.

Generación de un certificado de ID digital para autofirmar archivos de AIR

❖ Utilice el comando `-certificate` del ADT (en una sola línea de comandos):

```
adt -certificate -cn name [-ou org_unit] [-o org_name] [-c country] key_type pfx_file password
```

-cn name La cadena asignada como nombre común del nuevo certificado.

-ou org_unit Una cadena asignada como unidad organizativa emisora del certificado. (Opcional.)

-o org_name Una cadena asignada como organización emisora del certificado. (Opcional.)

-c country Código de país de dos letras de la norma ISO-3166. Si el código suministrado no es válido, no se genera el certificado. (Opcional.)

key_type El tipo de clave a utilizar para el certificado, que puede ser "1024-RSA" o "2048-RSA".

pfx_file La ruta del archivo del certificado que se va a generar.

password La contraseña para el nuevo certificado. Cuando se firmen archivos de AIR con este certificado, se necesitará la contraseña.

Ejemplos de generación de certificados

```
adt -certificate -cn SelfSign -ou QE -o "Example, Co" -c US 2048-RSA newcert.p12 39#wnetx3t1
adt -certificate -cn ADigitalID 1024-RSA SigningCert.p12 39#wnetx3t1
```

Cuando se utilizan estos certificados para firmar archivos de AIR, hay que usar las siguientes opciones de firma con los comandos `-package` o `-prepare` de ADT:

```
-storetype pkcs12 -keystore newcert.p12 -keypass 39#wnetx3t1
-storetype pkcs12 -keystore SigningCert.p12 -keypass 39#wnetx3t1
```

Utilización de Apache Ant con las herramientas del SDK

Este tema proporciona ejemplos del uso de la herramienta de construcción Apache Ant para probar y empaquetar aplicaciones de AIR.

Nota: esta discusión no tiene por objetivo facilitar una reseña exhaustiva de Apache Ant. Para obtener documentación de Ant, visite <http://Ant.Apache.org>.

Utilización de Ant para proyectos sencillos

Este ejemplo ilustra la creación de una aplicación de AIR con Ant y las herramientas de la línea de comandos de AIR. Se utiliza una estructura sencilla de proyecto con todos los archivos guardados en un solo directorio.

Para que resulte más fácil reutilizar el script de creación, en estos ejemplos se utilizan varias propiedades definidas. Una de las series de propiedades identifica los lugares donde están instaladas las herramientas de la línea de comandos:

```
<property name="SDK_HOME" value="C:/Flex3SDK"/>
<property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
<property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>
```

La segunda serie de propiedades es propia del proyecto. Estas propiedades adoptan una nomenclatura en la que los nombres del archivo descriptor de la aplicación y los archivos de AIR se basan en el archivo raíz de origen. Pueden utilizarse otras nomenclaturas sin problema.

```
<property name="APP_NAME" value="ExampleApplication"/>
<property name="APP_ROOT" value="."/>
<property name="APP_DESCRIPTOR" value="${APP_ROOT}/${APP_NAME}-app.xml"/>
<property name="AIR_NAME" value="${APP_NAME}.air"/>
<property name="STORETYPE" value="pkcs12"/>
<property name="KEYSTORE" value="ExampleCert.p12"/>
```

Invocación de ADL para probar una aplicación

Para ejecutar la aplicación con ADL, utilice una tarea "exec":

```
<target name="test" depends="compile">
<target name="test">
  <exec executable="${ADL}">
    <arg value="${APP_DESCRIPTOR}"/>
  </exec>
</target>
```

Invocación de ADT para empaquetar una aplicación

Para empaquetar la aplicación, utilice una tarea de Java para ejecutar la herramienta adt.jar:

```
<target name="package">
  <java jar="${ADT.JAR}" fork="true" failonerror="true">
    <arg value="-package"/>
    <arg value="-storetype"/>
    <arg value="${STORETYPE}"/>
    <arg value="-keystore"/>
    <arg value="${KEYSTORE}"/>
    <arg value="${AIR_NAME}"/>
    <arg value="${APP_DESCRIPTOR}"/>
    <arg value="${APP_NAME}.swf"/>
    <arg value="*.png"/>
  </java>
</target>
```

Si la aplicación tiene más archivos que empaquetar, se pueden añadir más elementos <arg>.

Utilización de Ant para proyectos más complejos

La estructura de directorios de una aplicación típica es más compleja que un solo directorio. El siguiente ejemplo ilustra un archivo de construcción que se utiliza para compilar, probar y empaquetar una aplicación de AIR cuya estructura de directorios es más práctica para un proyecto.

Este proyecto sencillo guarda en un directorio `src` los archivos de origen de la aplicación y otros componentes como archivos de iconos. El script de creación genera los siguientes directorios de trabajo:

build Guarda las versiones de publicación (no de depuración) de los archivos SWF compilados.

debug Guarda la versión de depuración de la aplicación sin empaquetar, incluidos los archivos SWF y de componentes. La utilidad ADL ejecuta la aplicación desde este directorio.

release Guarda el paquete final de AIR.

Para las herramientas de AIR hay que utilizar algunas opciones adicionales al trabajar en archivos sin estar en el directorio actual de trabajo:

Pruebas El segundo argumento que se pasa a ADL especifica el directorio raíz de la aplicación de AIR. Para especificar el directorio raíz de la aplicación se añade la siguiente línea a la tarea de pruebas:

```
<arg value="\${debug}"/>
```

Empaquetado El empaquetado de archivos situados en subdirectorios que no deben formar parte de la estructura del paquete final requiere el uso de la directiva `-C` para cambiar el directorio de trabajo de ADT. Cuando se utiliza la directiva `-C`, los archivos y directorios del nuevo directorio de trabajo se copian al nivel raíz del archivo del paquete de AIR. Así, `-C build file.png` copia `file.png` en la raíz del directorio de la aplicación. Asimismo, `-C assets icons` copia la carpeta de iconos en el nivel raíz y también copia todos los archivos y directorios que se encuentren en la carpeta de iconos. Por ejemplo, la siguiente secuencia de líneas de la tarea de empaquetado añade el directorio de iconos directamente en el nivel raíz del archivo del paquete de la aplicación:

```
<arg value="-C"/>
<arg value="\${assets}"/>
<arg value="icons"/>
```

***Nota:** si hay que cambiar muchos recursos y componentes a distintas ubicaciones relativas, suele resultar más fácil reunirlos en un directorio temporal con tareas de Ant que crear una lista compleja de argumentos para ADT. Una vez organizados los recursos, se puede utilizar una lista sencilla de argumentos de ADT para empaquetarlos.*

```
<project>
  <!-- SDK properties -->
  <property name="SDK_HOME" value="C:/Flex3SDK"/>
  <property name="ADL" value="\${SDK_HOME}/bin/adl.exe"/>
  <property name="ADT.JAR" value="\${SDK_HOME}/lib/adt.jar"/>

  <!-- Project properties -->
  <property name="PROJ_ROOT_DIR" value="."/>
  <property name="APP_NAME" value="ExampleApplication"/>
  <property name="APP_ROOT_DIR" value="."/>
  <property name="APP_ROOT_FILE" value="\${APP_NAME}.swf"/>
  <property name="APP_DESCRIPTOR" value="\${PROJ_ROOT_DIR}/\${APP_NAME}-app.xml"/>
  <property name="AIR_NAME" value="\${APP_NAME}.air"/>
  <property name="release" location="\${PROJ_ROOT_DIR}/release"/>
  <property name="assets" location="\${PROJ_ROOT_DIR}/src/assets"/>
  <property name="STORETYPE" value="pkcs12"/>
  <property name="KEYSTORE" value="ExampleCert.p12"/>

  <target name="init" depends="clean">
    <mkdir dir="\${release}"/>
  </target>

  <target name="test">
    <exec executable="\${ADL}">
      <arg value="\${APP_DESCRIPTOR}"/>
    </exec>
  </target>
</project>
```

```
        <arg value="\${APP_ROOT_DIR}"/>
    </exec>
</target>

<target name="package" depends="init">
    <java jar="\${ADT.JAR}" fork="true" failonerror="true">
        <arg value="-package"/>
        <arg value="-storetype"/>
        <arg value="\${STORETYPE}"/>
        <arg value="-keystore"/>
        <arg value="\${KEYSTORE}"/>
        <arg value="\${release}/${AIR_NAME}"/>
        <arg value="\${APP_DESCRIPTOR}"/>
        <arg value="-C"/>
        <arg value="\${APP_ROOT_DIR}"/>
        <arg value="\${APP_ROOT_FILE}"/>
        <arg value="-C"/>
        <arg value="\${assets}"/>
        <arg value="icons"/>
    </java>
</target>

<target name="clean" description="clean up">
    <delete dir="\${release}"/>
</target>
</project>
```


Índice

Símbolos

- ? (signo de interrogación) carácter, en parámetros SQL sin nombre 176
- (?), signo de interrogación carácter, en parámetros SQL sin nombre 176
- @ (arroba), en los nombres de parámetros de declaración SQL 175
- \
 - (dos puntos), en los nombres de parámetros de declaración SQL 175

Numéricos

- 1024-RSA 334
- 2048-RSA 334

A

- AC_FL_RunContent(), función (en default_badge.html) 294
- AC_RuntimeActiveContent.js 294
- acceptDragDrop(), método (clase NativeDragManager) 128, 133
- acompc, compilador 229
- Acrobat 203, 256
- ActionScript
 - objetos de visualización 68
 - reutilizar scripts JavaScript 225
- activas
 - ventanas 69
- activate(), método (clase NativeWindow) 64, 69, 70
- active, evento 75
- activeWindow, propiedad (clase NativeApplication) 69
- actividad (usuario), detectar 281
- actividad del usuario, detectar 281
- actualización personalizada, interfaz de usuario 310
- actualizar aplicaciones de AIR 51
- addChild(), método (clase Stage) 67
- addChildAt(), método (clase Stage) 67
- administración de derechos digitales 261
- Adobe AIR
 - actualizar 23
 - desinstalar 2
 - instalar 1, 23
 - introducción 7
 - nuevas funciones 53
- Adobe Media Player 261

- Adobe Reader 203, 256
- Adobe, documentación 9
- AIR Debug Launcher (ADL)
 - códigos de error y de salida 325
- AIR Developer Tool (ADT)
 - archivos AIRI 332
 - crear certificados autofirmados 333
 - empaquetar archivos de AIR 326
 - opciones de firma 330
- AIR, archivos
 - empaquetar 326
 - firmar 301
- air, propiedad (archivo AIRAliases.js) 202, 223
- AIRAliases.js, archivo 202, 223
- AIRI, archivos
 - crear con AIR Developer Tool (ADT) 332
- AIRLocalizer.js, archivo 314
- Ajax
 - compatibilidad en el entorno limitado de la aplicación 33
 - seguridad 33
- allowBrowserInvocation, elemento (archivo descriptor de la aplicación) 51, 271, 274
- allowCrossDomainXHR, atributo (elementos frame e iframe) 206, 211
- allowLoadBytesCodeExecution, propiedad (clase LoaderContext) 41
- almacenes de claves 330, 334
- alwaysInFront, propiedad (clase NativeWindow) 70
- API de archivos 101
- API del sistema de archivos 101
- aplicaciones
 - Consulte* aplicaciones de AIR
- aplicaciones de AIR
 - actualizar 23, 51, 308
 - asociaciones de tipos de archivos 51, 273, 280
 - cerrar 271
 - configurar 44, 46, 278
 - desinstalar 26
 - detectar la instalación 298
 - distribuir 292
 - ejecutar 292, 300
 - iconos 50
 - información de copyright 48
 - iniciar 271
 - instalar 23, 292
 - invocar 271
 - invocar desde el navegador 51
 - ruta de instalación 46
 - salir 271
 - versiones 46, 281, 309
- aplicaciones de AIR, actualizar 308
- aplicaciones de Internet enriquecidas (RIA) 7
- aplicaciones de muestra 2
- app, esquema de URL 39, 42, 65, 107, 204, 224, 232, 257
- AppInstallDisabled (parámetros del registro de Windows) 26
- applicationDescriptor, propiedad (clase NativeApplication) 278
- ApplicationDomain, clase 226
- applicationStorageDirectory, propiedad (clase File) 103
- app-storage
 - esquema de URL 42
- app-storage, esquema de URL 25, 39, 107, 257
- app-support, esquema de URL 232
- archivo descriptor de la aplicación 44
- leer 278
- archivos
 - base de datos 165
 - compatibilidad con arrastrar y colocar 127
 - compatibilidad con copiar y pegar 143
 - copiar 114
 - eliminar 115
 - escribir 116
 - leer 116
 - mover 114
 - referencias 105
- archivos de AIR, firmar 326
- archivos SWF
 - cargar a través de una etiqueta de script 228
- archivos temporales 116
- argumentos de la línea de comandos, captura 272
- arguments, propiedad
 - clase BrowserInvokeEvent 275
 - clase InvokeEvent 272

- arrastrar y colocar
 - clases asociadas 128
 - compatibilidad con Flex 129
 - comportamiento predeterminado en HTML 135
 - efectos del cursor 133, 137
 - en contenido de entorno limitado ajeno a la aplicación (en HTML) 141
 - eventos en HTML 135
 - formatos de transferencia 127
 - gestos 127
 - HTML 135, 208
 - teclas modificadoras 133
- arroba (@), en los nombres de parámetros de declaración SQL 175
- asfunction, protocolo 28
- asistencia técnica 9
- asociaciones de tipos de archivos 51, 273, 280
- attach(), método (clase SQLConnection) 186
- autoExit, propiedad
 - clase NativeApplication 276
- AUTOINCREMENT, columnas (SQL) 185
- autoridades de certificación (AC) 301

- B**
- bandeja del sistema, iconos 82, 85
 - compatibilidad 96
- barra de herramientas (Mac OS) 62
- barra de tareas, iconos 70, 96
- barra de título, iconos (Windows) 62
- barras de menú 83
- bases de datos
 - archivos 165
 - cambiar datos 186
 - campos 166
 - clases usadas 167
 - claves principales 184, 185
 - columnas 166
 - conectarse 172
 - crear 169
 - eliminar datos 186
 - en memoria 169
 - errores 186
 - estructura 166
 - filas 166
 - identificadores de fila 185
 - información 165
 - introducir datos 176, 189
 - modo asíncrono 168
 - modo sincrónico 168
 - recuperar datos 177
 - rendimiento 176
 - seguridad 176
 - tablas 166, 170
 - usar varias 186
 - usos 165
- bases de datos en memoria 169
- bases de datos locales
 - Consulte bases de datos
- bases de datos relacionales
 - Consulte bases de datos
- big-endian, orden de bytes 156
- bitmaps, propiedad (clase Icon) 96
- bounce(), método (clase Icon) 97
- browseForDirectory(), método (clase File) 105
- browseForOpen(), método (clase File) 106
- browseForSave(), método (clase File) 106
- browserInvoke, evento 275, 300
- BrowserInvokeEvent, clase 274
- ByteArray, clase
 - constructor 154
 - método compress() 157
 - método readBytes() 154
 - método readFloat() 154
 - método readInt() 154
 - método readObject() 154
 - método readUTFBytes() 154
 - método uncompress() 157
 - método writeBytes() 154
 - método writeFloat() 154
 - método writeInt() 154
 - método writeObject() 154
 - método writeUTFBytes() 154
 - propiedad bytesAvailable 155
 - propiedad length 155
 - propiedad position 155
- bytesAvailable, propiedad (clase ByteArray) 155

- C**
- caché entre dominios. seguridad 29
- cambiar tamaño de ventanas 50, 59, 73
- campos (base de datos) 166
- Canvas, objeto 206, 213
- Capabilities
 - propiedad language 313
 - propiedad languages 313
- Capabilities, clase
 - propiedad playerType 281
- caracteres de letras de selección
 - elementos de menú 85
- Centro de desarrollo de Adobe Acrobat 257
- cerrar aplicaciones 276
- cerrar aplicaciones de AIR 271
- cerrar ventanas 59, 276
- certificados
 - autoridades (CA) 43
 - cadena 306
 - caducidad 303
 - cambiar 305, 332
 - firmar código 43
 - firmar para archivos de AIR 301
 - formatos 302
 - migrar 305, 332
 - opciones de la línea de comandos de ADT 330
- certificados autofirmados 43, 301, 333
- certificados de desarrolladores de AIR 303
- childSandboxBridge, propiedad
 - clase LoaderInfo 36
 - objeto Window 31
- cifrado 261
- cifrado AES-CBC de 128 bits 199
- cifrado de contenido de vídeo 261
- cifrado de datos 199
- clase NativeMenuItem
 - propiedad label 148
- claves principales
 - bases de datos 184
- claves privadas 330
- clearData(), método
 - objeto ClipboardData 207
 - objeto DataTransfer 136, 208
- clientX, propiedad (eventos drag en HTML) 136
- clientY, propiedad (eventos drag en HTML) 136
- Clipboard 207
 - copiar y pegar 143
 - formatos de datos 149, 150
 - seguridad 144
 - sistema 143
- Clipboard, clase
 - método getData() 129, 133
 - método setData() 151
 - método setDataHandler() 151
 - propiedad generalClipboard 143
- clipboard, evento 208

- clipboard, propiedad (clase NativeDragEvent) 133
 - clipboardData, propiedad (eventos clipboard) 208
 - clipboardData, propiedad (eventos de copiar y pegar en HTML) 145
 - ClipboardFormats, clase 149
 - ClipboardTransferModes, clase 150
 - close, evento 75
 - close(), método
 - clase NativeWindow 71
 - close(), método (objeto window) 58
 - closing, evento 71, 75, 239, 276
 - códigos de error
 - DRM 268
 - códigos de salida (ADL) 325
 - columnas (base de datos) 166
 - Comando, tecla 84
 - comandos, menú
 - consulte* elementos de menú
 - Compatibilidad con base de datos SQLite
 - Véase también bases de datos* <\$<\$nopage 164
 - compatibilidad con base de datos SQLite 164
 - complete, evento 226, 230, 236
 - compress(), método (clase ByteArray) 157
 - CompressionAlgorithm, clase 157
 - conectarse a una base de datos 172
 - configuraciones regionales, elegir para una aplicación 313
 - conjuntos de bytes
 - orden de bytes 156
 - posición 155
 - tamaño 155
 - constructores de función (en JavaScript) 205
 - contenido PDF
 - cargar 257
 - cómo añadirlo a aplicaciones de AIR 256
 - comunicación en JavaScript 257
 - limitaciones conocidas 259
 - contenido SWF
 - en HTML 203
 - content, elemento (archivo descriptor de la aplicación) 49
 - contenteditable, atributo (HTML) 139
 - ContextMenu, clase 84, 87
 - contextmenu, evento 88
 - ContextMenuEvent, clase
 - propiedad contextMenuOwner 87
 - propiedad mouseTarget 87
 - ContextMenuEvent, clase 84
 - contextMenuOwner, propiedad (clase ContextMenuEvent) 87
 - contraseñas
 - configurar para contenido de medios cifrados 261
 - Control, tecla 84
 - cookies 207
 - copiar archivos 114
 - copiar directorios 112
 - copiar y pegar
 - comandos de menú 147
 - elementos de menú predeterminados (Mac OS) 148
 - HTML 145, 207
 - métodos abreviados de teclado 147
 - modos de transferencia 150
 - representación aplazada 151
 - copy, evento 145
 - copy(), clase (clase NativeApplication) 147
 - copyTo(), método (clase File) 114
 - copyToAsync(), método (clase File) 114
 - crear directorios 111
 - CREATE TABLE, declaración (SQL) 170
 - createDirectory(), método (clase File) 111
 - createElement(), método (objeto Document) 222
 - createRootWindow(), método (clase HTMLLoader) 64, 66, 217
 - createTempDirectory(), método (clase File) 111, 116
 - createTempFile(), método (clase File) 116
 - creationDate, propiedad (clase File) 113
 - creator, propiedad (clase File) 113
 - credenciales
 - para contenido con cifrado DRM 268
 - credenciales de usuario y seguridad 42
 - CSS
 - acceder a estilos HTML de ActionScript 230
 - extensiones en AIR 215
 - currentDirectory, propiedad (clase InvokeEvent) 272
 - currentDomain, propiedad (clase ApplicationDomain) 226
 - cursor, efectos de arrastrar y colocar 133, 137
 - customItems, propiedad (clase ContextMenu) 87
 - customUpdateUI, elemento (archivo descriptor de la aplicación) 51, 271, 310
 - cut, evento 145
- ## D
- data, propiedad
 - clase NativeMenuItem 85
 - DataTransfer, objeto
 - propiedad types 139
 - DataTransfer, objeto (arrastrar y colocar en HTML) 136, 137, 138, 139, 208
 - datos binarios
 - Consulte* conjuntos de bytes
 - datos cifrados, almacenar y recuperar 199
 - datos, compresión 157
 - deactivate, evento 75
 - declaración de prácticas de certificación (DPC) 306
 - declaraciones, SQL 173
 - default_badge.html 294
 - deflate, compresión 157
 - DELETE, declaración (SQL) 186
 - deleteDirectory(), método (clase File) 113
 - deleteDirectoryAsync(), método (clase File) 113
 - deleteFile(), método (clase File) 115
 - deleteFileAsync(), método (clase File) 115
 - delimitador de ruta (sistema de archivos) 105
 - desarrolladores de Apple, certificados 303
 - description, elemento (archivo descriptor de la aplicación) 47
 - designMode, propiedad (objeto Document) 139, 209
 - desinstalar
 - aplicaciones de AIR 26
 - motor de ejecución de AIR 2
 - desktopDirectory, propiedad (clase File) 103
 - Dictionary, clase 223
 - dimensiones de las ventanas 50
 - directorio Archivos de programa (Windows) 292
 - directorio de almacenamiento de la aplicación 25, 103, 107, 224
 - directorio de documentos 103
 - directorio de inicio 103
 - directorio de la aplicación 103
 - directorio del escritorio 103
 - directorios 103, 111
 - copiar 112
 - crear 111
 - eliminar 113, 115
 - enumerar 112
 - invocar aplicaciones 272
 - mover 112
 - referencias 103

- directorios temporales 111
 - dispatchEvent(), método (clase NativeWindow) 59
 - display(), método (clase NativeMenu) 91
 - displaying, evento 83, 93
 - displayState, propiedad (clase Stage) 76
 - displayStateChange, evento 59, 75
 - displayStateChanging, evento 59, 75
 - distribuir aplicaciones de AIR 292
 - dock, iconos 97
 - compatibilidad 96
 - efecto de rebote 97
 - minimizar ventanas 70
 - Document, objeto
 - método createElement() 222
 - método wirtelin() 209
 - método write() 33, 209, 222
 - método writeln() 33, 222
 - propiedad designMode 139, 209
 - propiedad stylesheets 230
 - documentación de ActionScript 9
 - documentación de Flash 9
 - documentación, relacionada 9
 - documentRoot, atributo (elementos frame e iframe) 30, 203, 211, 232
 - documentRoot, atributos (elementos frame e iframe) 30
 - documentsDirectory, propiedad (clase File) 103
 - doDrag(), método (clase NativeDragManager) 128, 130, 133
 - DOM HTML y ventanas nativas 58
 - dominitialize, evento 212
 - dos puntos (\
 -), en los nombres de parámetros de declaración SQL 175
 - DPAPI (asociación de datos cifrados con usuarios) 199
 - drag, evento 135, 208
 - dragend, evento 135, 208
 - dragenter, evento 135, 208
 - dragleave, evento 135, 208
 - dragover, evento 135, 208
 - dragstart, evento 135, 208
 - DRM 261
 - credenciales 268
 - DRMAuthenticateEvent, clase 262, 266
 - DRMErrorEvent, clase 262
 - códigos de error 268
 - propiedad subErrorID 268
 - DRMStatusEvent, clase 262
 - drop, evento 135, 208
 - dropAction, propiedad (clase NativeDragEvent) 132, 133
 - dropEffect, propiedad (objeto DataTransfer) 136, 137, 208
- E**
- effectAllowed, propiedad (objeto DataTransfer) 136, 137, 138, 208
 - ejecutar aplicaciones de AIR 292, 300
 - elementos de menú 83
 - activado 85
 - caracteres de letras de selección 85
 - copiar y pegar 148
 - crear 87
 - datos, asignar 85
 - equivalentes de teclas 84
 - estados 85
 - seleccionado 85
 - seleccionar 92
 - teclas de aceleración 84
 - elementos de menú activados 85
 - elementos de menú seleccionados 85
 - eliminar archivos 115
 - eliminar directorios 113, 115
 - emergentes, menús 91
 - empaquetar archivos de AIR
 - AIR Developer Tool (ADT) 326
 - encoding, propiedad (clase File) 110
 - EncryptedLocalStore, clase 199
 - Endian.BIG_ENDIAN 156
 - Endian.LITTLE_ENDIAN 156
 - enterFrame, evento 67
 - entorno limitado de la aplicación 27, 203, 204, 217, 218, 219, 222, 232, 281
 - entorno limitado local con acceso a la red 27
 - entorno limitado local con sistema de archivos 27, 204
 - entorno limitado local de confianza 27, 204
 - entorno limitado remoto 27
 - entornos limitados 27, 204, 232, 281
 - entornos limitados ajenos a la aplicación 29, 141, 203, 204, 218, 219, 232
 - entornos limitados remotos 204
 - enumerar directorios 112
 - enumerar pantallas 78
 - equivalentes de teclas para comandos de menú 84
 - error, evento 174
 - escribir archivos 116
 - escritorio, ventanas
 - Consulte ventanas
 - espacio de nombres XML (archivo descriptor de la aplicación) 45
 - esquemas de URL 107
 - etiquetas de script
 - propiedad src 33
 - eval(), función 28, 32, 205, 218, 219
 - eventos
 - clase NativeWindow 75
 - controladores 239
 - detectores 239
 - HTML 236
 - menú 83, 91
 - ventanas nativas 59
 - examinar
 - para seleccionar un archivo 106
 - para seleccionar un directorio 105
 - execute(), método (clase SQLStatement) 174, 177, 184
 - exists, propiedad (clase File) 113
 - exit(), método
 - clase NativeApplication 276
 - exiting, evento 276
 - extensiones (de archivos), asociar con una aplicación de AIR 51, 273, 280
- F**
- filas (base de datos) 166, 184
 - File, clase 101, 102
 - getRootDirectories() 102
 - hacer referencia a una base de datos local 169
 - método browseForDirectory() 105
 - método browseForOpen() 106
 - método browseForSave() 106
 - método copyTo() 114
 - método copyToAsync() 114
 - método createDirectory() 111
 - método createTempDirectory() 111, 116
 - método createTempFile() 116
 - método deleteDirectory() 113
 - método deleteDirectoryAsync() 113
 - método deleteFile() 115
 - método deleteFileAsync() 115
 - método getDirectoryListingAsync() 112
 - método getRootDirectories() 102
 - método moveTo() 114
 - método moveToAsync() 114
 - método moveToTrash() 115
 - método moveToTrashAsync() 115

- método relativize() 108
- método resolvePath() 102
- propiedad
 - applicationStorageDirectory 102
- propiedad creationDate 113
- propiedad creator 113
- propiedad desktopDirectory 102
- propiedad documentsDirectory 102
- propiedad encoding 110
- propiedad exists 113
- propiedad isDirectory 113
- propiedad lineEnding 110
- propiedad modificationDate 113
- propiedad name 113
- propiedad nativePath 102, 113
- propiedad parent 113
- propiedad separator 110
- propiedad size 113
- propiedad spaceAvailable 110
- propiedad type 113
- propiedad url 102, 113
- propiedad userDirectory 102
- file, esquema de URL 39, 107, 224
- FileMode, clase 101
- filename, elemento (archivo descriptor de la aplicación) 46
- FileStream, clase 101
- fileTypes, elemento (archivo descriptor de la aplicación) 51, 280
- firmar código 43, 301
- firmas
 - migrar 305, 332
- firmas digitales 301, 326, 330
- Flash Media Rights Management Server 261
- Flash Player 53, 201, 204, 223
- FlashVars, opciones (para utilizar badge.swf) 294
- Flex
 - compatibilidad con arrastrar y colocar 129
- FMRMS (Flash Media Rights Management Server) 261
- fondo cromático del sistema 60
 - ventanas HTML 65
- fondo cromático personalizado 60
- fondo de ventanas 61
- formato de mensaje de acción (AMF) 129, 154, 157
- formatos de datos, Clipboard 149
- fotogramas 30
- frame, elementos 203, 206, 211
- función de invocación desde el navegador 51, 274
- funciones (JavaScript)
 - constructor 221
 - definiciones 33
 - literales 33
- G**
 - generación de código dinámico 32
 - generalClipboard, propiedad (clase Clipboard) 143
 - gesto de arrastrar hacia dentro 127, 132
 - gesto de arrastrar hacia fuera 127, 129
 - getApplicationVersion(), método (archivo air.swf) 298
 - getData(), método
 - clase Clipboard 133
 - evento de copiar y pegar en HTML 145
 - objeto ClipboardData 207
 - objeto DataTransfer 139, 208
 - getData(), método (clase Clipboard) 129
 - getData(), método (de una propiedad dataTransfer de un evento drag en HTML) 136
 - getDefaultApplication(), método (clase NativeApplication) 280
 - getDirectoryListing(), método (clase File) 112
 - getDirectoryListingAsync(), método (clase File) 112
 - getResult(), método (clase SQLStatement) 184
 - getScreensForRectangle(), método (clase Screen) 78
 - getStatus(), método (air.swf file) 297
 - gráficos vectoriales escalables (SVG) 204
 - GZIP, formato 157
- H**
 - hasEventListener(), método 241
 - height, elemento (archivo descriptor de la aplicación) 50
 - height, propiedad (clase HTMLLoader) 217
 - hojas de estilo, HTML
 - manipular en ActionScript 230
 - hostContainer, propiedad (PDF) 258
 - HTML
 - cargar contenido 217
 - compatibilidad con arrastrar y colocar 127, 136
 - copiar y pegar 145
 - desplazamiento 236
 - DOM, acceder desde ActionScript 226
 - entornos limitados 204
 - eventos 236
 - extensiones en AIR 211
 - imprimir 204
 - objetos incorporados 203
 - plug-ins 203
 - seguridad 30, 203, 232
 - superponer contenido SWF 66
 - ventanas 64
 - htmlBoundsChanged, evento 236
 - htmlDOMInitialize, evento 236
 - HTMLLoader, clase 217
 - acceso de JavaScript 202
 - copiar y pegar 144
 - eventos 236
 - método createRootWindow() 64, 66, 217
 - método loadString() 217
 - propiedad height 217
 - propiedad paintsDefaultBackground 61, 67
 - propiedad pdfCapability 256
 - propiedad
 - runtimeApplicationDomain 226
 - propiedad width 217
 - htmlLoader, propiedad (objeto Window) 202, 210, 217
 - htmlLoader, propiedad (objeto window) 65
 - HTMLPDFCapability, clase 256
 - HTMLUncaughtScriptException, clase 237
- I**
 - Icon, clase
 - método bounce() 97
 - propiedad bitmaps 96
 - icon, elemento (archivo descriptor de la aplicación) 50
 - icon, propiedad (clase NativeApplication) 96
 - iconos
 - animar 96
 - aplicaciones 50
 - bandeja del sistema 96
 - barra de tareas 70, 96
 - dock 96, 97
 - eliminar 96
 - imágenes 96
 - iconos del dock
 - menús 85
 - ID de aplicación 46
 - id, elemento (archivo descriptor de la aplicación) 46

- id, elemento (clase `NativeApplication`) 278
- identificadores de editor 278, 302
- `idleThreshold`, propiedad (clase `NativeApplication`) 281
- `iframe`, elementos 30, 203, 206, 211
- imágenes de mapa de bits, establecer para iconos 96
- `img`, etiquetas (en el contenido del objeto `TextField`) 28
- imprimir 204
- `Info.plist`, archivos (Mac OS) 48
- información de copyright para aplicaciones de AIR 48
- iniciar aplicaciones de AIR 271
- iniciar sesión en el sistema, iniciar una aplicación de AIR 274
- iniciar sesión, iniciar una aplicación de AIR 274
- inicio (sistema), iniciar una aplicación de AIR 274
- inicio automático (iniciar una aplicación de AIR al iniciar sesión) 274
- Inicio, menú (Windows) 49
- `initialWindow`, elemento (archivo descriptor de la aplicación) 49, 58
- `innerHTML`, propiedad 33, 209, 222
- `INSERT`, declaración (SQL) 189
- instalar
 - motor de ejecución de AIR 1
- instalar aplicaciones de AIR 292
- `installApplication()`, método (archivo `air.swf`) 299
- `installFolder`, elemento (archivo descriptor de la aplicación) 48
- `INTEGER PRIMARY KEY`, columnas (SQL) 185
- invocar aplicaciones de AIR 271
- `invoke`, evento 271
- `InvokeEvent`, clase 52, 272
 - propiedad `arguments` 272
 - propiedad `currentDirectory` 272
- `isDirectory`, propiedad (clase `File`) 113
- `isHTTPS`, propiedad (clase `BrowserInvokeEvent`) 275
- `isSetAsDefaultApplication()`, método (clase `NativeApplication`) 280
- J**
- Java Cryptography Architecture (JCA) 330
- JavaScript
 - acceder a las API de AIR 222
 - archivo `AIRAliases.js` 202, 223
 - compatibilidad con AIR 204
 - errores 218, 226, 237, 240
 - eventos de error 236
 - eventos, gestión 239
 - evitar errores de seguridad 219
 - motor de ejecución de AIR 201
 - PDF 257
 - programación 217
 - reutilizar scripts `ActionScript` 225
 - seguridad 232
- JavaScript, esquema URL 33, 211, 221
- JavaScript, seguridad 32
- JavaSoft Developer, certificados 303
- JSON 205
- K**
- `Keyboard`, clase 84
- `KeyChain` (asociación de datos cifrados con usuarios) 199
- `keyEquivalent`, propiedad (clase `NativeMenuItem`) 84
- `keyEquivalentModifiers`, propiedad (clase `NativeMenuItem`) 84
- L**
- `label`, propiedad (clase `NativeMenuItem`) 148
- `lastInsertRowID`, propiedad (clase `SQLResult`) 184
- `lastUserInput`, propiedad (clase `NativeApplication`) 281
- leer archivos 116
- `length`, propiedad (clase `ByteArray`) 155
- libros de Adobe Press 9
- licencias, utilizar para contenido con cifrado DRM 261
- líneas separadoras, menú 87
- `lineEnding`, propiedad (clase `File`) 110
- lista de revocación de certificados (LRC) 306
- listas de archivos
 - compatibilidad con arrastrar y colocar 136
- `listRootDirectories()`, método (clase `File`) 103
- literales de objetos (en JavaScript) 32
- `little-endian`, orden de bytes 156
- `load`, evento 203, 205, 219, 226
- `load`, eventos 222
- `loadBytes()`, método (clase `Loader`) 41
- `Loader`, clase 65
- `Loader.loadBytes()`, método 41
- `LoaderContext`, clase
 - propiedad
 - `allowLoadBytesCodeExecution` 41
 - propiedad `applicationDomain` 35
 - propiedad `securityDomain` 35
- `LoaderInfo`, clase
 - propiedad `childSandboxBridge` 36
 - propiedad `parentSandboxBridge` 36
- `loadString()`, método (clase `HTMLLoader`) 217
- `LocalConnection`, clase 293, 300
- localización 312
- `locationChange`, evento 236
- M**
- Mac OS
 - barra de herramientas 62
 - iconos proxy 62
- `mainScreen`, propiedad (clase `Screen`) 78
- mapas de bits
 - compatibilidad con arrastrar y colocar 127, 136
 - compatibilidad con copiar y pegar 143
- marcas de hora 303
- `maximizable`, elemento (archivo descriptor de la aplicación) 50
- maximizar ventanas 50, 71
- `maximize()`, método (clase `NativeWindow`) 71
- `maxSize`, elemento (archivo descriptor de la aplicación) 50
- Mayús, tecla 84
- menú
 - aplicación 93
 - estructura 83
 - eventos 93
- `menuItemSelect`, eventos 84
- menús 81
 - aplicación 85
 - clases para utilizar 82
 - comandos de copiar y pegar 147
 - crear 85
 - dock 82
 - elemento del dock 85
 - elementos 83
 - emergentes 85, 91
 - equivalentes de teclas 84
 - estructura 82
 - flujo de eventos 83, 91
 - icono de la bandeja del sistema 85
 - iconos de la bandeja del sistema 82

- líneas separadoras 87
 - menús contextuales 87
 - personalizados 82
 - predeterminados por el sistema 82
 - submenús 83, 86
 - tipos 82
 - ventana 85, 93
 - XML, definir con 89
 - menús contextuales 81, 87
 - HTML 88
 - menús de aplicación 81, 93
 - crear 85
 - menús de ventana 81, 93
 - crear 85
 - menús del dock 82
 - menús emergentes 81
 - crear 85
 - menús nativos
 - consulte* menús
 - menuSelect, eventos 84
 - messageHandler, propiedad (PDF) 258
 - métodos abreviados de teclado
 - copiar y pegar 147
 - Microsoft Authenticode, certificados 303
 - Microsoft Authenticode, ID digitales 303
 - Microsoft Windows
 - iconos de barra de título 62
 - migrar una firma 305, 332
 - minimizable, elemento (archivo descriptor de la aplicación) 50
 - minimizar ventanas 50, 59, 71
 - minimize(), método (clase NativeWindow) 71
 - minimumPatchLevel, atributo (archivo descriptor de la aplicación) 45
 - minSize, elemento (archivo descriptor de la aplicación) 50
 - Mis documentos, directorio (Windows) 103
 - mnemonicIndex, propiedad
 - clase NativeMenuItem 85
 - modificationDate, propiedad (clase File) 113
 - monitores
 - Consulte* pantallas
 - motor de ejecución de AIR
 - actualizar 23
 - desinstalar 2
 - detectar 281, 297
 - instalar 1
 - niveles de revisión 45, 281
 - nuevas funciones 53
 - mouseDown, evento 73, 128
 - mouseMove, evento 128
 - mouseTarget, propiedad (clase ContextMenuEvent) 87
 - move, evento 59, 75
 - mover archivos 114
 - mover directorios 112
 - mover ventanas 59, 73
 - moveTo(), método
 - clase File 114
 - objeto Window 58
 - moveToAsync(), método (clase File) 114
 - moveToTrash(), método (clase File) 115
 - moveToTrashAsync(), método (clase File) 115
 - moving, evento 75
- N**
- name, elemento (archivo descriptor de la aplicación) 47
 - name, propiedad (clase File) 113
 - nativas, ventanas
 - Consulte* ventanas
 - NativeApplication, clase 211
 - método addEventListener() 271
 - método copy() 147
 - método exit() 276
 - método getDefaultApplication() 280
 - método isSetAsDefaultApplication() 280
 - método
 - removeAsDefaultApplication() 280
 - método setAsDefaultApplication() 51
 - propiedad applicationDescriptor 278
 - propiedad autoExit 276
 - propiedad icon 96
 - propiedad id 278
 - propiedad idleThreshold 281
 - propiedad lastUserInput 281
 - propiedad publisherID 278, 302
 - propiedad runtimePatchLevel 281
 - propiedad runtimeVersion 281
 - propiedad startAtLogin 274
 - propiedad supportsDockIcon 96
 - propiedad supportsMenu 93
 - propiedad supportsSystemTrayIcon 96
 - NativeApplication, clase
 - propiedad activeWindow 69
 - NativeApplication.setAsDefaultApplication(), método 280
 - NativeBoundsEvent, clase 75
 - nativeDragComplete, evento 128, 132, 134
 - nativeDragDrop, evento 128
 - nativeDragEnter, evento 128, 132, 133, 134
 - NativeDragEvent, clase
 - propiedad clipboard 133
 - propiedad dropAction 132, 133
 - nativeDragExit, evento 128, 134
 - NativeDragManager, clase
 - método acceptDragDrop() 128, 133
 - método doDrag() 128, 130, 133
 - nativeDragOver, evento 128, 132, 133, 134
 - nativeDragStart, evento 128, 134
 - nativeDragUpdate, evento 128, 134
 - NativeMenu, clase 83, 91
 - NativeMenuItem, clase 83
 - propiedad data 85
 - propiedad keyEquivalent 84
 - propiedad keyEquivalentModifiers 84
 - propiedad mnemonicIndex 85
 - propiedad submenu 83
 - nativePath, propiedad (clase File) 103, 113
 - NativeWindow, clase 57
 - acceso de JavaScript 202
 - constructor 64
 - crear instancias 68
 - dispatchEvent(), método 59
 - eventos 75
 - método activate 69
 - método activate() 64, 70
 - método addEventListener() 75
 - método close() 71
 - método maximize() 71
 - método minimize() 71
 - método orderBehind() 70
 - método orderInBackOf() 70
 - método orderInFrontOf() 70
 - método orderToBack() 70
 - método orderToFront() 70
 - método restore() 71
 - método startMove() 73
 - método startResize() 73
 - propiedad alwaysInFront 70
 - propiedad stage 67
 - propiedad systemChrome 60
 - propiedad systemMaxSize 64
 - propiedad systemMinSize 64
 - propiedad transparent 60, 61
 - propiedad type 60
 - propiedad visible 64, 69

- nativeWindow, propiedad
 - clase Stage 64, 69
 - objeto Window 202, 210
 - nativeWindow, propiedad (objeto window) 58, 65
 - NativeWindowDisplayStateEvent, clase 75
 - NativeWindowInitOptions, clase 63, 64, 65
 - navegadores Web
 - detectar el motor de ejecución de AIR 297
 - detectar instalación de una aplicación de AIR 298
 - iniciar aplicaciones de AIR 300
 - instalar aplicaciones de AIR 299
 - navegadores Web
 - iniciar aplicaciones de AIR 274
 - NetStream, clase
 - contenido cifrado, reproducir 262
 - método resetDRMVouchers() 264
 - método
 - setDRMAuthenticationCredentials() 262, 264
 - niveles de revisión
 - motor de ejecución de AIR 281
 - niveles de revisión, motor de ejecución de AIR 45
 - _ROWID_ nombre de columna (SQL) 185
 - SQL
 - _ROWID_ nombre de columna 185
 - nombre de editor desconocido (en instalador de aplicaciones de AIR) 301
 - nombre del editor 301
 - nombres de usuario
 - configurar para contenido de medios cifrados 261
 - NSHumanReadableCopyright, campo (Mac OS) 48
- O**
- objetos Date, convertir entre ActionScript y JavaScript 230
 - objetos incorporados (en HTML) 203
 - objetos RegExp, convertir entre ActionScript y JavaScript 230
 - objetos serializados
 - compatibilidad con arrastrar y colocar 127
 - compatibilidad con copiar y pegar 143
 - OID, nombre de columna (SQL) 185
 - onclick, controlador 221
 - ondominitialize, atributo 212
 - onload, controlador 32
 - onmouseover, controlador 221
 - open(), clase (clase SQLConnection) 169
 - open(), método
 - clase SQLConnection 169
 - objeto Window 35, 64, 211
 - openAsync(), método (clase SQLConnection) 169, 172
 - opener, propiedad (objeto window) 65
 - orden de bytes 156
 - orden de visualización, ventanas 70
 - orderBehind(), método (clase NativeWindow) 70
 - orderInBackOf(), método (clase NativeWindow) 70
 - orderInFrontOf(), método (clase NativeWindow) 70
 - orderToBack(), método (clase NativeWindow) 70
 - orderToFront(), método (clase NativeWindow) 70
 - outerHTML, propiedades 209
- P**
- P12, archivos 302
 - paintsDefaultBackground, propiedad (clase HTMLLoader) 61, 67
 - pantalla completa, ventanas 76
 - pantallas 77
 - enumerar 78
 - principal 78
 - ventanas, desplazamiento 78
 - papelera (eliminar archivos) 115
 - parameters, propiedad (clase SQLStatement) 174, 175
 - parámetros con nombre, (en declaraciones SQL) 175
 - parámetros sin nombre, (en declaraciones SQL) 176
 - parámetros, en declaraciones SQL 175
 - parent, propiedad (clase File) 113
 - parent, propiedad (objeto window) 65
 - parentSandboxBridge, propiedad
 - clase LoaderInfo 36
 - objeto Window 31
 - parentSandboxBridge, propiedad (objeto Window) 210
 - parentSandboxBridge, propiedad (objeto Window) 233
 - PDF
 - compatibilidad 203, 256
 - pdfCapability, propiedad (clase HTMLLoader) 256
 - PFX, archivos 302
 - playerType, propiedad
 - clase Capabilities 281
 - plug-ins (en HTML) 203
 - posición de las ventanas 50
 - posición del ratón al arrastrar 134
 - position, propiedad (clase ByteArray) 155
 - postMessage(), método (objeto PDF) 258
 - principal, pantalla 78
 - print(), método (objeto Window) 204
 - privilegios necesarios para actualizar el motor de ejecución de AIR o una aplicación de AIR 293, 299
 - privilegios requeridos para actualizar el motor de ejecución de AIR o una aplicación de AIR 24
 - programación asíncrona
 - bases de datos 168, 172, 190
 - sistema de archivos 101
 - XMLHttpRequests 222
 - programación sincrónica
 - bases de datos 168, 172, 190
 - sistema de archivos 101
 - XMLHttpRequests 222
 - programMenuFolder, elemento (archivo descriptor de la aplicación) 49
 - webkit-user-drag, propiedad de CSS 138
 - proxy, iconos
 - Mac OS 62
 - publisherid, archivo 279
 - publisherID, propiedad (clase NativeApplication) 278, 302
 - puentes de entorno limitado 31, 35, 203, 204, 219, 232
- R**
- readBytes(), método (clase ByteArray) 154
 - readFloat(), método (clase ByteArray) 154
 - readInt(), método (clase ByteArray) 154
 - readObject(), método (clase ByteArray) 154
 - readUTFBytes(), método (clase ByteArray) 154
 - referencias de objetos
 - compatibilidad con arrastrar y colocar 127
 - compatibilidad con copiar y pegar 143
 - registrar tipos de archivos 280
 - registro de Windows, parámetros 26
 - relativas, rutas 108
 - relativize(), método (clase File) 108
 - removeAsDefaultApplication(), método (clase NativeApplication) 280
 - removeEventListener(), método 240

- representación aplazada (copiar y pegar) 151
 - requisitos
 - procesar PDF 256
 - resetDRMVouchers(), método (clase NetStream) 264
 - resizable, elemento (archivo descriptor de la aplicación) 50
 - resize, evento 59, 75
 - resizing, evento 75
 - resolvePath(), método (clase File) 103
 - Responder, clase 174, 184
 - restaurar ventanas 59, 71
 - restore(), método (clase NativeWindow) 71
 - result, evento 174
 - ROWID, nombre de columna (SQL) 185
 - runtime, propiedad (objeto Window) 65, 202, 210, 223
 - runtimeApplicationDomain, propiedad (clase HTMLLoader) 226
 - runtimePatchLevel, propiedad (clase NativeApplication) 281
 - runtimeVersion, propiedad (clase NativeApplication) 281
 - rutas (archivos y directorios) 107
 - rutas relativas (entre archivos) 108
- S**
- salir de aplicaciones de AIR 271
 - sandboxRoot, atributo (elementos frame e iframe) 203, 206, 211, 232
 - sandboxRoot, propiedad
 - fotogramas 30
 - iframe 30
 - sandboxType, propiedad
 - clase BrowserInvokeEvent 275
 - clase Security 281
 - scaleMode, propiedad
 - clase Stage 73
 - Screen, clase 77
 - método getScreenForRectangle() 78
 - propiedad mainScreen 78
 - propiedad screens 78
 - screens, propiedad (clase Screen) 78
 - screenX, propiedad (eventos drag en HTML) 136
 - screenY, propiedad (eventos drag en HTML) 136
 - script, etiquetas 205, 209, 222, 224, 228
 - scripts, reutilizar 225, 232
 - scroll, evento 236
 - Security, clase
 - método allowDomain() 35, 40
 - propiedad sandboxType 281
 - securityDomain, propiedad (clase BrowserInvokeEvent) 275
 - seguridad
 - ataques de desactualización 43
 - base de datos 176
 - caché entre dominios 29
 - campos de texto 28
 - cargar contenido 65
 - cifrado de datos 199
 - Clipboard 144
 - credenciales de usuario 42
 - CSS 29
 - directorio de almacenamiento de la aplicación 25
 - entorno limitado de la aplicación 27
 - entornos limitados 27, 203, 204, 232, 281
 - entornos limitados ajenos a la aplicación 29
 - errores de JavaScript 218
 - etiquetas img 28
 - fotogramas 30
 - frames 29
 - función de invocación desde el navegador 275
 - función eval() 32
 - generación de código dinámico 32
 - HTML 30, 32, 201, 203, 218
 - iframes 29, 30
 - instalar (aplicación y motor de ejecución) 23
 - JavaScript 232
 - marcos de Ajax 33
 - método Loader.loadBytes() 41
 - objetos XMLHttpRequest 34
 - prácticas recomendadas 41
 - privilegios de usuario para la instalación 24
 - protocolo asfunction 28
 - puentes de entorno limitado 31, 35, 232
 - sistema de archivos 39
 - usar scripts 35
 - window.open() 35
 - XMLHttpRequest 212
 - seguridad y ataque de desactualización 43
 - SELECT, declaración (SQL) 177, 189
 - select, evento 83, 92, 93
 - selector de archivos, cuadros de diálogo 106
 - selector de directorios, cuadros de diálogo 105
 - separator, propiedad (clase File) 110
 - serializar objetos 129
 - setAsDefaultApplication(), método (clase NativeApplication) 51, 280
 - setData(), clase
 - método Clipboard 151
 - setData(), método
 - objeto ClipboardData 207
 - objeto DataTransfer 136, 138, 208
 - setDataHandler(), método (clase Clipboard) 151
 - setDragImage(), método (de una propiedad dataTransfer de un evento drag de HTML) 136
 - setDRMAuthenticationCredentials(), método (clase NetStream) 262, 264
 - setInterval(), función 33, 210, 221
 - setTimeout(), función 33, 210, 221
 - sistema de archivos
 - seguridad 39
 - sitio Web de asistencia técnica de Adobe 9
 - size, propiedad (clase File) 113
 - spaceAvailable, propiedad (clase File) 110
 - SQL
 - clases usadas 167
 - columnas AUTOINCREMENT 185
 - columnas INTEGER PRIMARY KEY 185
 - conceptos 166
 - declaración CREATE TABLE 170
 - declaración DELETE 186
 - declaración INSERT 189
 - declaración SELECT 177, 189
 - declaración UPDATE 186
 - declaraciones 173
 - introducir datos 176, 189
 - nombre de columna OID 185
 - nombre de columna ROWID 185
 - parámetros con nombre (en declaraciones) 175
 - parámetros en declaraciones 175
 - parámetros sin nombre (en declaraciones) 176
 - SQLCollationType, clase 167
 - SQLColumnNameStyle, clase 167
 - SQLConnection, clase 167
 - método attach() 186
 - método open 169
 - método open() 169
 - método openAsync() 169, 172

- sqlConnection, propiedad (clase SQLStatement) 174
 - SQLException, clase 167, 174
 - SQLExceptionEvent, clase 167, 174
 - SQLEvent, clase 167
 - SQLIndexSchema, clase 167
 - SQLMode, clase 167, 173
 - SQLResult, clase 167
 - SQLSchemaResult, clase 167
 - SQLStatement, clase 167, 173
 - método execute 174
 - método execute() 177, 184
 - método getResult() 184
 - objeto parameters 174
 - propiedad parameters 175
 - propiedad sqlConnection 174
 - propiedad text 174, 175, 177, 186
 - SQLTableSchema, clase 167
 - SQLTransactionLockType, clase 167
 - SQLTriggerSchema, clase 167
 - SQLUpdateEvent, clase 167
 - SQLViewSchema, clase 167
 - Stage, clase
 - método addChild() 67
 - método addChildAt() 67
 - propiedad displayState 76
 - propiedad nativeWindow 64, 69
 - propiedad scaleMode 64, 73
 - stage, propiedad
 - clase NativeWindow 67
 - StageDisplayState, clase 76
 - StageScaleMode, clase 64, 73
 - startAtLogin, propiedad (clase NativeApplication) 274
 - startMove(), método (clase NativeWindow) 73
 - startResize(), método (clase NativeWindow) 73
 - StatusEvent, clase 262
 - styleSheets, propiedad (objeto Document) 230
 - subErrorID, propiedad (clase DRMErrorEvent) 268
 - submenu, propiedad
 - clase NativeMenuItem 83
 - submenús 83, 86
 - Sun Java, ID de firma digital 303
 - superpuestas, ventanas 70
 - supportsDockIcon, propiedad (clase NativeApplication) 96
 - supportsMenu, propiedad (clase NativeApplication) 93
 - supportsSystemTrayIcon, propiedad (clase NativeApplication) 96
 - SWF, contenido
 - superponer sobre HTML 66
 - systemChrome, propiedad (clase NativeWindow) 60
 - systemMaxSize, propiedad (clase NativeWindow) 64
 - systemMinSize, propiedad (clase NativeWindow) 64
- T**
- tablas (base de datos) 166
 - crear 170
 - tamaño de las ventanas 50
 - tamaño, ventanas 64
 - teclas de aceleración para comandos de menú 84
 - teclas modificadoras
 - elementos de menú 84
 - teclas principales
 - elementos de menú 84
 - text, propiedad (clase SQLStatement) 174, 175, 177, 186
 - TextField, clase
 - copiar y pegar 144
 - etiquetas img 28
 - HTML cargado en 217
 - texto
 - compatibilidad con arrastrar y colocar 127, 136
 - compatibilidad con copiar y pegar 143
 - Thawte, certificados 301, 302
 - tiempo de inactividad (usuario) 281
 - tipos de datos, base de datos 189
 - tipos MIME
 - arrastrar y colocar en HTML 136
 - copiar y pegar en HTML 150, 207
 - title, elemento (archivo descriptor de la aplicación) 49
 - traducción de aplicaciones 312
 - transparent, elemento (archivo descriptor de la aplicación) 50
 - transparent, propiedad (clase NativeWindow) 60, 61
 - type, propiedad (clase File) 113
 - type, propiedad (clase NativeWindow) 60
 - types, propiedad
 - evento de copiar y pegar en HTML 145
 - evento drag en HTML 136
 - objeto DataTransfer 208
 - types, propiedad (objeto DataTransfer) 139
- U**
- uncaughtScriptException, evento 236
 - uncompress(), método (clase ByteArray) 157
 - unload, eventos 209
 - UntrustedAppInstallDisabled (parámetros del registro de Windows) 26
 - UPDATE, declaración (SQL) 186
 - update(), método (clase Updater) 308
 - UpdateDisabled (parámetros del registro de Windows) 26
 - Updater, clase 308
 - URL 224
 - cargar contenido HTML 217
 - compatibilidad con arrastrar y colocar 127, 136
 - compatibilidad con copiar y pegar 143
 - url, propiedad
 - clase File 103, 113
 - url, propiedad (clase File) 103
 - URLStream, clase 206
 - usar scripts entre contenidos 35
 - userDirectory, propiedad (clase File) 103
 - userIdle, evento 281
 - userPresent, evento 281
- V**
- vaciar directorios 113
 - validación de datos, invocar aplicaciones 275
 - ventana
 - fondo cromático 60
 - ventanas 57
 - activar 64
 - activas 69, 70
 - aparición 60
 - cambiar de tamaño 50, 59, 73
 - cerrar 59, 71, 276
 - clases para trabajar 58
 - comportamiento 60
 - crear 62, 68, 217
 - desplazamiento 78
 - estilo 60
 - eventos 75
 - flujo de eventos 59
 - fondo 61
 - fondo cromático del sistema 60
 - fondo cromático personalizado 60
 - gestionar 69
 - inicial 58
 - inicializar 62

- ligeras 60
 - maximizar 50, 59, 71
 - minimizar 50, 59, 70, 71
 - mostrar 69
 - mover 59, 73
 - no rectangulares 61
 - ocultar 69
 - orden de visualización 70
 - ordenar 70
 - posición 50
 - propiedades 49
 - restaurar 59, 71
 - tamaño 50, 64
 - tamaño máximo 64
 - tamaño mínimo 64
 - tipos 60
 - transparencia 50, 61
 - ventanas de utilidades 60
 - ventanas normales 60
 - visibilidad 50
 - ventanas de utilidades 60
 - ventanas ligeras 60
 - ventanas normales 60
 - ventanas transparentes 50, 61
 - ventanas, activar 64, 70
 - ventanas, apariencia 60
 - ventanas, cerrar 71
 - ventanas, maximizar 59
 - ventanas, minimizar 70
 - ventanas, mostrar 69
 - ventanas, ocultar 69
 - ventanas, orden 70
 - ventanas, ordenar 70
 - Verisign, certificados 301, 302
 - version, elemento (archivo descriptor de la aplicación) 46
 - versiones, aplicación de AIR 281
 - vídeos FLV, cifrado 261
 - vinculación fuerte de datos cifrados 199
 - visibilidad de las ventanas 50
 - visible, elemento (archivo descriptor de la aplicación) 50
 - visible, propiedad
 - clase NativeWindow 64, 69
 - visualización, objetos (ActionScript) 68
 - visualizaciones
 - Consulte pantallas
 - volúmenes raíz 103
- W**
- WebKit 201, 204, 215
 - webkit-border-horizontal-spacing, propiedad de CSS 215
 - webkit-border-vertical-spacing, propiedad de CSS 215
 - webkit-line-break, propiedad de CSS 215
 - webkit-margin-bottom-collapse, propiedad de CSS 215
 - webkit-margin-collapse, propiedad de CSS 215
 - webkit-margin-start, propiedad de CSS 215
 - webkit-margin-top-collapse, propiedad de CSS 215
 - webkit-nowrap-mode, propiedad de CSS 215
 - webkit-padding-start, propiedad de CSS 215
 - webkit-rtl-ordering, propiedad de CSS 215
 - webkit-text-fill-color, propiedad de CSS 215
 - webkit-text-security, propiedad de CSS 215
 - webkit-user-drag, propiedad de CSS 135, 216
 - webkit-user-modify, propiedad de CSS 216
 - webkit-user-select, propiedad de CSS 135, 138, 216
 - width, elemento (archivo descriptor de la aplicación) 50
 - width, propiedad (clase HTMLLoader) 217
 - Window objeto
 - método moveTo() 58
 - Window, clase 57
 - Window, objeto
 - método close() 58
 - método open 211
 - método open() 35, 64
 - método print() 204
 - objeto htmlLoader 202
 - objeto nativeWindow 202
 - propiedad childSandboxBridge 31
 - propiedad htmlLoader 65, 210, 217
 - propiedad nativeWindow 58, 65, 210
 - propiedad opener 65
 - propiedad parent 65
 - propiedad parentSandboxBridge 31, 210, 233
 - propiedad runtime 28, 34, 65, 202, 210, 223
- WindowedApplication, clase 57
- windows
 - modos de escala de escenario 64
- write(), método (objeto Document) 209, 222
 - writeBytes(), método (clase ByteArray) 154
 - writeFloat(), método (clase ByteArray) 154
 - writeInt(), método (clase ByteArray) 154
 - writeln(), método (objeto Document) 209, 222
 - writeObject(), método (clase ByteArray) 154
 - writeUTFBytes(), método (clase ByteArray) 154
- X**
- x, elemento (archivo descriptor de la aplicación) 50
 - XML
 - clase 223
 - definir menús con 89
 - XMLHttpRequest, objeto 34, 205, 211, 222
 - XMLList, clase 223
 - xmlns (archivo descriptor de la aplicación) 45
- Y**
- y, elemento (archivo descriptor de la aplicación) 50
- Z**
- ZIP, formato de archivo 159
 - ZLIB, compresión 157