

Creazione di applicazioni ADOBE AIR®



Note legali

Per le note legali, vedete http://help.adobe.com/it_IT/legalnotices/index.html.

Sommario

Capitolo 1: Informazioni su Adobe AIR

Capitolo 2: Installazione di Adobe AIR

Installazione di Adobe AIR	3
Rimozione di Adobe AIR	5
Installazione ed esecuzione delle applicazioni AIR di esempio	6
Aggiornamenti di Adobe AIR	6

Capitolo 3: Operazioni con le API AIR

Classi ActionScript 3.0 specifiche per AIR	8
Classi di Flash Player con funzionalità specifiche per AIR	13
Componenti Flex specifici per AIR	16

Capitolo 4: Strumenti della piattaforma Adobe Flash per lo sviluppo AIR

Installazione di AIR SDK	18
Configurazione di Flex SDK	20
Configurazione di kit SDK esterni	21

Capitolo 5: Creazione della prima applicazione AIR

Creazione della prima applicazione AIR Flex desktop in Flash Builder	22
Creazione della prima applicazione AIR desktop con Flash Professional	25
Creazione della prima applicazione AIR per Android con Flash Professional	27
Creazione della prima applicazione AIR per iOS	28
Creazione della prima applicazione AIR basata su HTML con Dreamweaver	32
Creazione della prima applicazione AIR basata su HTML con AIR SDK	35
Creazione della prima applicazione AIR desktop con Flex SDK	39
Creazione della prima applicazione AIR for Android con Flex SDK	43

Capitolo 6: Sviluppo di applicazioni AIR per il desktop

Flusso di lavoro per lo sviluppo di un'applicazione AIR desktop	48
Impostazione delle proprietà delle applicazioni desktop	49
Debug di un'applicazione AIR desktop	54
Creazione del pacchetto di un file di installazione AIR desktop	56
Creazione del pacchetto di un file di installazione nativo desktop	59
Creazione di un pacchetto runtime autonomo per computer desktop	63
Distribuzione di pacchetti AIR per computer desktop	65

Capitolo 7: Sviluppo di applicazioni AIR per dispositivi mobili

Impostazione dell'ambiente di sviluppo	68
Considerazioni sulla progettazione di applicazioni per dispositivi mobili	69
Flusso di lavoro per la creazione di applicazioni AIR per dispositivi mobili	73
Impostazione delle proprietà delle applicazioni per dispositivi mobili	74
Creazione del pacchetto di un'applicazione AIR per dispositivi mobili	97
Debug di un'applicazione AIR per dispositivi mobili	104

Sommario

Installazione di AIR e di applicazioni AIR sui dispositivi mobili	112
Aggiornamento di applicazioni AIR per dispositivi mobili	115
Uso delle notifiche push	116
 Capitolo 8: Sviluppo di applicazioni AIR per dispositivi TV	
Funzionalità AIR per dispositivi TV	125
Considerazioni sulla progettazione di applicazioni AIR per TV	127
Flusso di lavoro per lo sviluppo di un'applicazione AIR per TV	142
Proprietà del descrittore applicazione AIR per TV	144
Creazione del pacchetto di un'applicazione AIR per TV	148
Debug di applicazioni AIR per TV	149
 Capitolo 9: Uso di estensioni native per Adobe AIR	
File ANE (AIR Native Extension)	154
Estensioni native e la classe ActionScript NativeProcess	155
Estensioni native e librerie delle classi ActionScript (file SWC)	155
Dispositivi supportati	155
Profili dispositivo supportati	156
Elenco delle operazioni da eseguire per utilizzare un'estensione nativa	156
Dichiarazione dell'estensione nel file descrittore dell'applicazione	156
Inserimento del file ANE nel percorso di libreria dell'applicazione	157
Creazione del pacchetto di un'applicazione che usa estensioni native	158
 Capitolo 10: Compilatori ActionScript	
Informazioni sugli strumenti della riga di comando AIR in Flex SDK	160
Impostazione del compilatore	160
Compilazione di file di origine MXML e ActionScript per AIR	161
Compilazione di un componente o libreria di codice AIR (Flex)	163
 Capitolo 11: ADL (AIR Debug Launcher)	
Utilizzo di ADL	165
Esempi ADL	168
Codici di errore e di uscita di ADL	169
 Capitolo 12: ADT (AIR Developer Tool)	
Comandi ADT	171
Serie di opzioni ADT	185
Messaggi di errore di ADT	190
Variabili d'ambiente ADT	195
 Capitolo 13: Firma di applicazioni AIR	
Firma digitale di un file AIR	196
Creazione di un file AIR intermedio non firmato mediante ADT	205
Firma di un file AIR intermedio mediante ADT	205
Firma di una versione aggiornata di un'applicazione AIR	206
Creazione di un certificato autofirmato mediante ADT	210

Capitolo 14: File descrittore delle applicazioni AIR

Modifiche del descrittore dell'applicazione	213
Struttura del file descrittore dell'applicazione	215
Elementi del descrittore dell'applicazione AIR	216

Capitolo 15: Profili dispositivo

Restrizione dei profili target nel file descrittore dell'applicazione	255
Funzionalità supportate dai profili	255

Capitolo 16: API interna al browser AIR.SWF

Personalizzazione del file badge.swf per l'installazione invisibile	258
Uso del file badge.swf per l'installazione di un'applicazione AIR	258
Caricamento del file air.swf	262
Verificare se il runtime è installato	262
Verificare da una pagina Web se un'applicazione AIR è installata	263
Installazione di un'applicazione AIR dal browser	264
Avvio dal browser di un'applicazione AIR installata	265

Capitolo 17: Aggiornamento di applicazioni AIR

Informazioni sull'aggiornamento delle applicazioni	268
Presentazione di un'interfaccia di aggiornamento applicazioni personalizzata	270
Scaricamento di un file AIR nel computer dell'utente	270
Verificare se un'applicazione viene eseguita per la prima volta	272
Uso del framework di aggiornamento	274

Capitolo 18: Visualizzazione del codice sorgente

Caricamento, configurazione e apertura del visualizzatore del codice sorgente	288
Interfaccia utente del visualizzatore del codice sorgente	291

Capitolo 19: Debug con AIR HTML Introspector

Informazioni su AIR Introspector	292
Caricamento del codice di AIR Introspector	292
Analisi di un oggetto nella scheda Console	293
Configurazione di AIR Introspector	295
Interfaccia di AIR Introspector	295
Uso di AIR Introspector con contenuto in una sandbox non dell'applicazione	302

Capitolo 20: Localizzazione di applicazioni AIR

Localizzazione del nome e della descrizione dell'applicazione nel programma di installazione dell'applicazione AIR	305
Localizzazione di contenuto HTML con il framework di localizzazione AIR HTML	305

Capitolo 21: Variabili d'ambiente dei percorsi

Impostazione del percorso su Linux e Mac OS con la shell Bash	316
Impostazione del percorso in Windows	317

Capitolo 1: Informazioni su Adobe AIR

Adobe® AIR® è un runtime multischermo compatibile con diversi sistemi operativi che consente di compilare e distribuire applicazioni rich Internet (RIA) per il desktop e i dispositivi mobili sfruttando le capacità di sviluppo per il Web che avete già acquisito. Le applicazioni AIR per dispositivi desktop, TV e mobili possono essere realizzate con ActionScript 3.0 utilizzando Adobe® Flex e Adobe® Flash® (nel caso di applicazioni basate su SWF). Le applicazioni AIR desktop possono anche essere create in HTML, JavaScript® e Ajax (nel caso di applicazioni basate su HTML).

Potete trovare ulteriori informazioni sull'uso di Adobe AIR e sulle prime operazioni con il programma nel sito Adobe AIR Developer Connection (<http://www.adobe.com/devnet/air/>).

AIR vi permette di lavorare negli ambienti a voi familiari, sfruttando gli strumenti di cui già disponete e utilizzando i metodi che preferite. Con il supporto di Flash, Flex, HTML, JavaScript e Ajax, potete realizzare le migliori esperienze possibili in base alle vostre esigenze specifiche.

Ad esempio, potete sviluppare applicazioni utilizzando una o più delle tecnologie seguenti:

- Flash / Flex / ActionScript
- HTML / JavaScript / CSS / Ajax

Gli utenti interagiscono con applicazioni AIR nello stesso modo in cui interagiscono con applicazioni native. Il runtime viene installato una volta nel computer o dispositivo dell'utente, quindi le applicazioni AIR vengono installate ed eseguite come qualunque altra applicazione desktop. (In iOS, il runtime AIR non viene installato separatamente; ogni applicazione AIR per iOS è autonoma.)

La piattaforma e il framework per l'implementazione delle applicazioni forniti dal runtime hanno caratteristiche omogenee nei diversi sistemi operativi e pertanto eliminano la necessità di testare le applicazioni in più browser, garantendo funzionalità e interazioni uniformi in tutti gli ambienti desktop. Invece di sviluppare per un sistema operativo specifico, potete utilizzare il runtime come unico riferimento, ottenendo i seguenti vantaggi:

- Le applicazioni sviluppate per AIR funzionano in più sistemi operativi senza ulteriori interventi da parte vostra. Il runtime garantisce caratteristiche visive e interazioni omogenee e prevedibili nell'intera gamma dei sistemi operativi supportati da AIR.
- Il processo di creazione delle applicazioni risulta più rapido perché avete la possibilità di sfruttare le tecnologie Web e i modelli di design più diffusi. Potete estendere le applicazioni basate su Web portandole sul desktop senza dover apprendere le tradizionali tecnologie di sviluppo per il desktop e senza addentrarvi nella complessità del codice nativo.
- Lo sviluppo delle applicazioni è più semplice rispetto all'uso di linguaggi di livello più basso come C e C++. Non dovete gestire le complesse API di basso livello specifiche di ciascun sistema operativo.

Quando sviluppate applicazioni per AIR, potete avvalervi di un'ampia serie di framework e API:

- API specifiche di AIR fornite dal runtime e dal framework AIR
- API di ActionScript utilizzate nei file SWF e nel framework Flex (nonché altre librerie e framework basati su ActionScript)
- HTML, CSS e JavaScript
- La maggior parte dei framework Ajax
- Le estensioni native per Adobe AIR forniscono API di ActionScript che consentono di accedere a funzionalità specifiche del dispositivo programmate nel codice nativo. Le estensioni native possono inoltre fornire l'accesso a codice nativo precedente e a codice nativo che garantisce prestazioni più elevate.

AIR modifica notevolmente le modalità di creazione, implementazione e interazione delle applicazioni. Avete a disposizione un maggiore controllo, più spazio per la creatività e potete estendere le vostre applicazioni basate su Flash, Flex, HTML e Ajax all'ambiente desktop e ai dispositivi TV e mobili.

Per informazioni sul contenuto di ogni nuovo aggiornamento AIR, vedete le note sulla versione di Adobe AIR (http://www.adobe.com/go/learn_air_relnotes_it).

Capitolo 2: Installazione di Adobe AIR

Il runtime Adobe® AIR® permette di eseguire le applicazioni AIR. È possibile installare il runtime nei modi seguenti:

- Installando il runtime separatamente (senza l'ulteriore installazione di un'applicazione AIR)
- Installando un'applicazione AIR per la prima volta mediante un “badge” di installazione via pagina Web (viene richiesto di installare anche il runtime)
- Mediante la creazione di un programma di installazione personalizzato che installa sia l'applicazione che il runtime. Dovete ottenere l'approvazione di Adobe per distribuire il runtime AIR in questa modalità. Potete richiedere l'approvazione dalla pagina [della concessione in licenza del runtime Adobe](#). Tenete presente che Adobe non fornisce strumenti per la creazione di questo programma di installazione. Sono comunque disponibili numerosi toolkit per la creazione di programmi di installazione di terze parti.
- Installando un'applicazione AIR che contiene AIR come runtime autonomo. Un runtime autonomo viene usato solo dall'applicazione del pacchetto. Non è utilizzato per eseguire altre applicazioni AIR. L'inclusione del runtime è un'opzione in Mac e Windows. In iOS, tutte le applicazioni includono un runtime impacchettato. A partire da AIR 3.7, tutte le applicazioni Android includono un runtime impacchettato per impostazione predefinita, ma è possibile scegliere di usare un runtime separato.
- Impostando un ambiente di sviluppo AIR come AIR SDK, Adobe® Flash® Builder™ oppure Adobe Flex® SDK (che include gli strumenti di sviluppo AIR della riga di comando). Il runtime incluso nell'SDK viene utilizzato solo per il debug delle applicazioni, non per eseguire le applicazioni AIR installate.

I requisiti di sistema per l'installazione di AIR e l'esecuzione di applicazioni AIR sono descritti dettagliatamente sul sito [Adobe AIR: requisiti di sistema \(http://www.adobe.com/it/products/air/systemreqs/\)](http://www.adobe.com/it/products/air/systemreqs/).

Sia il programma di installazione del runtime che il programma di installazione dell'applicazione AIR creano dei file di registro durante l'installazione, l'aggiornamento o la rimozione dell'applicazione o del runtime AIR. Potete consultare questi registri per individuare la causa di eventuali problemi di installazione. Vedete [Installation logs](#).

Installazione di Adobe AIR

Per installare o aggiornare il runtime, l'utente deve disporre dei privilegi amministrativi per il computer in uso.

Installare il runtime su un computer Windows

- 1 Scaricate il file di installazione del runtime da <http://get.adobe.com/it/air>.
- 2 Fate doppio clic sul file di installazione del runtime.
- 3 Nella finestra di installazione, eseguite i comandi per completare l'installazione.

Installare il runtime su un computer Mac

- 1 Scaricate il file di installazione del runtime da <http://get.adobe.com/it/air>.
- 2 Fate doppio clic sul file di installazione del runtime.
- 3 Nella finestra di installazione, eseguite i comandi per completare l'installazione.
- 4 Se il programma di installazione visualizza una finestra di autenticazione, immettete il vostro nome utente e la password di Mac OS.

Installare il runtime su un computer Linux

Nota: in questo momento, AIR 2.7 e successivi non sono supportati su Linux. Le applicazioni AIR distribuite su Linux devono continuare a utilizzare AIR 2.6 SDK.

Utilizzando il file di installazione binario:

- 1 Individuate il file di installazione binario da http://kb2.adobe.com/it/cps/853/cpsid_85304.html e scaricatelo.
- 2 Impostate le autorizzazioni del file per consentire l'esecuzione dell'applicazione di installazione. Da una riga di comando potete impostare le autorizzazioni file con:

```
chmod +x AdobeAIRInstaller.bin
```

Alcune versioni di Linux consentono di impostare le autorizzazioni dei file nella finestra di dialogo delle proprietà aperta tramite un menu di scelta rapida.

- 3 Eseguite il programma di installazione dalla riga di comando o facendo doppio clic sul file di installazione del runtime.
- 4 Nella finestra di installazione, eseguite i comandi per completare l'installazione.

Adobe AIR viene installato come pacchetto nativo, ovvero come rpm in una distribuzione basata su rpm e come deb in una distribuzione Debian. Attualmente AIR non supporta altri formati di pacchetto.

Utilizzando i file di installazione pacchetto:

- 1 Individuate il file del pacchetto AIR da http://kb2.adobe.com/it/cps/853/cpsid_85304.html. Scaricate il pacchetto rpm o Debian, a seconda del formato di pacchetto supportato dal vostro sistema.
- 2 Se necessario, fate doppio clic sul file di pacchetto AIR per installare il pacchetto.

Potete eseguire l'installazione anche dalla riga di comando:

- a Su un sistema Debian:

```
sudo dpkg -i <path to the package>/adobeair-2.0.0.xxxxx.deb
```

- b Su un sistema basato su rpm:

```
sudo rpm -i <path to the package>/adobeair-2.0.0-xxxxx.i386.rpm
```

Oppure, se aggiornate una versione esistente (AIR 1.5.3 o successivo):

```
sudo rpm -U <path to the package>/adobeair-2.0.0-xxxxx.i386.rpm
```

Per installare applicazioni AIR 2 e AIR dovete disporre di privilegi amministrativi per il computer utilizzato.

Adobe AIR viene installato nel percorso seguente: /opt/Adobe AIR/Versions/1.0

AIR registra il tipo mime "application/vnd.adobe.air-application-installer-package+zip", associando in questo modo i file .air a questo tipo mime e registrandoli con il runtime AIR.

Installare il runtime su un dispositivo Android

Potete installare la release più recente del runtime AIR da Android Market.

È possibile installare versioni di sviluppo del runtime AIR da un collegamento a una pagina web oppure utilizzando il comando ADT `-installRuntime`. Potete installare una sola versione per volta del runtime AIR; una versione release e una versione di sviluppo non possono coesistere.

Vedete "Comando ADT `installRuntime`" a pagina 183 per ulteriori informazioni.

Installare il runtime su un dispositivo iOS

Il runtime AIR necessario è impacchettato insieme a ogni applicazione creata per dispositivi iPhone, iPod touch e iPad. Non dovete installare un componente runtime separato.

Altri argomenti presenti nell' Aiuto

“[AIR for iOS](#)” a pagina 73

Rimozione di Adobe AIR

Dopo l'installazione del runtime, potete rimuoverlo mediante le seguenti procedure.

Rimuovere il runtime in un computer Windows

- 1 Nel menu Start di Windows, selezionate Impostazioni > Pannello di controllo.
- 2 Aprite il pannello di controllo Programmi, Programmi e funzioni o Installazione applicazioni (a seconda della versione di Windows utilizzata).
- 3 Selezionate "Adobe AIR" per rimuovere il runtime.
- 4 Fate clic sul pulsante Cambia/Rimuovi.

Rimuovere il runtime in un computer Mac

- Fate doppio clic su “Adobe AIR Uninstaller”, che si trova nella cartella /Applications/Utilities.

Rimuovere il runtime in un computer Linux

Effettuate una delle seguenti operazioni.

- Selezionate il comando relativo al programma di disinstallazione di Adobe AIR dal menu Applicazioni.
- Eseguite il file binario del programma di installazione di AIR con l'opzione `-uninstall`
- Rimuovete i pacchetti AIR (`adobeair` e `adobecerts`) utilizzando lo strumento di gestione pacchetti.

Rimuovere il runtime da un dispositivo Android

- 1 Aprite l'applicazione Impostazioni sul dispositivo.
- 2 Toccate la voce Adobe AIR in Applicazioni > Gestione applicazioni.
- 3 Toccate il pulsante Disinstalla.

Potete anche usare il comando `ADT -uninstallRuntime`. Vedete “[Comando ADT uninstallRuntime](#)” a pagina 184 per ulteriori informazioni.

Eliminazione di un runtime di pacchetto

Per rimuovere un runtime di pacchetto autonomo, dovete rimuovere l'applicazione in cui è installato. Tenete presente che i runtime autonomi vengono usati solo per eseguire l'applicazione di installazione.

Installazione ed esecuzione delle applicazioni AIR di esempio

Per installare o aggiornare un'applicazione AIR, l'utente deve disporre dei privilegi amministrativi per il computer in uso.

Sono disponibili alcune applicazioni di esempio che illustrano le funzionalità di AIR. Potete accedere a tali applicazioni e installarle mediante le seguenti istruzioni:

- 1 Scaricate ed eseguite le [applicazioni di esempio AIR](#). Sono disponibili sia le applicazioni compilate che il codice sorgente.
- 2 Per scaricare ed eseguire un'applicazione di esempio, fate clic sul pulsante Installa adesso dell'applicazione. Viene richiesto di installare ed eseguire l'applicazione.
- 3 Se scegliete di scaricare le applicazioni di esempio e di eseguirle in seguito, selezionate gli appositi collegamenti. Potete eseguire applicazioni AIR in qualsiasi momento:
 - In Windows, facendo doppio clic sull'icona dell'applicazione sul desktop o selezionandola nel menu Start di Windows.
 - In Mac OS, facendo doppio clic sull'icona dell'applicazione, che per impostazione predefinita è installata nella cartella Applicazioni della vostra directory utente (ad esempio, in Macintosh HD/Utenti/JoeUser/Applicazioni/).

Nota: verificate se sono disponibili aggiornamenti a queste istruzioni nelle note sulla versione di AIR, che si trovano al seguente indirizzo: http://www.adobe.com/go/learn_air_relnotes_it.

Aggiornamenti di Adobe AIR

Adobe rilascia periodicamente aggiornamenti di Adobe AIR che contengono nuove funzionalità o correzioni di problemi di lieve entità. La funzionalità di notifica e aggiornamento automatico consente di inviare automaticamente una notifica agli utenti quando è disponibile una versione aggiornata di Adobe AIR.

Gli aggiornamenti di Adobe AIR garantiscono un funzionamento corretto del prodotto e spesso contengono importanti miglioramenti a livello di sicurezza. Adobe consiglia di eseguire l'aggiornamento alla versione più recente di Adobe AIR ogni volta che viene rilasciata una nuova versione, soprattutto se quest'ultima riguarda un aggiornamento relativo alla sicurezza.

Per impostazione predefinita, all'avvio di un'applicazione AIR il runtime verifica se è disponibile un aggiornamento. Questo controllo viene eseguito se sono trascorse più di due settimane dall'ultima verifica degli aggiornamenti. Se è disponibile un aggiornamento, AIR lo scarica in background.

È possibile disattivare la funzionalità di aggiornamento automatico utilizzando l'applicazione SettingsManager di AIR. L'applicazione SettingsManager di AIR è disponibile per il download all'indirizzo <http://airdownload.adobe.com/air/applications/SettingsManager/SettingsManager.air>.

Il processo di installazione standard di Adobe AIR prevede la connessione a <http://airinstall.adobe.com> per l'invio di informazioni di base sull'ambiente di installazione, ad esempio la lingua e la versione del sistema operativo in uso. Queste informazioni vengono trasmesse una sola volta per ogni installazione e consentono ad Adobe di confermare che l'installazione è stata eseguita correttamente. Non vengono raccolte né trasmesse informazioni che consentono l'identificazione personale dell'utente.

Aggiornamento di runtime autonomi

Se distribuite l'applicazione con un pacchetto di runtime autonomo, il runtime autonomo non viene aggiornato automaticamente. Per garantire la protezione dei vostri utenti, dovete monitorare gli aggiornamenti pubblicati da Adobe e aggiornare l'applicazione con l'ultima versione del runtime disponibile, ogni volta che viene pubblicata un'importante modifica di sicurezza.

Capitolo 3: Operazioni con le API AIR

Adobe® AIR® include funzionalità che non sono disponibili per il contenuto SWF in esecuzione in Adobe® Flash® Player.

Sviluppatori ActionScript 3.0

Le API di Adobe AIR sono documentate nei due manuali seguenti:

- [Guida per gli sviluppatori di ActionScript 3.0](#)
- [Guida di riferimento di ActionScript 3.0 per la piattaforma Adobe Flash](#)

Sviluppatori HTML

Se realizzate applicazioni AIR basate su HTML, le API disponibili in JavaScript tramite il file AIRAliases.js (vedete [Accessing AIR API classes from JavaScript](#)) sono documentate nei due manuali seguenti:

- [HTML Developer's Guide for Adobe AIR](#)
- [Adobe AIR API Reference for HTML Developers](#)

Classi ActionScript 3.0 specifiche per AIR

La tabella seguente contiene le classi di runtime specifiche per Adobe AIR. Queste classi non sono disponibili per il contenuto SWF eseguito in Adobe® Flash® Player nel browser.

Sviluppatori HTML

Le classi disponibili in JavaScript tramite il file AIRAliases.js sono elencate in [Adobe AIR API Reference for HTML Developers](#).

Classe	Pacchetto ActionScript 3.0	Aggiunta nella versione AIR
ARecord	flash.net.dns	2.0
AAAARecord	flash.net.dns	2.0
ApplicationUpdater	air.update	1.5
ApplicationUpdaterUI	air.update	1.5
AudioPlaybackMode	flash.media	3.0
AutoCapitalize	flash.text	3.0
BrowserInvokeEvent	flash.events	1.0
CameraPosition	flash.media	3.0
CameraRoll	flash.media	2.0
CameraRollBrowseOptions	flash.media	3.0
CameraUI	flash.media	2.5

Classe	Pacchetto ActionScript 3.0	Aggiunta nella versione AIR
CertificateStatus	flash.security	2.0
CompressionAlgorithm	flash.utils	1.0
DatagramSocket	flash.net	2.0
DatagramSocketDataEvent	flash.events	2.0
DNSResolver	flash.net.dns	2.0
DNSResolverEvent	flash.events	2.0
DockIcon	flash.desktop	1.0
DownloadErrorEvent	air.update.events	1.5
DRMAuthenticateEvent	flash.events	1.0
DRMDeviceGroup	flash.net.drm	3.0
DRMDeviceGroupErrorEvent	flash.net.drm	3.0
DRMDeviceGroupEvent	flash.net.drm	3.0
DRMManagerError	flash.errors	1.5
EncryptedLocalStore	flash.data	1.0
ExtensionContext	flash.external	2.5
File	flash.filesystem	1.0
FileListEvent	flash.events	1.0
FileMode	flash.filesystem	1.0
FileStream	flash.filesystem	1.0
FocusDirection	flash.display	1.0
GameInput	flash.ui	3.0
GameInputControl	flash.ui	3.0
GameInputControlType	flash.ui	3.6 e versioni precedenti; eliminato a partire dalla versione 3.7
GameInputDevice	flash.ui	3.0
GameInputEvent	flash.ui	3.0
GameInputFinger	flash.ui	3.6 e versioni precedenti; eliminato a partire dalla versione 3.7
GameInputHand	flash.ui	3.6 e versioni precedenti; eliminato a partire dalla versione 3.7
Geolocation	flash.sensors	2.0
GeolocationEvent	flash.events	2.0
HTMLHistoryItem	flash.html	1.0
HTMLHost	flash.html	1.0

Classe	Pacchetto ActionScript 3.0	Aggiunta nella versione AIR
HTMLLoader	flash.html	1.0
HTMLPDFCapability	flash.html	1.0
HTMLSWFCapability	flash.html	2.0
HTMLUncaughtScriptExceptionEvent	flash.events	1.0
HTMLWindowCreateOptions	flash.html	1.0
Icon	flash.desktop	1.0
IFilePromise	flash.desktop	2.0
ImageDecodingPolicy	flash.system	2.6
Interactivelcon	flash.desktop	1.0
InterfaceAddress	flash.net	2.0
InvokeEvent	flash.events	1.0
InvokeEventReason	flash.desktop	1.5.1
IPVersion	flash.net	2.0
IURIDereferencer	flash.security	1.0
LocationChangeEvent	flash.events	2.5
MediaEvent	flash.events	2.5
MediaPromise	flash.media	2.5
MediaType	flash.media	2.5
MXRecord	flash.net.dns	2.0
NativeApplication	flash.desktop	1.0
NativeDragActions	flash.desktop	1.0
NativeDragEvent	flash.events	1.0
NativeDragManager	flash.desktop	1.0
NativeDragOptions	flash.desktop	1.0
NativeMenu	flash.display	1.0
NativeMenuItem	flash.display	1.0
NativeProcess	flash.desktop	2.0
NativeProcessExitEvent	flash.events	2.0
NativeProcessStartupInfo	flash.desktop	2.0
NativeWindow	flash.display	1.0
NativeWindowBoundsEvent	flash.events	1.0
NativeWindowDisplayState	flash.display	1.0
NativeWindowDisplayStateEvent	flash.events	1.0
NativeWindowInitOptions	flash.display	1.0

Classe	Pacchetto ActionScript 3.0	Aggiunta nella versione AIR
NativeWindowRenderMode	flash.display	3.0
NativeWindowResize	flash.display	1.0
NativeWindowSystemChrome	flash.display	1.0
NativeWindowType	flash.display	1.0
NetworkInfo	flash.net	2.0
NetworkInterface	flash.net	2.0
NotificationType	flash.desktop	1.0
OutputProgressEvent	flash.events	1.0
PaperSize	flash.printing	2.0
PrintMethod	flash.printing	2.0
PrintUIOptions	flash.printing	2.0
PTRRecord	flash.net.dns	2.0
ReferencesValidationSetting	flash.security	1.0
ResourceRecord	flash.net.dns	2.0
RevocationCheckSettings	flash.security	1.0
Screen	flash.display	1.0
ScreenMouseEvent	flash.events	1.0
SecureSocket	flash.net	2.0
SecureSocketMonitor	air.net	2.0
ServerSocket	flash.net	2.0
ServerSocketConnectEvent	flash.events	2.0
ServiceMonitor	air.net	1.0
SignatureStatus	flash.security	1.0
SignerTrustSettings	flash.security	1.0
SocketMonitor	air.net	1.0
SoftKeyboardType	flash.text	3.0
SQLCollationType	flash.data	1.0
SQLColumnNameStyle	flash.data	1.0
SQLColumnSchema	flash.data	1.0
SQLConnection	flash.data	1.0
SQLException	flash.errors	1.0
SQLExceptionEvent	flash.events	1.0
SQLExceptionOperation	flash.errors	1.0
SQLEvent	flash.events	1.0

Classe	Pacchetto ActionScript 3.0	Aggiunta nella versione AIR
SQLIndexSchema	flash.data	1.0
SQLMode	flash.data	1.0
SQLResult	flash.data	1.0
SQLSchema	flash.data	1.0
SQLSchemaResult	flash.data	1.0
SQLStatement	flash.data	1.0
SQLTableSchema	flash.data	1.0
SQLTransactionLockType	flash.data	1.0
SQLTriggerSchema	flash.data	1.0
SQLUpdateEvent	flash.events	1.0
SQLViewSchema	flash.data	1.0
SRVRecord	flash.net.dns	2.0
StageAspectRatio	flash.display	2.0
StageOrientation	flash.display	2.0
StageOrientationEvent	flash.events	2.0
StageText	flash.text	3.0
StageTextInitOptions	flash.text	3.0
StageWebView	flash.media	2.5
StatusFileUpdateErrorEvent	air.update.events	1.5
StatusFileUpdateEvent	air.update.events	1.5
StatusUpdateErrorEvent	air.update.events	1.5
StatusUpdateEvent	air.update.events	1.5
StorageVolume	flash.filesystem	2.0
StorageVolumeChangeEvent	flash.events	2.0
StorageVolumeInfo	flash.filesystem	2.0
SystemIdleMode	flash.desktop	2.0
SystemTrayIcon	flash.desktop	1.0
TouchEventIntent	flash.events	3.0
UpdateEvent	air.update.events	1.5
Updater	flash.desktop	1.0
URLFilePromise	air.desktop	2.0
URLMonitor	air.net	1.0
URLRequestDefaults	flash.net	1.0
XMLSignatureValidator	flash.security	1.0

Classi di Flash Player con funzionalità specifiche per AIR

Le classi seguenti sono disponibili per il contenuto SWF in esecuzione nel browser, ma AIR offre proprietà o metodi aggiuntivi:

Pacchetto	Classe	Proprietà, metodo o evento	Aggiunta nella versione AIR
flash.desktop	Clipboard	supportsFilePromise	2.0
	ClipboardFormats	BITMAP_FORMAT	1.0
		FILE_LIST_FORMAT	1.0
		FILE_PROMISE_LIST_FORMAT	2.0
		URL_FORMAT	1.0
flash.display	LoaderInfo	childSandboxBridge	1.0
		parentSandboxBridge	1.0
	Stage	assignFocus()	1.0
		autoOrients	2.0
		deviceOrientation	2.0
		nativeWindow	1.0
		orientation	2.0
		Evento orientationChange	2.0
		Evento orientationChanging	2.0
		setAspectRatio	2.0
		setOrientation	2.0
		softKeyboardRect	2.6
		supportedOrientations	2.6
		supportsOrientationChange	2.0
	NativeWindow	owner	2.6
		listOwnedWindows	2.6
	NativeWindowInitOptions	owner	2.6

Pacchetto	Classe	Proprietà, metodo o evento	Aggiunta nella versione AIR
flash.events	Event	CLOSING	1.0
		DISPLAYING	1.0
		PREPARING	2.6
		EXITING	1.0
		HTML_BOUNDS_CHANGE	1.0
		HTML_DOM_INITIALIZE	1.0
		HTML_RENDER	1.0
		LOCATION_CHANGE	1.0
		NETWORK_CHANGE	1.0
		STANDARD_ERROR_CLOSE	2.0
		STANDARD_INPUT_CLOSE	2.0
		STANDARD_OUTPUT_CLOSE	2.0
		USER_IDLE	1.0
		USER_PRESENT	1.0
		HTTPStatusEvent	HTTP_RESPONSE_STATUS
	responseHeaders		1.0
	responseURL		1.0
	KeyboardEvent	commandKey	1.0
		controlKey	1.0

Pacchetto	Classe	Proprietà, metodo o evento	Aggiunta nella versione AIR
flash.net	FileReference	extension	1.0
		Evento httpResponseStatus	1.0
		uploadUnencoded()	1.0
	NetStream	Evento drmAuthenticate	1.0
		Evento onDRMContentData	1.5
		preloadEmbeddedData()	1.5
		resetDRMvouchers()	1.0
		setDRMAuthenticationCredentials()	1.0
	URLRequest	authenticate	1.0
		cacheResponse	1.0
		followRedirects	1.0
		idleTimeout	2.0
		manageCookies	1.0
		useCache	1.0
		userAgent	1.0
URLStream	evento httpResponseStatus	1.0	

Pacchetto	Classe	Proprietà, metodo o evento	Aggiunta nella versione AIR
flash.printing	PrintJob	active	2.0
		copies	2.0
		firstPage	2.0
		isColor	2.0
		jobName	2.0
		lastPage	2.0
		maxPixelsPerInch	2.0
		paperArea	2.0
		printableArea	2.0
		printer	2.0
		printers	2.0
		selectPaperSize()	2.0
		showPageSetupDialog()	2.0
		start2()	2.0
		supportsPageSetupDialog	2.0
	terminate()	2.0	
		PrintJobOptions	pixelsPerInch
		printMethod	2.0
flash.system	Capabilities	languages	1.1
	LoaderContext	allowLoadBytesCodeExecution	1.0
	Security	APPLICATION	1.0
flash.ui	KeyLocation	D_PAD	2.5

Per la maggior parte, questi nuovi metodi e proprietà sono disponibili solo per il contenuto della sandbox di sicurezza di AIR. I nuovi membri delle classi URLRequest, tuttavia, sono disponibili anche per il contenuto in esecuzione in altre sandbox.

I metodi `ByteArray.compress()` e `ByteArray.uncompress()` includono ciascuno un nuovo parametro `algorithm` che consente di scegliere tra la compressione di tipo deflate e zlib. Questo parametro è disponibile soltanto per il contenuto in esecuzione in AIR.

Componenti Flex specifici per AIR

Quando si sviluppa il contenuto per Adobe AIR, sono disponibili i seguenti componenti Adobe® Flex™ MX:

- [FileEvent](#)
- [FileSystemComboBox](#)
- [FileSystemDataGrid](#)

- [FileSystemEnumerationMode](#)
- [FileSystemHistoryButton](#)
- [FileSystemList](#)
- [FileSystemSizeDisplayMode](#)
- [FileSystemTree](#)
- [FlexNativeMenu](#)
- [HTML](#)
- [Window](#)
- [WindowedApplication](#)
- [WindowedSystemManager](#)

Flex 4 comprende inoltre i seguenti componenti spark AIR:

- [Window](#)
- [WindowedApplication](#)

Per ulteriori informazioni sui componenti Flex AIR, vedete [Using the Flex AIR components](#).

Capitolo 4: Strumenti della piattaforma Adobe Flash per lo sviluppo AIR

Per sviluppare applicazioni AIR potete utilizzare gli strumenti della piattaforma Adobe Flash indicati di seguito.

Per gli sviluppatori ActionScript 3.0 (Flash e Flex):

- Adobe Flash Professional (vedete [Pubblicazione per Adobe AIR](#))
- SDK Adobe Flex 3.x e 4.x (vedete “[Configurazione di Flex SDK](#)” a pagina 20 e “[ADT \(AIR Developer Tool\)](#)” a pagina 171)
- Adobe Flash Builder (vedete [Developing AIR Applications with Flash Builder](#))

Per gli sviluppatori HTML e Ajax:

- Adobe AIR SDK (vedete “[Installazione di AIR SDK](#)” a pagina 18 e “[ADT \(AIR Developer Tool\)](#)” a pagina 171)
- Adobe Dreamweaver CS3, CS4, CS5 (vedete [Estensione AIR per Dreamweaver](#))

Installazione di AIR SDK

Adobe AIR SDK contiene i seguenti strumenti della riga di comando, che potete utilizzare per avviare le applicazioni e creare i relativi pacchetti:

ADL (AIR Debug Launcher) Consente di eseguire le applicazioni AIR senza prima installarle. Vedete “[ADL \(AIR Debug Launcher\)](#)” a pagina 165.

ADT (AIR Developer Tool) Inserisce le applicazioni AIR in pacchetti di installazione che possono essere distribuiti agli utenti. Vedete “[ADT \(AIR Developer Tool\)](#)” a pagina 171.

Per poter utilizzare gli strumenti della riga di comando AIR dovete installare Java. Potete utilizzare la Macchina virtuale Java di JRE o JDK (versione 1.5 o successiva). Java JRE e Java JDK sono disponibili all'indirizzo <http://java.sun.com/>.

L'esecuzione del tool ADT richiede almeno 2 GB di memoria del sistema.

***Nota:** non è necessario che gli utenti finali installino Java per eseguire le applicazioni AIR.*

Per una rapida panoramica sulla creazione di un'applicazione AIR con AIR SDK, vedete “[Creazione della prima applicazione AIR basata su HTML con AIR SDK](#)” a pagina 35.

Download e installazione di AIR SDK

Per scaricare e installare AIR SDK, attenetevi alle seguenti istruzioni:

Installare AIR SDK in Windows

- Scaricate il file di installazione di AIR SDK.
- AIR SDK viene distribuito come file di archivio standard. Per installare AIR, estraete il contenuto dell'SDK in una cartella del computer (ad esempio C:\Programmi\Adobe\AIRSDK o C:\AIRSDK).

- Gli strumenti ADL e ADT si trovano nella cartella bin di AIR SDK. Aggiungete il percorso di questa cartella alla variabile di ambiente PATH.

Installare AIR SDK in Mac OS X

- Scaricate il file di installazione di AIR SDK.
- AIR SDK viene distribuito come file di archivio standard. Per installare AIR, estraete il contenuto dell'SDK in una cartella del computer (ad esempio /Utenti/<nome utente>/Applicazioni/AIRSDK).
- Gli strumenti ADL e ADT si trovano nella cartella bin di AIR SDK. Aggiungete il percorso di questa cartella alla variabile di ambiente PATH.

Installare AIR SDK in Linux

- L'SDK è disponibile in formato tbz2.
- Per installare l'SDK, create la cartella in cui desiderate decomprimerlo e utilizzate il seguente comando: tar -jxvf <percorso di AIR-SDK.tbz2>

Per informazioni sull'uso degli strumenti di AIR SDK, vedete Creazione di un'applicazione AIR mediante gli strumenti della riga di comando.

Contenuto di AIR SDK

La tabella seguente descrive la funzione dei file contenuti in AIR SDK:

Cartella SDK	Descrizione dei file o degli strumenti
bin	ADL (AIR Debug Launcher) consente di eseguire un'applicazione AIR senza prima crearne il pacchetto e installarla. Per ulteriori informazioni sull'uso di questo strumento, vedete " ADL (AIR Debug Launcher) " a pagina 165. ADT (AIR Developer Tool) crea il pacchetto dell'applicazione in un file AIR che può essere distribuito agli utenti. Per ulteriori informazioni sull'uso di questo strumento, vedete " ADT (AIR Developer Tool) " a pagina 171.
frameworks	La directory libs contiene le librerie di codice da utilizzare nelle applicazioni AIR. La directory projects contiene il codice per le librerie SWF e SWC compilate.
include	La directory include contiene il file di intestazione C-language per la scrittura delle estensioni native.
install	La directory install contiene i driver USB di Windows per i dispositivi Android. (Sono i driver forniti da Google in Android SDK.)
lib	Contiene il codice di supporto per i tool di AIR SDK.

Cartella SDK	Descrizione dei file o degli strumenti
runtimes	<p>I runtime AIR per i dispositivi desktop e mobili.</p> <p>Il runtime desktop viene usato da ADL per avviare le applicazioni AIR prima che siano state inserite in un pacchetto o installate.</p> <p>I runtime AIR per Android (pacchetti APK) possono essere installati sui dispositivi o emulatori Android a scopo di sviluppo e test. Per dispositivi ed emulatori vengono utilizzati pacchetti APK distinti. (Il runtime AIR pubblico per Android è disponibile su Android Market.)</p>
samples	<p>Questa cartella contiene un esempio di file descrittore dell'applicazione, un esempio della funzionalità di installazione invisibile (badge.swf) e le icone predefinite dell'applicazione AIR.</p>
templates	<p>descriptor-template.xml - Modello del file descrittore dell'applicazione, obbligatorio per tutte le applicazioni AIR. Per una descrizione dettagliata del file descrittore dell'applicazione, vedete "File descrittore delle applicazioni AIR" a pagina 212.</p> <p>In questa cartella si trovano anche i file di schema per la struttura XML del descrittore dell'applicazione per ogni versione release di AIR.</p>

Configurazione di Flex SDK

Per sviluppare applicazioni Adobe® AIR® con Adobe® Flex™, potete scegliere tra le seguenti opzioni:

- Potete scaricare e installare Adobe® Flash® Builder™, che fornisce strumenti integrati con cui potete creare progetti Adobe AIR ed eseguire il test, il debug e la creazione di pacchetti per le applicazioni AIR. Vedete "Creazione della prima applicazione AIR Flex desktop in Flash Builder" a pagina 22.
- Potete scaricare Adobe® Flex™ SDK e sviluppare applicazioni Flex AIR utilizzando il vostro editor di testo preferito e gli strumenti della riga di comando.

Per una rapida panoramica sulla creazione di un'applicazione AIR con Flex SDK, vedete "Creazione della prima applicazione AIR desktop con Flex SDK" a pagina 39.

Installare Flex SDK

Per poter creare applicazioni AIR con gli strumenti della riga di comando dovete installare Java. Potete utilizzare la Macchina virtuale Java di JRE o JDK (versione 1.5 o successiva). Java JRE e JDK sono disponibili all'indirizzo <http://java.sun.com/>.

Nota: non è necessario che gli utenti finali installino Java per eseguire le applicazioni AIR.

Flex SDK vi fornisce l'API e gli strumenti della riga di comando AIR che vi consentono di eseguire la creazione di pacchetti, la compilazione e il debug per le applicazioni AIR.

- 1 Se non lo avete già fatto, scaricate Flex SDK da <http://opensource.adobe.com/wiki/display/flexsdk/Downloads>.
- 2 Inserite il contenuto dell'SDK in una cartella (ad esempio Flex SDK).
- 3 Copiate il contenuto di AIR SDK sui file di Flex SDK.

Nota: sui computer Mac, assicuratevi di copiare o sostituire i singoli file delle cartelle SDK, non le intere directory. Per impostazione predefinita, quando in un sistema Mac si copia una directory sopra una directory con lo stesso nome, i file presenti nella directory di destinazione vengono rimossi, non uniti al contenuto dell'altra directory. Potete usare il comando `ditto` in una finestra di terminale per unire l'SDK di AIR all'SDK Flex: `ditto air_sdk_folder flex_sdk_folder`

4 Le utilità della riga di comando AIR si trovano nella cartella bin.

Configurazione di kit SDK esterni

Lo sviluppo di applicazioni per Android e iOS richiede che vengano scaricati i file di provisioning, i kit SDK o altri tool di sviluppo dai siti dei produttori delle piattaforme.

Per informazioni su come scaricare e installare Android SDK, vedete [Sviluppatori Android: installazione del kit SDK](#). A partire da AIR 2.6, non è più necessario scaricare Android SDK. AIR SDK infatti ora include i componenti di base necessari per installare e avviare i pacchetti APK. Tuttavia, Android SDK può rivelarsi utile in varie attività di sviluppo, come la creazione e l'esecuzione di emulatori software e la cattura di schermate del dispositivo.

Un kit SDK esterno non è richiesto per lo sviluppo per l'ambiente iOS. Tuttavia, sono necessari alcuni certificati speciali e file di provisioning. Per ulteriori informazioni, vedete [Come ottenere file per sviluppatori da Apple](#).

Capitolo 5: Creazione della prima applicazione AIR

Creazione della prima applicazione AIR Flex desktop in Flash Builder

Per una rapida illustrazione pratica del funzionamento di Adobe® AIR®, attenetevi alle istruzioni seguenti per creare una semplice applicazione AIR "Hello World" basata su file SWF e generare il relativo pacchetto utilizzando Adobe® Flash® Builder.

Se ancora non l'avete fatto, scaricate e installate Flash Builder. Scaricate e installate la versione più recente di Adobe AIR dal percorso seguente: www.adobe.com/go/air_it.

Creare un progetto AIR

Flash Builder include gli strumenti che consentono di sviluppare applicazioni AIR e creare i relativi pacchetti.

Per creare applicazioni AIR in Flash Builder o Flex Builder, dovete eseguire le stesse operazioni richieste per la creazione di progetti di applicazioni basati su Flex, ovvero dovete definire un nuovo progetto.

- 1 Aprite Flex Builder.
- 2 Selezionate File > New (Nuovo) > Flex Project (Progetto Flex).
- 3 Immettete AIRHelloWorld come nome del progetto.
- 4 In Flex le applicazioni AIR sono considerate un tipo di applicazione. Potete scegliere tra due opzioni:
 - Un'applicazione web che viene eseguita in Adobe® Flash® Player
 - Un'applicazione desktop che viene eseguita in Adobe AIR

Selezionate Desktop come tipo di applicazione.

- 5 Fate clic su Finish (Fine) per creare il progetto.

Inizialmente i progetti AIR sono composti da due file: il file MXML principale e un file XML dell'applicazione (detto file descrittore dell'applicazione). Quest'ultimo specifica le proprietà dell'applicazione.

Per ulteriori informazioni, vedete [Developing AIR applications with Flash Builder](#) (Sviluppo di applicazioni AIR con Flash Builder).

Scrivere il codice dell'applicazione AIR

Per scrivere il codice per l'applicazione "Hello World", modificate il file MXML (AIRHelloWorld.mxml) aperto nell'editor. (Se il file non è aperto, apritelo mediante Project Navigator.)

Le applicazioni AIR Flex per il desktop sono contenute all'interno del tag MXML WindowedApplication. Il tag MXML WindowedApplication crea una finestra semplice che include controlli di base, ad esempio la barra del titolo e il pulsante Chiudi.

- 1 Aggiungete un attributo `title` al componente WindowedApplication e assegnategli il valore "Hello World":

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/mx"
                        title="Hello World">

</s:WindowedApplication>
```

- 2 Aggiungete un componente Label all'applicazione inserendolo all'interno del tag WindowedApplication, impostate la proprietà text del componente Label su "Hello AIR" e impostate i vincoli di layout per centrare il testo, come indicato di seguito:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/mx"
                        title="Hello World">

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>

</s:WindowedApplication>
```

- 3 Aggiungete il blocco di stile seguente subito dopo il tag WindowedApplication di apertura e prima del tag del componente Label appena immesso:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    s|WindowedApplication
    {

        skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
        background-color:#999999;
        background-alpha:"0.7";
    }
</fx:Style>
```

Queste impostazioni di stile si applicano a tutta l'applicazione e assicurano il rendering dello sfondo della finestra in un grigio leggermente trasparente.

Il codice dell'applicazione ha ora l'aspetto seguente:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                        xmlns:s="library://ns.adobe.com/flex/spark"
                        xmlns:mx="library://ns.adobe.com/flex/mx"
                        title="Hello World">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|WindowedApplication
        {

            skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
            background-color:#999999;
            background-alpha:"0.7";
        }
    </fx:Style>

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>

</s:WindowedApplication>
```

A questo punto modificate alcune impostazioni nel descrittore dell'applicazione per assicurare la trasparenza all'applicazione:


- 1 Nel riquadro Flex Navigator individuate il file descrittore dell'applicazione nella directory di origine del progetto. Se il progetto è denominato AIRHelloWorld, il file è denominato AIRHelloWorld-app.xml.
- 2 Fate doppio clic sul file descrittore dell'applicazione per modificarlo in Flex Builder.
- 3 Nel codice XML individuate le righe commentate per le proprietà `systemChrome` e `transparent` (della proprietà `initialWindow`). Rimuovete i commenti (ovvero i delimitatori di commento "`<!--`" e "`-->`").
- 4 Impostate il valore di testo della proprietà `systemChrome` su `none`, come nell'esempio seguente:

```
<systemChrome>none</systemChrome>
```
- 5 Impostate il valore di testo della proprietà `transparent` su `true`, come nell'esempio seguente:

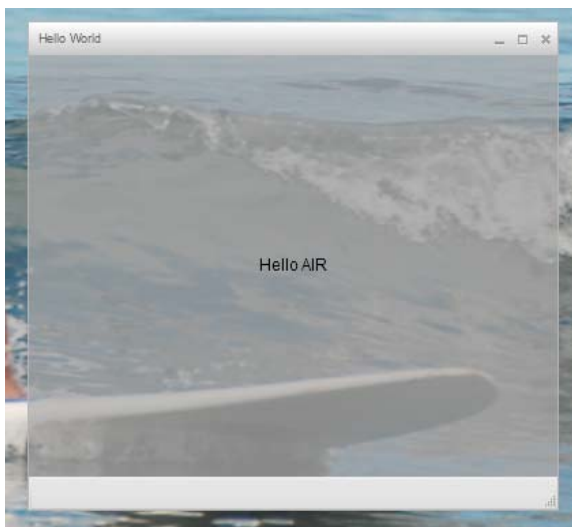
```
<transparent>true</transparent>
```
- 6 Salvate il file.

Eseguire il test dell'applicazione AIR

Per eseguire il test del codice dell'applicazione che avete scritto, eseguitelo in modalità di debug.

- 1 Fate clic sul pulsante Debug  nella barra degli strumenti principale. Potete inoltre selezionare il comando Run (Esegui) > Debug > AIRHelloWorld.

L'applicazione AIR risultante dovrebbe avere un aspetto analogo all'esempio seguente:



- 2 Mediante le proprietà `horizontalCenter` e `verticalCenter` del controllo Label, il testo viene posizionato al centro della finestra. Spostate o ridimensionate la finestra come fareste con qualsiasi altra applicazione desktop.

Nota: se non riuscite a compilare l'applicazione, correggete gli eventuali errori di sintassi o di ortografia che avete inserito involontariamente nel codice. Gli errori e gli avvisi sono visualizzati nella visualizzazione Problems (Problemi) in Flash Builder.

Creare un pacchetto dell'applicazione AIR, firmarla ed eseguirla

A questo punto potete creare un pacchetto dell'applicazione "Hello World" in un file AIR da distribuire. Un file AIR è un file di archivio che contiene i file dell'applicazione, ovvero tutti i file presenti nella cartella bin del progetto. In questo esempio semplice questi file sono il file SWF e il file XML dell'applicazione. Il pacchetto AIR viene distribuito agli utenti che lo utilizzano per installare l'applicazione. Un passaggio obbligatorio di questo processo è la firma digitale.

- 1 Verificate che l'applicazione non generi errori di compilazione e venga eseguita come previsto.
- 2 Selezionate Project > Export Release Build.
- 3 Controllate che il progetto AIRHelloWorld e l'applicazione AIRHelloWorld.mxml siano indicati come progetto e applicazione.
- 4 Selezionate l'opzione Export as signed AIR package, quindi fare clic su Avanti.
- 5 Se già disponete di un certificato digitale, fate clic su Sfoglia per cercarlo e selezionatelo.
- 6 Se dovete creare un certificato digitale autofirmato, selezionate Create.
- 7 Immettete le informazioni obbligatorie e fate clic su OK.
- 8 Fate clic su Fine per generare il pacchetto AIR denominato AIRHelloWorld.air.

A questo punto potete installare ed eseguire l'applicazione da Project Navigator in Flash Builder o dal file system facendo doppio clic sul file AIR.

Creazione della prima applicazione AIR desktop con Flash Professional

Per una rapida dimostrazione pratica del funzionamento di Adobe® AIR®, seguite le istruzioni fornite in questo argomento per creare una semplice applicazione AIR "Hello World" e generare il relativo pacchetto utilizzando Adobe® Flash® Professional.

Se ancora non l'avete fatto, scaricate e installate Adobe AIR dal percorso seguente: www.adobe.com/go/air_it.

Creazione dell'applicazione Hello World in Flash

La procedura per la creazione di applicazioni Adobe AIR in Flash è analoga a quella per la creazione di altri file FLA. La procedura descritta di seguito spiega come creare una semplice applicazione Hello World utilizzando Flash Professional.

Creazione dell'applicazione "Hello World"

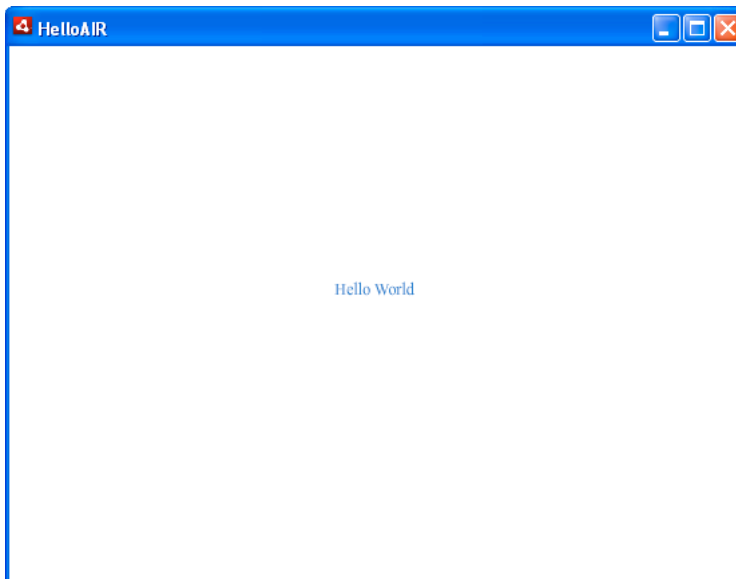
- 1 Avviate Flash.
- 2 Nella schermata di benvenuto, fate clic su AIR per creare un file FLA vuoto con le impostazioni di pubblicazione di Adobe AIR.
- 3 Selezionate lo strumento Testo nel pannello Strumenti e create un campo di testo statico (quello predefinito) al centro dello stage. Createlo di dimensioni sufficienti a contenere 15-20 caratteri.
- 4 Digitate il testo "Hello World" nel campo.
- 5 Salvate il file dandogli un nome, ad esempio HelloAIR.

Provare l'applicazione

- 1 Premete Ctrl + Invio o selezionate Controllo -> Prova filmato -> Prova per provare l'applicazione in Adobe AIR.
- 2 Per utilizzare la funzione Debug filmato, aggiungete innanzitutto all'applicazione il codice ActionScript. Potete provarla rapidamente aggiungendo una dichiarazione trace come la seguente:

```
trace("Running AIR application using Debug Movie");
```
- 3 Premete Ctrl + Maiusc + Invio o selezionate Debug -> Debug filmato -> Debug per eseguire l'applicazione con Debug filmato.

L'applicazione Hello World avrà l'aspetto seguente:



Creare il pacchetto dell'applicazione

- 1 Selezionate File > Pubblica.
- 2 Firmate il pacchetto Adobe AIR con un certificato digitale esistente oppure create un certificato autofirmato procedendo nel modo seguente:
 - a Fate clic sul pulsante Nuovo accanto al campo Certificato.
 - b Compilate i campi Nome editore, Unità organizzativa, Nome organizzazione, E-mail, Paese, Password e Conferma password.
 - c Specificate il tipo di certificato. L'opzione Tipo del certificato si riferisce al livello di protezione: 1024-RSA utilizza una chiave a 1024 bit (minore protezione) e 2048-RSA utilizza una chiave a 2048 bit (maggiore protezione).
 - d Salvate le informazioni in un file di certificato compilando la voce Salva con nome oppure facendo clic sul pulsante Sfoglia per specificare la cartella desiderata, ad esempio *C:/Temp/mycert.pfx*. Al termine, fate clic su OK.
 - e Flash visualizza di nuovo la finestra di dialogo Firma digitale. Il percorso e il nome file del certificato autofirmato appena creato appaiono nella casella di testo Certificato. Se non vengono visualizzati, immetteteli oppure fate clic sul pulsante Sfoglia per individuarli e selezionarli.

- f Nel campo di testo Password della finestra di dialogo Firma digitale, immettete la stessa password che avete specificato al punto b. Per ulteriori informazioni sulla firma delle applicazioni Adobe AIR, vedete “[Firma digitale di un file AIR](#)” a pagina 196.
- 3 Per creare il file dell'applicazione e del programma di installazione, fate clic sul pulsante Pubblica (in Flash CS4 e CS5, fate clic sul pulsante OK). Prima di creare il file AIR è necessario eseguire Prova filmato o Debug filmato per creare il file SWF e i file application.xml.
- 4 Per installare l'applicazione, fate doppio clic sul file AIR (*application.air*) nella stessa cartella in cui avete salvato l'applicazione stessa.
- 5 Fate clic sul pulsante Installa nella finestra di dialogo Installazione applicazione.
- 6 Ricontrollate le impostazioni in Preferenze installazione e Percorso e accertatevi che la casella di controllo Avvia l'applicazione dopo l'installazione contenga un segno di spunta, quindi fate clic su Continua.
- 7 Fate clic su Fine quando appare il messaggio Installazione completata.

Creazione della prima applicazione AIR per Android con Flash Professional

Per sviluppare applicazioni AIR for Android, dovete scaricare l'estensione Flash Professional CS5 per Android da [Adobe Labs](#).

Dovete anche scaricare e installare Android SDK dal sito web Android, come descritto in [Sviluppatori Android: installazione del kit SDK](#).

Creare un progetto

- 1 Aprite Flash Professional CS5.
- 2 Create un nuovo progetto AIR for Android.

La schermata iniziale di Flash Professional include un collegamento per creare un'applicazione AIR for Android. Potete anche selezionare File > Nuovo e quindi selezionare il modello AIR for Android.

- 3 Salvate il documento come HelloWorld fla.

Scrivere il codice

Poiché questa esercitazione non riguarda direttamente la scrittura del codice, limitatevi a usare lo strumento Testo per scrivere "Hello, World!" sullo stage.

Impostare le proprietà dell'applicazione.

- 1 Selezionate File > Impostazioni AIR for Android.
- 2 Nella scheda Generali, configurate le seguenti impostazioni:
 - File di output: HelloWorld.apk
 - Nome applicazione: HelloWorld
 - ID applicazione: HelloWorld
 - Numero di versione: 0.0.1
 - Proporzioni: verticale

3 Nella scheda Distribuzione, configurate le seguenti impostazioni:

- Certificato: indicate un certificato di firma codice AIR valido. Potete fare clic sul pulsante Crea per creare un nuovo certificato. (Le applicazioni Android distribuite su Android Market devono avere certificati validi almeno fino al 2033.) Immettete la password del certificato nel campo Password.
- Tipo di distribuzione Android: Debug
- Dopo la pubblicazione: selezionate entrambe le opzioni
- Immettete il percorso del tool ADB nella sottodirectory tools di Android SDK.

4 Chiudete la finestra di dialogo Impostazioni AIR for Android facendo clic su OK.

L'applicazione non necessita di icone o di autorizzazioni in questa fase del processo di sviluppo. La maggior parte delle applicazioni AIR scritte per Android richiede alcune autorizzazioni per accedere alle funzioni protette. Dovete impostare tali autorizzazioni solo se veramente richieste per l'applicazione specifica, poiché gli utenti potrebbero essere scoraggiati dall'uso dell'applicazione se richiede troppe autorizzazioni.

5 Salvate il file.

Creare il pacchetto dell'applicazione e installarlo sul dispositivo Android

- 1 Verificate che il debug via USB sia abilitato sul dispositivo. Potete attivare il debug USB nella sezione Applicazioni > Sviluppo delle Impostazioni.
- 2 Connettete il dispositivo al computer con un cavo USB.
- 3 Installate il runtime AIR, se non lo avete già fatto, accedendo Android Market e scaricando Adobe AIR. (Potete anche installare AIR localmente utilizzando il “[Comando ADT installRuntime](#)” a pagina 183. I pacchetti Android utilizzabili nei dispositivi ed emulatori Android sono inclusi in AIR SDK.)
- 4 Selezionate File > Pubblica.

Flash Professional crea il file APK, installa l'applicazione sul dispositivo Android collegato e la avvia.

Creazione della prima applicazione AIR per iOS

AIR 2.6 o successivo, iOS 4.2 o successivo

Potete codificare, compilare e testare le funzioni di base di un'applicazione iOS utilizzando esclusivamente strumenti Adobe. Tuttavia, per installare e provare un'applicazione iOS su un dispositivo e distribuirla, dovete aderire al programma Apple iOS Developer, un servizio gratuito. Una volta effettuata l'iscrizione al programma iOS Developer, potete accedere al sito iOS Provisioning Portal, dove è possibile procurarsi le risorse e i file indicati di seguito, necessarie per installare un'applicazione su un dispositivo a scopo di test e successiva distribuzione. Tali risorse e file sono:

- Certificati di sviluppo e di distribuzione
- ID applicazione
- File di provisioning di sviluppo e di distribuzione

Creare il contenuto dell'applicazione

Create un file SWF che visualizza il testo "Hello world!". Potete effettuare questa operazione utilizzando Flash Professional, Flash Builder o un altro ambiente IDE. In questo esempio vengono utilizzati un editor di testo e il compilatore SWF della riga di comando inclusi in Flex SDK.

- 1 Create una directory in una posizione comoda in cui salvare il file dell'applicazione. Create un file denominato *HelloWorld.as* e modificalo nel vostro editor di codice preferito.
- 2 Aggiungete il codice seguente:

```
package{

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldAutoSize;

    public class HelloWorld extends Sprite
    {
        public function HelloWorld():void
        {
            var textField:TextField = new TextField();
            textField.text = "Hello World!";
            textField.autoSize = TextFieldAutoSize.LEFT;

            var format:TextFormat = new TextFormat();
            format.size = 48;

            textField.setTextFormat ( format );
            this.addChild( textField );
        }
    }
}
```

- 3 Compilate la classe usando il compilatore amxmlc:

```
amxmlc HelloWorld.as
```

Un file SWF, *HelloWorld.swf*, viene creato nella stessa cartella.

Nota: l'esempio presuppone che sia stata impostata la variabile d'ambiente dei percorsi includendo la directory che contiene amxmlc. Per informazioni sull'impostazione del percorso, vedete "Variabili d'ambiente dei percorsi" a pagina 316. In alternativa, potete digitare l'intero percorso di amxmlc e degli altri tool della riga di comando usati nell'esempio.

Creare un disegno icona e un disegno schermata iniziale per l'applicazione

Tutte le applicazioni iOS dispongono di icone che vengono visualizzate nell'interfaccia utente dell'applicazione iTunes e sullo schermo del dispositivo.

- 1 Create una directory all'interno della directory di progetto e denominatela Icone.
- 2 Create tre file PNG nella directory Icone e denominateli Icon_29.png, Icon_57.png e Icon_512.png.
- 3 Modificate i file PNG per creare il disegno appropriato per l'applicazione. Le risoluzioni dei file devono essere 29x29 pixel, 57x57 pixel e 512x512 pixel. Per questa prova, potete semplicemente utilizzare quadrati a tinta unita.

Nota: quando inviate un'applicazione ad App Store di Apple, utilizzate una versione JPG (non una versione PNG) del file da 512 pixel. Potete utilizzare la versione PNG durante la prova delle versioni di sviluppo di un'applicazione.

Tutte le applicazioni iPhone visualizzano un'immagine iniziale mentre l'applicazione viene caricata sull'iPhone. Definite l'immagine iniziale in un file PNG:

- 1 Nella directory di sviluppo principale, create un file PNG denominato `Default.png`. (Non collocate questo file nella sottodirectory `icons`. Accertatevi di denominare il file `Default.png`, con una D maiuscola.)
- 2 Modificate il file in modo che sia largo 320 pixel e alto 480 pixel. Per il momento, il contenuto può essere un semplice rettangolo bianco. (Potrete cambiare questo aspetto in seguito.)

Per informazioni dettagliate su queste immagini, vedete [“Icone dell'applicazione”](#) a pagina 91.

Creare il file descrittore dell'applicazione

Create un file descrittore dell'applicazione che specifichi le proprietà di base dell'applicazione. Potete effettuare questa operazione utilizzando un ambiente IDE quale Flash Builder o un editor di testo.

- 1 Nella cartella del progetto che contiene `HelloWorld.as`, create un file XML denominato `HelloWorld-app.xml`. Modificate questo file nel vostro editor XML preferito.
- 2 Aggiungete il codice XML seguente:

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/2.7" minimumPatchLevel="0">
  <id>change_to_your_id</id>
  <name>Hello World iOS</name>
  <versionNumber>0.0.1</versionNumber>
  <filename>HelloWorld</filename>
  <supportedProfiles>mobileDevice</supportedProfiles>
  <initialWindow>
    <content>HelloWorld.swf</content>
    <title>Hello World!</title>
  </initialWindow>
  <icon>
    <image29x29>icons/AIRApp_29.png</image29x29>
    <image57x57>icons/AIRApp_57.png</image57x57>
    <image512x512>icons/AIRApp_512.png</image512x512>
  </icon>
</application>
```

Per semplicità, in questo esempio vengono impostate solo alcune delle proprietà disponibili.

Nota: se usate AIR 2 o una versione precedente, dovete utilizzare l'elemento `<version>` anziché `<versionNumber>`.

- 3 Impostate l'ID applicazione in modo che corrisponda all'ID applicazione specificato nel sito iOS Provisioning Portal. (Non includete la porzione del bundle seed casuale all'inizio dell'ID.)
- 4 Provate l'applicazione con ADL:

```
adl HelloWorld-app.xml -screenize iPhone
```

ADL dovrebbe aprire una finestra sul desktop nella quale compare il testo *Hello World!*. In caso contrario, controllate il codice di origine e il descrittore dell'applicazione per verificare se contengono errori.

Compilare il file IPA

A questo punto potete compilare il file del programma di installazione IPA con ADT. Dovete disporre del certificato sviluppatore Apple e della chiave private in formato file P12, nonché del profilo di provisioning di sviluppo Apple.

Eseguite l'utilità ADT con le opzioni seguenti, sostituendo i valori `keystore`, `storepass` e `provisioning-profile` con quelli appropriati:

```
adt -package -target ipa-debug
    -keystore iosPrivateKey.p12 -storetype pkcs12 -storepass qwerty12
    -provisioning-profile ios.mobileprovision
    HelloWorld.ipa
    HelloWorld-app.xml
    HelloWorld.swf icons Default.png
```

(Utilizzate una sola riga di comando; le interruzioni di riga nell'esempio sono presenti solo per facilitare la leggibilità.)

ADT genera il file del programma di installazione dell'applicazione iOS, *HelloWorld.ipa*, nella directory del progetto. La compilazione del file IPA può richiedere alcuni minuti.

Installare l'applicazione su un dispositivo

Per installare l'applicazione iOS a scopo di test:

- 1 Aprite l'applicazione iTunes.
- 2 Se non l'avete già fatto, aggiungete il profilo di provisioning per questa applicazione a iTunes. In iTunes, selezionate File > Aggiungi alla Libreria, quindi scegliete il file del profilo di provisioning (il cui tipo è mobileprovision).

Per provare l'applicazione sul dispositivo per sviluppatori, utilizzate per il momento il profilo di provisioning di sviluppo.

In seguito, durante la distribuzione di un'applicazione a iTunes Store, utilizzate il profilo di distribuzione. Per distribuire l'applicazione ad hoc a più dispositivi senza passare per iTunes Store, utilizzate il profilo di provisioning ad hoc.

Per ulteriori informazioni sui profili di provisioning, vedete “[Configurazione per iOS](#)” a pagina 69.

- 3 Alcune versioni di iTunes non sostituiscono l'applicazione se la stessa versione dell'applicazione è già installata. In questo caso, cancellate l'applicazione dal dispositivo e dall'elenco delle applicazioni in iTunes.
- 4 Fate doppio clic sul file IPA per l'applicazione. Il file dovrebbe essere visualizzato nell'elenco delle applicazioni in iTunes.
- 5 Collegare il dispositivo alla porta USB del computer.
- 6 In iTunes, controllate la scheda Applicazione per il dispositivo e verificate che l'applicazione sia selezionata nell'elenco delle applicazioni da installare.
- 7 Selezionate il dispositivo nell'elenco a sinistra dell'applicazione iTunes. Quindi fate clic sul pulsante Sincronizza. Al termine della sincronizzazione, l'applicazione Hello World viene visualizzata sull'iPhone.

Se la nuova versione non è installata, eliminatela dal dispositivo e dall'elenco delle applicazioni in iTunes, quindi ripetete questa procedura. Questa situazione può verificarsi se la versione installata utilizza lo stesso ID applicazione e la stessa versione.

Modificare il grafico della schermata iniziale

Prima di compilare l'applicazione, create un file Default.png (vedete “[Creare un disegno icona e un disegno schermata iniziale per l'applicazione](#)” a pagina 29). Questo file PNG serve come immagine di avvio durante il caricamento dell'applicazione. Durante la prova dell'applicazione sull'iPhone, potreste aver notato questo schermo vuoto all'avvio.

Modificate questa immagine per farla corrispondere alla schermata di avvio dell'applicazione ("Hello World!"):

- 1 Aprite l'applicazione sul dispositivo. Quando viene visualizzato il primo testo "Hello World", tenete premuto il pulsante Home (sotto lo schermo). Tenendo premuto il pulsante Home, premete il pulsante Power/Sleep (nella parte superiore dell'iPhone). Questa operazione consente di scattare un'istantanea e di inviarla all'album Rullino fotografico.
- 2 Caricate l'immagine nel computer di sviluppo trasferendo le fotografie da iPhoto o da un'altra applicazione di trasferimento foto. (In Mac OS, potete anche utilizzare l'applicazione Acquisizione immagine.)

Potete anche inviare la foto via e-mail al computer di sviluppo:

- Aprite l'applicazione Foto.
 - Aprite l'album Rullino fotografico.
 - Aprite l'istantanea acquisita.
 - Toccate l'immagine, quindi toccate il pulsante (freccia) "avanti" nell'angolo inferiore sinistro. Fate clic su Invia foto per e-mail e inviate l'immagine a voi stessi.
- 3 Sostituite il file Default.png (nella directory di sviluppo) con una versione PNG dell'immagine acquisita dallo schermo.
 - 4 Ricompilate l'applicazione (vedete "Compilare il file IPA" a pagina 30) e reinstallatela sul dispositivo.

L'applicazione utilizza ora la nuova schermata di avvio durante il caricamento.

Nota: potete creare qualsiasi disegno preferite per il file Default.png, purché sia delle dimensioni corrette (320 x 480 pixel). Tuttavia, spesso è preferibile che l'immagine Default.png corrisponda allo stato iniziale dell'applicazione.

Creazione della prima applicazione AIR basata su HTML con Dreamweaver

Per una rapida illustrazione pratica del funzionamento di Adobe® AIR®, attenetevi alle istruzioni seguenti per creare un'applicazione "Hello World" semplice basata su HTML e generarne il pacchetto mediante l'estensione Adobe® AIR® per Dreamweaver®.

Se ancora non l'avete fatto, scaricate e installate Adobe AIR dal percorso seguente: www.adobe.com/go/air_it.

Per istruzioni sull'installazione dell'estensione Adobe AIR per Dreamweaver, vedete [Installare l'estensione AIR per Dreamweaver](#).

Per una panoramica dell'estensione, compresi i requisiti di sistema, vedete [Estensione AIR per Dreamweaver](#).

Nota: le applicazioni AIR basate su HTML possono essere sviluppate solo per i profili desktop e extendedDesktop. il profilo mobile non è supportato.

Preparare i file dell'applicazione

L'applicazione Adobe AIR deve avere una pagina iniziale e tutte le relative pagine devono essere definite in un sito Dreamweaver:

- 1 Avviate Dreamweaver e controllate che sia definito un sito.
- 2 Aprite una nuova pagina HTML selezionando File > Nuovo, HTML nella colonna Tipo di pagina e selezionando Nessuno nella colonna Layout, quindi facendo clic su Crea.

3 Nella nuova pagina digitate **Hello World!**

Questo esempio è molto semplice ma, se lo desiderate, potete assegnare al testo uno stile e aggiungere contenuto alla pagina, collegare altre pagine a questa pagina iniziale e così via.

- 4** Salvate la pagina (File > Salva) con il nome hello_world.html. La pagina deve essere salvata in un sito Dreamweaver. Per ulteriori informazioni sui siti Dreamweaver, vedete la Guida di Dreamweaver.

Creare l'applicazione Adobe AIR

- 1** Verificate che la pagina hello_world.html sia aperta nella finestra del documento di Dreamweaver. Per istruzioni su come creare la pagina, vedete la sezione precedente.

- 2** Selezionate Sito > Impostazioni applicazione AIR.

La maggior parte delle impostazioni obbligatorie nell'applicazione AIR e nella finestra di dialogo Impostazioni viene inserita automaticamente. Dovrete tuttavia selezionare il contenuto iniziale (o la pagina iniziale) dell'applicazione.

- 3** Fate clic sul pulsante Sfoglia accanto all'opzione Contenuto iniziale, passate alla pagina hello_world.html e selezionatela.

- 4** Accanto all'opzione Firma digitale fate clic sul pulsante Imposta.

Una firma digitale garantisce che il codice di un'applicazione non sia stato modificato o danneggiato da quando è stato creato dall'autore del software ed è obbligatoria per tutte le applicazioni Adobe AIR.

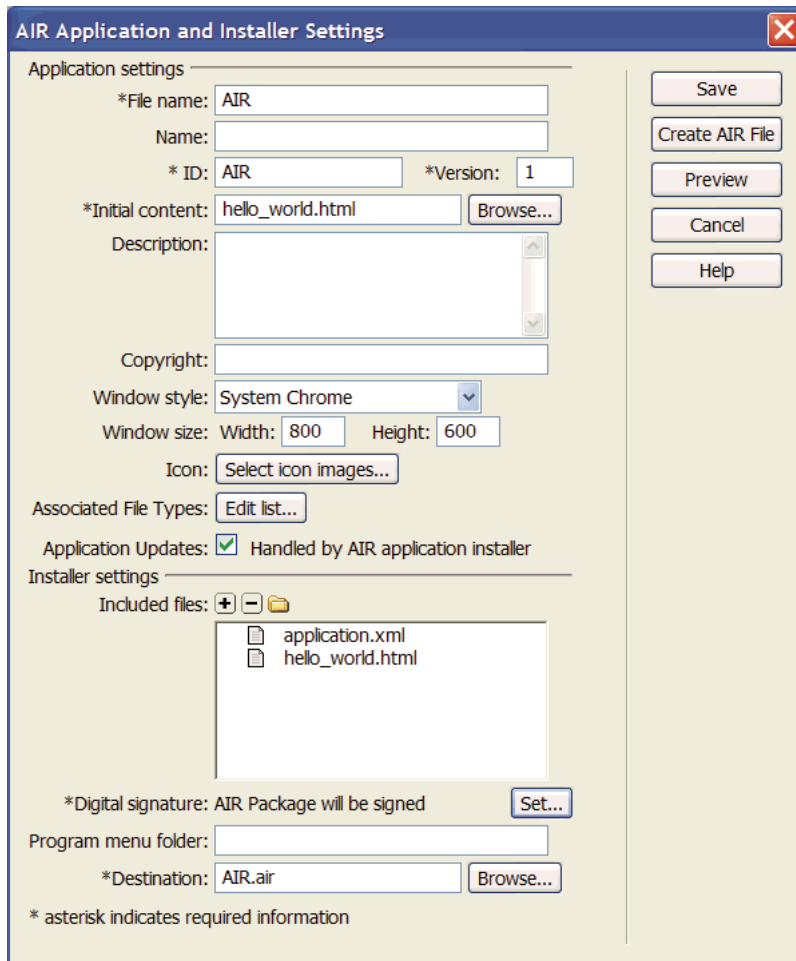
- 5** Nella finestra di dialogo Firma digitale selezionate l'opzione per la firma del pacchetto AIR con un certificato digitale e fate clic sul pulsante Crea. Se invece avete accesso a un certificato digitale, potete fare clic sul pulsante Sfoglia e selezionarne uno.

- 6** Compilate i campi necessari nella finestra di dialogo Crea certificato digitale autofirmato. Dovrete specificare il vostro nome, immettere una password e confermarla e digitare un nome per il file del certificato digitale. Il certificato digitale viene salvato nella cartella principale del sito.

- 7** Fate clic su OK per tornare alla finestra di dialogo Firma digitale.

- 8** Nella finestra di dialogo Firma digitale immettete la password specificata per il certificato digitale e fate clic su OK.

L'applicazione AIR completa e la finestra di dialogo delle impostazioni del programma di installazione potrebbero avere l'aspetto seguente:



Per ulteriori spiegazioni su tutte le opzioni della finestra di dialogo e come modificarle, vedete [Creazione di un'applicazione AIR in Dreamweaver](#).

9 Fate clic sul pulsante Crea file AIR.

Il file dell'applicazione Adobe AIR viene creato e salvato nella cartella principale del sito. Viene inoltre creato un file application.xml e salvato nella stessa cartella. Questo file viene utilizzato come manifesto per la definizione di diverse proprietà dell'applicazione.

Installare l'applicazione su un desktop

In seguito alla creazione, il file dell'applicazione può essere installato in qualsiasi desktop.

1 Spostate il file dell'applicazione Adobe AIR all'esterno del sito Dreamweaver sul desktop oppure su un desktop diverso.

Questo passaggio è facoltativo. In realtà, se preferite, potete installare la nuova applicazione sul computer direttamente dalla directory del sito Dreamweaver.

2 Fate doppio clic sul file eseguibile dell'applicazione (file .air) per installare l'applicazione.

Eseguire l'anteprima dell'applicazione Adobe AIR

Potete eseguire l'anteprima delle pagine che faranno parte delle applicazioni AIR in qualsiasi momento. Non è necessario creare un pacchetto dell'applicazione per verificare che aspetto avrà in seguito all'installazione.

- 1 Verificate che la pagina `hello_world.html` sia aperta nella finestra del documento di Dreamweaver.
- 2 Nella barra degli strumenti Documento fate clic sul pulsante Anteprima/debug nel browser e quindi selezionate Anteprima in AIR.

Potete inoltre premere `Ctrl+Maiusc+F12` (Windows) o `Cmd+Maiusc+F12` (Macintosh).

L'anteprima della pagina consente essenzialmente di vedere quello che un utente vedrebbe come pagina iniziale dell'applicazione in seguito all'installazione dell'applicazione sul desktop.

Creazione della prima applicazione AIR basata su HTML con AIR SDK

Per una rapida illustrazione pratica del funzionamento di Adobe® AIR®, attenetevi alle istruzioni seguenti per creare un'applicazione "Hello World" semplice basata su HTML e generarne il pacchetto.

Per iniziare, è necessario che sia installato il runtime e impostato AIR SDK. In questa esercitazione utilizzerete *ADL* (*AIR Debug Launcher*) e *ADT* (*AIR Developer Tool*). ADL e ADT sono programmi di utilità della riga di comando presenti nella directory `bin` di AIR SDK. Vedete “[Installazione di AIR SDK](#)” a pagina 18. In questa esercitazione si presume che abbiate già familiarità con i programmi eseguiti dalla riga di comando e siate in grado di impostare le variabili di ambiente path necessarie per il sistema operativo.

Nota: se siete utenti di Adobe® Dreamweaver®, leggete “[Creazione della prima applicazione AIR basata su HTML con Dreamweaver](#)” a pagina 32.

Nota: le applicazioni AIR basate su HTML possono essere sviluppate solo per i profili `desktop` e `extendedDesktop`. il profilo `mobile` non è supportato.

Creare i file di progetto

Ogni progetto AIR basato su HTML contiene i due file seguenti: un file descrittore dell'applicazione che specifica i metadati dell'applicazione e una pagina HTML di primo livello. Oltre a questi file obbligatori, questo progetto include un file di codice JavaScript, `AIRAliases.js`, nel quale sono definite comode variabili alias per le classi delle API di AIR.

- 1 Create una directory denominata `HelloWorld` in cui inserire i file di progetto.
- 2 Create un file XML denominato `HelloWorld-app.xml`.
- 3 Create un file HTML denominato `HelloWorld.html`.
- 4 Copiate `AIRAliases.js` dalla cartella `frameworks` di AIR SDK nella directory di progetto.

Creare il file descrittore dell'applicazione AIR

Per iniziare la compilazione dell'applicazione AIR, create un file descrittore dell'applicazione XML con la seguente struttura:


```
<application xmlns="...">
  <id>...</id>
  <versionNumber>...</versionNumber>
  <filename>...</filename>
  <initialWindow>
    <content>...</content>
    <visible>...</visible>
    <width>...</width>
    <height>...</height>
  </initialWindow>
</application>
```

- 1 Aprire il file HelloWorld-app.xml per la modifica.
- 2 Aggiungete l'elemento principale <application> con l'attributo per lo spazio dei nomi AIR:
<application xmlns="http://ns.adobe.com/air/application/2.7"> L'ultimo segmento dello spazio dei nomi, "2.7", specifica la versione del runtime richiesta dall'applicazione.
- 3 Aggiungete l'elemento <id>:
<id>examples.html.HelloWorld</id> L'ID applicazione identifica in modo univoco l'applicazione insieme all'ID editore (che viene ricavato dal certificato utilizzato per firmare il pacchetto dell'applicazione). L'ID applicazione viene utilizzato per l'installazione, l'accesso alla directory privata di memorizzazione del file system per l'applicazione, l'accesso allo spazio di memorizzazione crittografato privato e la comunicazione tra applicazioni.
- 4 Aggiungete l'elemento <versionNumber>:
<versionNumber>0.1</versionNumber> Semplifica l'identificazione della versione dell'applicazione installata dagli utenti.
Nota: se usate AIR 2 o una versione precedente, dovete utilizzare l'elemento <version> anziché <versionNumber>.
- 5 Aggiungete l'elemento <filename>:
<filename>HelloWorld</filename> Nome utilizzato per l'eseguibile dell'applicazione, la directory di installazione e altri riferimenti all'applicazione nel sistema operativo.
- 6 Aggiungete l'elemento <initialWindow> contenente gli elementi secondari seguenti per specificare le proprietà della finestra iniziale dell'applicazione:
<content>HelloWorld.html</content> Identifica il file HTML principale da caricare in AIR.
<visible>true</visible> Rende visibile la finestra immediatamente.
<width>400</width> Imposta la larghezza della finestra in pixel.
<height>200</height> Imposta l'altezza della finestra.
- 7 Salvate il file. Il file descrittore dell'applicazione completato dovrebbe avere l'aspetto seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
  <id>examples.html.HelloWorld</id>
  <versionNumber>0.1</versionNumber>
  <filename>HelloWorld</filename>
  <initialWindow>
    <content>HelloWorld.html</content>
    <visible>true</visible>
    <width>400</width>
    <height>200</height>
  </initialWindow>
</application>
```

In questo esempio vengono impostate solo alcune delle proprietà possibili per l'applicazione. Per il set completo delle proprietà dell'applicazione che consente di specificare ad esempio il chrome della finestra, la dimensione della finestra, la trasparenza, la directory di installazione predefinita, i tipi di file associati e le icone dell'applicazione, vedete ["File descrittori delle applicazioni AIR"](#) a pagina 212.

Creare la pagina HTML dell'applicazione

A questo punto dovete creare una pagina HTML semplice da utilizzare come file principale per l'applicazione AIR.

- 1 Aprite il file `HelloWorld.html` per la modifica. Aggiungete il seguente codice HTML:

```
<html>
<head>
  <title>Hello World</title>
</head>
<body onLoad="appLoad()">
  <h1>Hello World</h1>
</body>
</html>
```

- 2 Nella sezione `<head>` dell'HTML importate il file `AIRAliases.js`:

```
<script src="AIRAliases.js" type="text/javascript"></script>
```

In AIR viene definita una proprietà denominata `runtime` sull'oggetto della finestra HTML. La proprietà `runtime` consente di accedere alle classi AIR incorporate utilizzando il nome del pacchetto completo della classe. Per creare un oggetto File di AIR, ad esempio, potreste aggiungere l'istruzione seguente in JavaScript:

```
var textFile = new runtime.flash.filesystem.File("app:/textfile.txt");
```

Il file `AIRAliases.js` definisce alias comodi per le API AIR più utili. Mediante `AIRAliases.js` potreste accorciare il riferimento alla classe File come indicato di seguito:

```
var textFile = new air.File("app:/textfile.txt");
```

- 3 Sotto il tag script `AIRAliases` aggiungete un altro tag script contenente una funzione JavaScript per gestire l'evento `onLoad`:

```
<script type="text/javascript">
function appLoad() {
  air.trace("Hello World");
}
</script>
```

La funzione `appLoad()` chiama semplicemente la funzione `air.trace()`. Il messaggio di trace viene visualizzato nella console dei comandi quando eseguite l'applicazione mediante ADL. Le istruzioni trace possono essere molto utili per il debug.

4 Salvate il file.

Il file `HelloWorld.html` dovrebbe avere ora l'aspetto seguente:

```
<html>
<head>
  <title>Hello World</title>
  <script type="text/javascript" src="AIRAliases.js"></script>
  <script type="text/javascript">
    function appLoad() {
      air.trace("Hello World");
    }
  </script>
</head>
<body onLoad="appLoad()">
  <h1>Hello World</h1>
</body>
</html>
```

Provare l'applicazione

Per eseguire e testare l'applicazione dalla riga di comando, utilizzate l'utilità ADL (AIR Debug Launcher). L'eseguibile di ADL si trova nella directory `bin` di AIR SDK. Se non avete già configurato AIR SDK, vedete [“Installazione di AIR SDK”](#) a pagina 18.

- 1 Aprire una console o shell di comando. Passate alla directory creata per il progetto.
- 2 Eseguite il comando seguente:

```
adl HelloWorld-app.xml
```

Si apre una finestra di AIR in cui è visualizzata l'applicazione. Nella finestra della console viene inoltre visualizzato il messaggio risultante dalla chiamata a `air.trace()`.

Per ulteriori informazioni, vedete [“File descrittori delle applicazioni AIR”](#) a pagina 212.

Creare il file di installazione AIR

Se l'applicazione viene eseguita correttamente, potete utilizzare l'utilità ADT per creare un pacchetto dell'applicazione e inserirlo in un file di installazione AIR. Un file di installazione AIR è un file di archivio che contiene tutti i file dell'applicazione che possono così essere distribuiti agli utenti. Dovete installare Adobe AIR prima di installare il pacchetto di un file AIR.

Per garantire la sicurezza dell'applicazione tutti i file di installazione di AIR devono essere firmati digitalmente. A scopo di sviluppo potete generare un certificato autofirmato di base con ADT o un altro strumento di generazione di certificati. Potete inoltre acquistare un certificato commerciale per la firma di codice da un'autorità di certificati commerciale, ad esempio VeriSign o Thawte. Quando gli utenti installano un file AIR autofirmato, l'editore viene visualizzato come "sconosciuto" durante il processo di installazione perché un certificato autofirmato garantisce solo che il file AIR non sia stato modificato in seguito alla creazione. Chiunque potrebbe autofirmare un file AIR mascherato e presentarlo come se fosse la vostra applicazione. Per i file AIR da rilasciare al pubblico è fortemente consigliato l'uso di un certificato commerciale verificabile. Per una panoramica dei problemi di sicurezza relativi ad AIR, vedete [Sicurezza in AIR](#) (sviluppatori ActionScript) o [AIR Security](#) (sviluppatori HTML).

Generare un certificato autofirmato e una coppia di chiavi

- ❖ Dal prompt dei comandi immettete il comando seguente (l'eseguibile di ADT si trova nella directory `bin` di AIR SDK):

```
adt -certificate -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

Con ADT viene generato un file dell'archivio di chiavi denominato *sampleCert.pfx* che contiene un certificato e la relativa chiave privata.

In questo esempio viene utilizzato il numero minimo di attributi impostabile per un certificato. Il tipo di chiave deve essere *1024-RSA* o *2048-RSA* (vedete “[Firma di applicazioni AIR](#)” a pagina 196).

Creare il file di installazione AIR

❖ Dal prompt dei comandi immettete il comando seguente (su una sola riga):

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air  
HelloWorld-app.xml HelloWorld.html AIRAliases.js
```

Verrà chiesta la password del file dell'archivio di chiavi.

L'argomento *HelloWorld.air* si trova nel file AIR prodotto da ADT. *HelloWorld-app.xml* è il file descrittore dell'applicazione. Gli argomenti successivi sono i file utilizzati dall'applicazione. In questo esempio vengono utilizzati solo due file, ma potete includere il numero di file e directory desiderato. ADT verifica che il file del contenuto principale, *HelloWorld.html*, sia incluso nel pacchetto, ma se dimenticate di includere *AIRAliases.js*, l'applicazione non potrà funzionare.

In seguito alla creazione del pacchetto AIR, potete installare ed eseguire l'applicazione facendo doppio clic sul file del pacchetto. Potete inoltre digitare il nome del file AIR come comando in una finestra di comando o della shell.

Passaggi successivi

In AIR il codice HTML e JavaScript si comporta in genere come in un normale browser Web. In effetti in AIR viene utilizzato lo stesso motore di rendering WebKit utilizzato dal browser Web Safari. Sono tuttavia presenti differenze importanti che vale la pena di comprendere prima di sviluppare applicazioni HTML in AIR. Per ulteriori informazioni su queste differenze e su altri argomenti importanti, vedete [Programming HTML and JavaScript in AIR](#).

Creazione della prima applicazione AIR desktop con Flex SDK

Per una rapida illustrazione pratica del funzionamento di Adobe® AIR®, attenetevi alle istruzioni seguenti per creare un semplice applicazione "Hello World" basata su SWF utilizzando Flex SDK. Questa esercitazione mostra come compilare, provare e impacchettare un'applicazione AIR con i tool della riga di comando disponibili in Flex SDK (Flex SDK include AIR SDK).

Per iniziare, dovete avere installato il runtime e configurato Adobe® Flex™. In questa esercitazione utilizzerete il compilatore *AMXMLC*, *AIR Debug Launcher (ADL)* e *AIR Developer Tool (ADT)*. Questi programmi sono presenti nella directory *bin* di Flex SDK. Vedete “[Configurazione di Flex SDK](#)” a pagina 20.

Creare il file descrittore dell'applicazione AIR

Questa sezione descrive come creare il descrittore dell'applicazione, ovvero un file XML con la seguente struttura:

```
<application xmlns="...">
  <id>...</id>
  <versionNumber>...</versionNumber>
  <filename>...</filename>
  <initialWindow>
    <content>...</content>
    <visible>...</visible>
    <width>...</width>
    <height>...</height>
  </initialWindow>
</application>
```

- 1 Create un file XML denominato `HelloWorld-app.xml` e salvatelo nella directory del progetto.
- 2 Aggiungete l'elemento `<application>` con l'attributo namespace AIR:
`<application xmlns="http://ns.adobe.com/air/application/2.7">` L'ultimo segmento dello spazio dei nomi, "2.7", specifica la versione del runtime richiesta dall'applicazione.
- 3 Aggiungete l'elemento `<id>`:
`<id>samples.flex.HelloWorld</id>` L'ID applicazione identifica in modo univoco l'applicazione insieme all'ID editore (che viene ricavato dal certificato utilizzato per firmare il pacchetto dell'applicazione). La forma consigliata è una stringa in stile DNS inverso delimitata da punti, ad esempio "com.company.AppName". L'ID applicazione viene utilizzato per l'installazione, l'accesso alla directory privata di memorizzazione del file system per l'applicazione, l'accesso allo spazio di memorizzazione crittografato privato e la comunicazione tra applicazioni.
- 4 Aggiungete l'elemento `<versionNumber>`:
`<versionNumber>1.0</versionNumber>` Semplifica l'identificazione della versione dell'applicazione installata dagli utenti.
Nota: se usate AIR 2 o una versione precedente, dovete utilizzare l'elemento `<version>` anziché `<versionNumber>`.
- 5 Aggiungete l'elemento `<filename>`:
`<filename>HelloWorld</filename>` Nome utilizzato per l'eseguibile dell'applicazione, la directory di installazione e analoghi riferimenti nel sistema operativo.
- 6 Aggiungete l'elemento `<initialWindow>` contenente gli elementi secondari seguenti per specificare le proprietà della finestra iniziale dell'applicazione:
`<content>HelloWorld.swf</content>` Identifica il file SWF principale da caricare in AIR.
`<visible>true</visible>` Rende visibile la finestra immediatamente.
`<width>400</width>` Imposta la larghezza della finestra in pixel.
`<height>200</height>` Imposta l'altezza della finestra.
- 7 Salvate il file. Il file descrittore dell'applicazione completo dovrebbe avere l'aspetto seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
  <id>samples.flex.HelloWorld</id>
  <versionNumber>0.1</versionNumber>
  <filename>HelloWorld</filename>
  <initialWindow>
    <content>HelloWorld.swf</content>
    <visible>true</visible>
    <width>400</width>
    <height>200</height>
  </initialWindow>
</application>
```

In questo esempio vengono impostate solo alcune delle proprietà possibili per l'applicazione. Per il set completo delle proprietà dell'applicazione che consente di specificare ad esempio il chrome della finestra, la dimensione della finestra, la trasparenza, la directory di installazione predefinita, i tipi di file associati e le icone dell'applicazione, vedete ["File descrittore delle applicazioni AIR"](#) a pagina 212.

Scrivere il codice dell'applicazione

***Nota:** le applicazioni AIR basate su SWF possono usare una classe principale definita con MXML oppure con Adobe® ActionScript® 3.0. Nell'esempio seguente viene usato un file MXML per definire la classe principale. Il processo di creazione di un'applicazione AIR con una classe principale ActionScript è analogo. Anziché compilare un file MXML ottenendo il file SWF, si compila il file di classe ActionScript. Quando si usa ActionScript, il file di classe principale deve estendere `flash.display.Sprite`.*

Come tutte le applicazioni basate su Flex, le applicazioni AIR realizzate con il framework Flex contengono un file MXML principale. Le applicazioni AIR desktop usano il componente `WindowedApplication` come elemento principale anziché il componente `Application`. Il componente `WindowedApplication` dispone di proprietà, metodi ed eventi che permettono di controllare l'applicazione la sua finestra iniziale. Nelle piattaforme e nei profili per i quali AIR non supporta le finestre multiple, continuate a usare il componente `Application`. Nelle applicazioni Flex per dispositivi mobili, potete anche utilizzare i componenti `View` o `TabbedViewNavigatorApplication`.

La procedura seguente crea l'applicazione Hello World:

- 1 Usando un editor di testo, create un file denominato `HelloWorld.mxml` e aggiungete il codice MXML seguente:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  title="Hello World">
</s:WindowedApplication>
```

- 2 Quindi, aggiungete un componente `Label` all'applicazione (inserirlo fuori del tag `WindowedApplication`).
- 3 Impostate la proprietà `text` del componente `Label` su `"Hello AIR"`.
- 4 Impostate i vincoli del layout in modo da mantenerlo sempre centrato.

L'esempio seguente mostra il codice scritto fino a questo punto:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    title="Hello World">

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

Compilare l'applicazione

Per poter avviare l'applicazione ed eseguirne il debug, è prima necessario compilare il codice MXML con il compilatore amxmlc, per ottenere il file SWF dell'applicazione. Il compilatore amxmlc si trova nella directory bin di Flex SDK. Se volete, potete impostare l'ambiente di percorso del vostro computer in modo che includa la directory bin di Flex SDK. L'impostazione del percorso facilita l'esecuzione delle utilità dalla riga di comando.

- 1 Aprite una shell di comando o una finestra di terminale e andate alla cartella del progetto dell'applicazione AIR.
- 2 Immettete il comando seguente:

```
amxmlc HelloWorld.mxml
```

L'esecuzione di amxmlc produce HelloWorld.swf, che contiene il codice compilato dell'applicazione.

Nota: se non riuscite a compilare l'applicazione, correggete gli eventuali errori di sintassi o di ortografia. Gli errori e gli avvisi vengono visualizzati nella finestra di console utilizzata per eseguire il compilatore amxmlc.

Per ulteriori informazioni, vedete “[Compilazione di file di origine MXML e ActionScript per AIR](#)” a pagina 161.

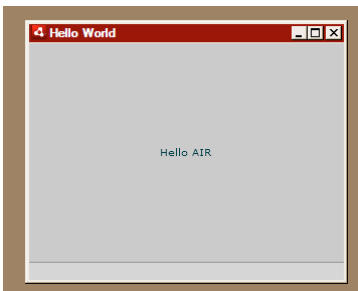
Provare l'applicazione

Per eseguire e testare l'applicazione dalla riga di comando, utilizzate l'utilità ADL (AIR Debug Launcher) per lanciare l'applicazione utilizzando il relativo file descrittore. (Il programma ADL si trova nella directory bin di Flex SDK.)

- ❖ Dal prompt dei comandi, immettete il comando seguente:

```
adl HelloWorld-app.xml
```

L'aspetto dell'applicazione AIR risultante è simile a quello della figura seguente:



Mediante le proprietà horizontalCenter e verticalCenter del controllo Label, il testo viene posizionato al centro della finestra. Spostate o ridimensionate la finestra come fareste con qualsiasi altra applicazione desktop.

Per ulteriori informazioni, vedete “[ADL \(AIR Debug Launcher\)](#)” a pagina 165.

Creare il file di installazione AIR

Se l'applicazione viene eseguita correttamente, potete utilizzare l'utilità ADT per creare un pacchetto dell'applicazione e inserirlo in un file di installazione AIR. Un file di installazione AIR è un file di archivio che contiene tutti i file dell'applicazione che possono così essere distribuiti agli utenti. Dovete installare Adobe AIR prima di installare il pacchetto di un file AIR.

Per garantire la sicurezza dell'applicazione tutti i file di installazione di AIR devono essere firmati digitalmente. A scopo di sviluppo potete generare un certificato autofirmato di base con ADT o un altro strumento di generazione di certificati. Potete inoltre acquistare un certificato commerciale di firma del codice da un'autorità di certificazione commerciale. Quando gli utenti installano un file AIR autofirmato, l'editore viene visualizzato come "sconosciuto" durante il processo di installazione perché un certificato autofirmato garantisce solo che il file AIR non sia stato modificato in seguito alla creazione. Chiunque potrebbe autofirmare un file AIR mascherato e presentarlo come se fosse la vostra applicazione. Per i file AIR da rilasciare al pubblico è fortemente consigliato l'uso di un certificato commerciale verificabile. Per una panoramica dei problemi di sicurezza relativi ad AIR, vedete [Sicurezza in AIR](#) (sviluppatori ActionScript) o [AIR Security](#) (sviluppatori HTML).

Generare un certificato autofirmato e una coppia di chiavi

- ❖ Dal prompt dei comandi, immettete il comando seguente (l'eseguibile di ADT si trova nella directory `bin` di Flex SDK):

```
adt -certificate -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

In questo esempio viene utilizzato il numero minimo di attributi impostabile per un certificato. Il tipo di chiave deve essere *1024-RSA* o *2048-RSA* (vedete ["Firma di applicazioni AIR"](#) a pagina 196).

Creare il pacchetto AIR

- ❖ Dal prompt dei comandi immettete il comando seguente (su una sola riga):

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air  
HelloWorld-app.xml HelloWorld.swf
```

Verrà chiesta la password del file dell'archivio di chiavi. Digitate la password e premete Invio. Per motivi di sicurezza, i caratteri della password non vengono visualizzati.

L'argomento `HelloWorld.air` si trova nel file AIR prodotto da ADT. `HelloWorld-app.xml` è il file descrittore dell'applicazione. Gli argomenti successivi sono i file utilizzati dall'applicazione. In questo esempio vengono utilizzati solo tre file, ma potete includere qualsiasi numero di file e directory.

In seguito alla creazione del pacchetto AIR, potete installare ed eseguire l'applicazione facendo doppio clic sul file del pacchetto. Potete inoltre digitare il nome del file AIR come comando in una finestra di comando o della shell.

Per ulteriori informazioni, vedete ["Creazione del pacchetto di un file di installazione AIR desktop"](#) a pagina 56.

Creazione della prima applicazione AIR for Android con Flex SDK

Per iniziare, è necessario che AIR SDK e Flex SDK siano installati e configurati. In questa esercitazione viene utilizzato il compilatore *AMXMLC* di Flex SDK e i tool *AIR Debug Launcher* (ADL) e *AIR Developer Tool* (ADT) di AIR SDK. Vedete ["Configurazione di Flex SDK"](#) a pagina 20.

Dovete anche scaricare e installare Android SDK dal sito web Android, come descritto in [Sviluppatori Android: installazione del kit SDK](#).

Nota: per informazioni sullo sviluppo per iPhone, vedete [Creazione di un'applicazione iPhone Hello World con Flash Professional CS5](#).

Creare il file descrittore dell'applicazione AIR

Questa sezione descrive come creare il descrittore dell'applicazione, ovvero un file XML con la seguente struttura:

```
<application xmlns="...">
  <id>...</id>
  <versionNumber>...</versionNumber>
  <filename>...</filename>
  <initialWindow>
    <content>...</content>
  </initialWindow>
  <supportedProfiles>...</supportedProfiles>
</application>
```

- 1 Create un file XML denominato `HelloWorld-app.xml` e salvatelo nella directory del progetto.
- 2 Aggiungete l'elemento `<application>` con l'attributo namespace AIR:
`<application xmlns="http://ns.adobe.com/air/application/2.7">` L'ultimo segmento dello spazio dei nomi, "2.7", specifica la versione del runtime richiesta dall'applicazione.
- 3 Aggiungete l'elemento `<id>`:
`<id>samples.android.HelloWorld</id>` L'ID applicazione identifica in modo univoco l'applicazione insieme all'ID editore (che viene ricavato dal certificato utilizzato per firmare il pacchetto dell'applicazione). La forma consigliata è una stringa in stile DNS inverso delimitata da punti, ad esempio "com.company.AppName".
- 4 Aggiungete l'elemento `<versionNumber>`:
`<versionNumber>0.0.1</versionNumber>` Semplifica l'identificazione della versione dell'applicazione installata dagli utenti.
- 5 Aggiungete l'elemento `<filename>`:
`<filename>HelloWorld</filename>` Nome utilizzato per l'eseguibile dell'applicazione, la directory di installazione e analoghi riferimenti nel sistema operativo.
- 6 Aggiungete l'elemento `<initialWindow>` contenente gli elementi secondari seguenti per specificare le proprietà della finestra iniziale dell'applicazione:
`<content>HelloWorld.swf</content>` Identifica il file HTML principale da caricare in AIR.
- 7 Aggiungete l'elemento `<supportedProfiles>`.
`<supportedProfiles>mobileDevice</supportedProfiles>` Specifica che l'applicazione può essere eseguita solo nel profilo mobile.
- 8 Salvate il file. Il file descrittore dell'applicazione completo dovrebbe avere l'aspetto seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
  <id>samples.android.HelloWorld</id>
  <versionNumber>0.0.1</versionNumber>
  <filename>HelloWorld</filename>
  <initialWindow>
    <content>HelloWorld.swf</content>
  </initialWindow>
  <supportedProfiles>mobileDevice</supportedProfiles>
</application>
```

In questo esempio vengono impostate solo alcune delle proprietà possibili per l'applicazione. Vi sono altre impostazioni che potete usare nel file descrittore dell'applicazione. Ad esempio, potete aggiungere `<fullScreen>true</fullScreen>` all'elemento `initialWindow` per realizzare un'applicazione a schermo intero. Per abilitare il debug remoto e le funzioni soggetto a controllo dell'accesso in Android, dovrete anche aggiungere le autorizzazioni Android al descrittore dell'applicazione. Le autorizzazioni non sono necessarie per questa semplice applicazione, quindi non dovete aggiungerle in questo caso.

Per ulteriori informazioni, vedete [“Impostazione delle proprietà delle applicazioni per dispositivi mobili”](#) a pagina 74.

Scrivere il codice dell'applicazione

Create un file denominato `HelloWorld.as` e aggiungete il codice seguente utilizzando un editor di testo:

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

    public class HelloWorld extends Sprite
    {
        public function HelloWorld()
        {
            var textField:TextField = new TextField();
            textField.text = "Hello, World!";
            stage.addChild( textField );
        }
    }
}
```

Compilare l'applicazione

Per poter avviare l'applicazione ed eseguirne il debug, è prima necessario compilare il codice MXML con il compilatore `amxmlc`, per ottenere il file SWF dell'applicazione. Il compilatore `amxmlc` si trova nella directory `bin` di Flex SDK. Se volete, potete impostare l'ambiente di percorso del vostro computer in modo che includa la directory `bin` di Flex SDK. L'impostazione del percorso facilita l'esecuzione delle utilità dalla riga di comando.

- 1 Aprite una shell di comando o una finestra di terminale e andate alla cartella del progetto dell'applicazione AIR.
- 2 Immettete il comando seguente:

```
amxmlc HelloWorld.as
```

L'esecuzione di `amxmlc` produce `HelloWorld.swf`, che contiene il codice compilato dell'applicazione.

Nota: se non riuscite a compilare l'applicazione, correggete gli eventuali errori di sintassi o di ortografia. Gli errori e gli avvisi vengono visualizzati nella finestra di console utilizzata per eseguire il compilatore `amxmlc`.

Per ulteriori informazioni, vedete [“Compilazione di file di origine MXML e ActionScript per AIR”](#) a pagina 161.

Provare l'applicazione

Per eseguire e testare l'applicazione dalla riga di comando, utilizzate l'utilità ADL (AIR Debug Launcher) per lanciare l'applicazione utilizzando il relativo file descrittore. (Il programma ADL si trova nella directory `bin` di AIR SDK e Flex SDK.)

- ❖ Dal prompt dei comandi, immettete il comando seguente:

```
adl HelloWorld-app.xml
```

Per ulteriori informazioni, vedete [“Simulazione del dispositivo con ADL”](#) a pagina 104.

Creare il file del pacchetto APK

Se l'applicazione viene eseguita correttamente, potete utilizzare l'utilità ADT per creare un pacchetto dell'applicazione e inserirlo in un file di pacchetto APK. APK è il formato di file nativo di Android per le applicazioni, che potete distribuire ai vostri utenti.

Tutte le applicazioni Android devono essere firmate. A differenza dei file AIR, le applicazioni Android vengono generalmente firmate con un certificato autofirmato. Il sistema operativo Android non tenta di determinare l'identità dello sviluppatore dell'applicazione. Potete usare un certificato generato da ADT per firmare i pacchetti Android. I certificati utilizzati per le applicazioni inviate ad Android Market devono avere un periodo di validità di almeno 25 anni.

Generare un certificato autofirmato e una coppia di chiavi

- ❖ Dal prompt dei comandi, immettete il comando seguente (l'eseguibile di ADT si trova nella directory `bin` di Flex SDK):

```
adt -certificate -validityPeriod 25 -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

In questo esempio viene utilizzato il numero minimo di attributi impostabile per un certificato. Il tipo di chiave deve essere *1024-RSA* o *2048-RSA* (vedete [“Comando ADT certificate”](#) a pagina 180).

Creare il pacchetto AIR

- ❖ Dal prompt dei comandi immettete il comando seguente (su una sola riga):

```
adt -package -target apk -storetype pkcs12 -keystore sampleCert.p12 HelloWorld.apk  
HelloWorld-app.xml HelloWorld.swf
```

Verrà chiesta la password del file dell'archivio di chiavi. Digitate la password e premete Invio.

Per ulteriori informazioni, vedete [“Creazione del pacchetto di un'applicazione AIR per dispositivi mobili”](#) a pagina 97.

Installare il runtime AIR

Potete installare la versione più recente del runtime AIR sul vostro dispositivo da Android Market. Potete anche installare il runtime incluso nel kit SDK su un dispositivo o su un emulatore Android.

- ❖ Dal prompt dei comandi immettete il comando seguente (su una sola riga):

```
adt -installRuntime -platform android -platformsdk
```

Impostate il flag `-platformsdk` sulla directory di Android SDK (specificate la directory principale della cartella `tools`).

ADT installa il file `Runtime.apk` incluso nel kit SDK.

Per ulteriori informazioni, vedete [“Installare il runtime AIR e le applicazioni per fini di sviluppo”](#) a pagina 113.

Installare l'applicazione AIR

- ❖ Dal prompt dei comandi immettete il comando seguente (su una sola riga):

```
adt -installApp -platform android -platformsdk path-to-android-sdk -package path-to-app
```

Impostate il flag `-platformsdk` sulla directory di Android SDK (specificate la directory principale della cartella `tools`).

Per ulteriori informazioni, vedete “[Installare il runtime AIR e le applicazioni per fini di sviluppo](#)” a pagina 113.

Potete avviare l'applicazione toccando la relativa icona sullo schermo del dispositivo o dell'emulatore.

Capitolo 6: Sviluppo di applicazioni AIR per il desktop

Flusso di lavoro per lo sviluppo di un'applicazione AIR desktop

Il flusso di lavoro per lo sviluppo di un'applicazione AIR è uguale alla maggior parte dei modelli di sviluppo tradizionali: scrittura del codice, compilazione, prova e, verso la fine del ciclo, creazione del pacchetto del file di installazione.

Potete scrivere il codice dell'applicazione utilizzando Flash, Flex e ActionScript e compilarlo con Flash Professional, Flash Builder o i compilatori della riga di comando mxmclc e compc. Potete anche scrivere il codice dell'applicazione utilizzando HTML e JavaScript e saltare la fase di compilazione.

Le applicazioni AIR desktop possono essere testate con il tool ADL, che permette di eseguire un'applicazione senza doverla prima compilare in un pacchetto e installarla. Il debugger Flash è integrato in Flash Professional, Flash Builder, Dreamweaver e nell'IDE Aptana. Potete anche avviare il debugger FDB manualmente quando utilizzate ADL dalla riga di comando. ADL fornisce anche un feedback sotto forma di errori e istruzioni trace.

Tutte le applicazioni AIR devono essere impacchettate in un file di installazione. Il formato file AIR multiplatforma è consigliato tranne che nei seguenti casi:

- Dovete accedere ad API dipendenti dalla piattaforma, quali la classe NativeProcess.
- La vostra applicazione utilizza estensioni native.

In tali casi, potete compilare un'applicazione AIR in un file di installazione nativo specifico della piattaforma.

Applicazioni basate su SWF

- 1 Scrivete il codice MXML o ActionScript.
- 2 Create le risorse necessarie, ad esempio i file bitmap delle icone.
- 3 Create il descrittore dell'applicazione.
- 4 Compilate il codice ActionScript.
- 5 Provate l'applicazione.
- 6 Create il pacchetto di un file AIR e firmatelo utilizzando la destinazione *air*.

Applicazioni basate su HTML

- 1 Scrivete il codice HTML o JavaScript.
- 2 Create le risorse necessarie, ad esempio i file bitmap delle icone.
- 3 Create il descrittore dell'applicazione.
- 4 Provate l'applicazione.
- 5 Create il pacchetto di un file AIR e firmatelo utilizzando la destinazione *air*.

Creazione di programmi di installazione nativi per applicazioni AIR

- 1 Scrivete il codice (ActionScript o HTML e JavaScript).
- 2 Create le risorse necessarie, ad esempio i file bitmap delle icone.
- 3 Create il descrittore dell'applicazione, specificando il profilo *extendedDesktop*.
- 4 Compilate il codice ActionScript, se presente.
- 5 Provate l'applicazione.
- 6 Create il pacchetto dell'applicazione su ciascuna piattaforma di destinazione utilizzando il target *native*.

Nota: il programma di installazione nativo per una piattaforma di destinazione deve essere creato su tale piattaforma. Non potete, ad esempio, creare un programma di installazione per Windows su Mac. Potete usare una macchina virtuale, quale VMWare, per eseguire più piattaforme sullo stesso computer.

Creazione di applicazioni AIR con un pacchetto runtime autonomo

- 1 Scrivete il codice (ActionScript o HTML e JavaScript).
- 2 Create le risorse necessarie, ad esempio i file bitmap delle icone.
- 3 Create il descrittore dell'applicazione, specificando il profilo *extendedDesktop*.
- 4 Compilate il codice ActionScript, se presente.
- 5 Provate l'applicazione.
- 6 Create il pacchetto dell'applicazione su ciascuna piattaforma di destinazione utilizzando il target *bundle*.
- 7 Create un programma di installazione usando i file contenuti nel pacchetto. (L'SDK di AIR non fornisce strumenti per creare un tale programma di installazione, ma sono disponibili numerosi toolkit di terze parti.)

Nota: il pacchetto per una piattaforma di destinazione deve essere creato su tale piattaforma. Non potete, ad esempio, creare pacchetto per Windows su Mac. Potete usare una macchina virtuale, quale VMWare, per eseguire più piattaforme sullo stesso computer.

Impostazione delle proprietà delle applicazioni desktop

Impostate le proprietà di base dell'applicazione nel file descrittore dell'applicazione. Questa sezione descrive le proprietà relative alle applicazioni AIR desktop. Gli elementi del file descrittore dell'applicazione sono descritti in modo dettagliato nella sezione “[File descrittore delle applicazioni AIR](#)” a pagina 212.

Versione del runtime AIR necessaria

Specificate la versione del runtime AIR necessaria per l'applicazione utilizzando lo spazio dei nomi del file descrittore dell'applicazione.

Lo spazio dei nomi, assegnato nell'elemento `application`, determina in larga parte quali funzioni può utilizzare la vostra applicazione. Ad esempio, se l'applicazione usa lo spazio dei nomi AIR 1.5 e l'utente ha installato AIR 3.0, l'applicazione vede il comportamento AIR (anche se il comportamento è cambiato in AIR 3.0). Solo quando cambiate lo spazio dei nomi e pubblicate un aggiornamento l'applicazione avrà accesso al nuovo comportamento e alle nuove funzioni. Le modifiche relative alla sicurezza e a WebKit sono le principali eccezioni a questo criterio.

Specificate lo spazio dei nomi utilizzando l'attributo `xmlns` dell'elemento root `application`:

```
<application xmlns="http://ns.adobe.com/air/application/3.0">
```

Altri argomenti presenti nell’Aiuto

“[application](#)” a pagina 217

Identità dell'applicazione

Diverse impostazioni devono essere univoche per ciascuna applicazione che pubblicate. Tali impostazioni includono l’ID, il nome e il nome del file.

```
<id>com.example.MyApplication</id>  
<name>My Application</name>  
<filename>MyApplication</filename>
```

Altri argomenti presenti nell’Aiuto

“[id](#)” a pagina 234

“[filename](#)” a pagina 229

“[name](#)” a pagina 242

Versione dell'applicazione

Nelle versioni di AIR anteriori alla AIR 2.5, specificate versione dell'applicazione nell'elemento `version`. Potete usare qualsiasi stringa. Il runtime AIR non interpreta la stringa; ad esempio, "2.0" non viene trattato come una versione successiva a "1.0".

```
<!-- AIR 2 or earlier -->  
<version>1.23 Beta 7</version>
```

In AIR 2.5 e versioni successive, specificate la versione dell'applicazione nell'elemento `versionNumber`. L'uso dell'elemento `version` non è più valido. Quando specificate un valore per `versionNumber`, dovete usare una sequenza composta da un massimo di tre numeri separati da punti, come in "0.1.2". Ogni segmento del numero di versione può contenere fino a tre cifre. (In altre parole, "999.999.999" è il numero di versione più alto possibile.) Non dovete includere necessariamente tutti e tre i segmenti nel numero: "1" e "1.0" sono numeri di versione perfettamente validi.

Potete anche specificare un'etichetta per la versione utilizzando l'elemento `versionLabel`. Quando aggiungete un'etichetta di versione, questa viene visualizzata al posto del numero di versione, ad esempio nelle finestre di installazione dell'applicazione AIR.

```
<!-- AIR 2.5 and later -->  
<versionNumber>1.23.7</versionNumber>  
<versionLabel>1.23 Beta 7</versionLabel>
```

Altri argomenti presenti nell’Aiuto

“[version](#)” a pagina 250

“[versionLabel](#)” a pagina 251

“[versionNumber](#)” a pagina 251

Proprietà delle finestre principali

Quando AIR avvia un'applicazione sul desktop, crea una finestra nella quale carica il file SWF o la pagina HTML principale. AIR utilizza gli elementi secondari dell'elemento `initialWindow` per controllare l'aspetto iniziale e il comportamento di questa prima finestra dell'applicazione.

- **content** - Il file SWF principale dell'applicazione nell'elemento secondario `content` dell'elemento `initialWindow`. Quando un'applicazione ha come target dispositivi nel profilo desktop, potete usare un file SWF o HTML.

```
<initialWindow>
  <content>MyApplication.swf</content>
</initialWindow>
```

Dovete includere il file nel pacchetto AIR (usando ADT o il vostro ambiente IDE). Un semplice riferimento al nome nel descrittore dell'applicazione non determina l'inclusione automatica del file nel pacchetto.

- **depthAndStencil** - Specifica di utilizzare il buffer di profondità o di stencil. Solitamente questi buffer vengono utilizzati quando si lavora con contenuti 3D.

```
<depthAndStencil>true</depthAndStencil>
```

- **height** - L'altezza della finestra iniziale.
- **maximizable** - Indica se il chrome di sistema per l'ingrandimento della finestra viene visualizzato.
- **maxSize** - La dimensione massima consentita.
- **minimizable** - Indica se il chrome di sistema per la riduzione a icona della finestra viene visualizzato.
- **minSize** - La dimensione minima consentita.
- **renderMode** - In AIR 3 o successivi, la modalità di rendering può essere impostata su *auto*, *cpu*, *direct* o *gpu* per le applicazioni desktop. Nelle versioni precedenti di AIR, questa impostazione viene ignorata sulle piattaforme desktop. L'impostazione `renderMode` non può essere modificata in fase runtime.

- **auto** - Essenzialmente uguale al modo *cpu*.
- **cpu** - Viene eseguito il rendering degli oggetti di visualizzazione che poi vengono copiati per visualizzare la memoria nel software. `StageVideo` è disponibile solo quando una finestra è in modalità schermo intero. `Stage3D` impiega il rendering del software.
- **direct** - Viene eseguito il rendering degli oggetti di visualizzazione dal software di runtime, tuttavia la copia del fotogramma di cui è stato eseguito il rendering per la visualizzazione della memoria (blitting) è accelerata mediante hardware. `StageVideo` è disponibile. `Stage3D` usa l'accelerazione hardware, se possibile. Se la trasparenza finestra è impostata su *true*, la finestra "ripiega" sul rendering software e il blitting.

*Nota: per poter sfruttare l'accelerazione GPU dei contenuti Flash con AIR per le piattaforme mobili, Adobe consiglia di utilizzare `renderMode="direct"` (ovvero, `Stage3D`) anziché `renderMode="gpu"`. Adobe supporta e consiglia ufficialmente i seguenti framework basati su `Stage3D`: *Starling (2D)* e *Away3D (3D)*. Per maggiori dettagli su `Stage3D` e *Starling/Away3D*, visitate <http://gaming.adobe.com/getstarted/>.*

- **gpu** - accelerazione hardware utilizzata, se disponibile.
- **requestedDisplayResolution** - Se l'applicazione dovrebbe utilizzare la modalità di risoluzione *standard* o *elevata* sui computer MacBook Pro con schermi ad alta risoluzione. Su tutte le altre piattaforme il valore viene ignorato. Se il valore è *standard*, ciascun pixel stage viene renderizzato come quattro pixel sullo schermo. Se il valore è *elevato*, ciascun pixel stage corrisponde a un unico pixel fisico sullo schermo. Il valore specificato viene utilizzato per tutte le finestre dell'applicazione. L'utilizzo dell'elemento `requestedDisplayResolution` per le applicazioni AIR desktop (come elemento secondario dell'elemento `initialWindow`) è disponibile in AIR 3.6 e versioni successive.
- **resizable** - Indica se il chrome di sistema per il ridimensionamento della finestra viene visualizzato.

- **systemChrome** - Indica se viene utilizzato lo stile di finestra standard del sistema operativo. L'impostazione systemChrome di una finestra non può essere modificata in fase runtime.
- **title** - Il titolo della finestra.
- **transparent** - Indica se la finestra esegue una fusione alfa rispetto allo sfondo. La finestra non può usare il chrome di sistema se è attivata la trasparenza. L'impostazione transparent di una finestra non può essere modificata in fase runtime.
- **visible** - Indica se la finestra è visibile non appena viene creata. Per impostazione predefinita, la finestra inizialmente non è visibile, in modo tale che l'applicazione possa disegnarne il contenuto prima di rendersi visibile.
- **width** - La larghezza della finestra.
- **x** - La posizione orizzontale della finestra.
- **y** - La posizione verticale della finestra.

Altri argomenti presenti nell' Aiuto

[“content”](#) a pagina 223

[“depthAndStencil”](#) a pagina 225

[“height”](#) a pagina 233

[“maximizable”](#) a pagina 241

[“maxSize”](#) a pagina 241

[“minimizable”](#) a pagina 242

[“minimizable”](#) a pagina 242

[“minSize”](#) a pagina 242

[“renderMode”](#) a pagina 244

[“requestedDisplayResolution”](#) a pagina 245

[“resizable”](#) a pagina 246

[“systemChrome”](#) a pagina 249

[“title”](#) a pagina 250

[“transparent”](#) a pagina 250

[“visible”](#) a pagina 252

[“width”](#) a pagina 252

[“x”](#) a pagina 252

[“y”](#) a pagina 253

Funzioni desktop

I seguenti elementi controllano l'installazione desktop e le funzioni di aggiornamento.

- **customUpdateUI** - Permette di specificare delle finestre personalizzate per l'aggiornamento di un'applicazione. Se è impostato su `false`, il valore predefinito, vengono utilizzate le finestre di dialogo AIR standard.

- `fileTypes` - Specifica i tipi di file che l'applicazione tenta di registrarsi come applicazione di apertura predefinita. Se esiste già un'applicazione predefinita per l'apertura di un determinato tipo di file, AIR non cambia tale impostazione. Tuttavia, l'applicazione può cambiare la registrazione in fase runtime utilizzando il metodo `setAsDefaultApplication()` dell'oggetto `NativeApplication`. È più "educato" chiedere l'autorizzazione all'utente prima di cambiare le associazioni dei tipi di file con le applicazioni.

***Nota:** la registrazione del tipo di file viene ignorata quando create il pacchetto di un'applicazione come pacchetto runtime autonomo (utilizzando il target `-bundle`). Per registrare un determinato tipo di file, dovete creare un programma di installazione che esegue la registrazione.*

- `installFolder` - Specifica un percorso relativo alla cartella di installazione standard dell'applicazione in cui deve essere installata l'applicazione. Potete usare questa impostazione per specificare un nome di cartella personalizzato oppure anche per raggruppare più applicazioni in una sola cartella comune.
- `programMenuFolder` - Specifica la gerarchia dei menu per il menu Tutti i programmi di Windows. Potete usare questa impostazione per raggruppare più applicazioni nello stesso menu. Se non viene specificata una cartella di menu, la scorciatoia dell'applicazione viene aggiunta direttamente al menu principale.

Altri argomenti presenti nell' Aiuto

[“customUpdateUI”](#) a pagina 224

[“fileTypes”](#) a pagina 230

[“installFolder”](#) a pagina 238

[“programMenuFolder”](#) a pagina 244

Profili supportati

Se la vostra applicazione è pensata unicamente per un ambiente desktop, potete impedirne l'installazione su dispositivi di altri profili escludendo tali profili dall'elenco dei profili supportati. Se l'applicazione utilizza la classe `NativeProcess` o estensioni native, dovete necessariamente supportare il profilo `extendedDesktop`.

Se omettete l'elemento `supportedProfile` nel descrittore dell'applicazione, l'applicazione supporterà tutti i profili definiti. Per limitare il supporto dell'applicazione a un gruppo specifico di profili, elencate tali profili, separati da spazi:

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

Per un elenco delle classi `ActionScript` supportate nei profili `desktop` e `extendedDesktop`, vedete [“Funzionalità supportate dai profili”](#) a pagina 255.

Altri argomenti presenti nell' Aiuto

[“supportedProfiles”](#) a pagina 248

Estensioni native obbligatorie

Le applicazioni che supportano il profilo `extendedDesktop` possono utilizzare le estensioni native.

Dichiarate tutte le estensioni native utilizzate dall'applicazione AIR nel descrittore dell'applicazione. L'esempio seguente illustra la sintassi da adottare per specificare due estensioni native richieste:

```
<extensions>
  <extensionID>com.example.extendedFeature</extensionID>
  <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

L'elemento `extensionID` ha lo stesso valore dell'elemento `id` nel file descrittore dell'estensione. Quest'ultimo è un file XML denominato `extension.xml`, impacchettato nel file ANE fornito dallo sviluppatore dell'estensione nativa.

Icone dell'applicazione

Sul desktop, le icone specificate nel descrittore dell'applicazione vengono utilizzate per il file dell'applicazione, la scorciatoia (collegamento all'applicazione) e il menu dei programmi. Le icone dell'applicazione devono essere fornite come serie di immagini PNG da 16x16, 32x32, 48x48 e 128x128 pixel. Specificate il percorso dei file di icone nell'elemento `icon` del file descrittore dell'applicazione:

```
<icon>
  <image16x16>assets/icon16.png</image16x16>
  <image32x32>assets/icon32.png</image32x32>
  <image48x48>assets/icon48.png</image48x48>
  <image128x128>assets/icon128.png</image128x128>
</icon>
```

Se non fornite un'icona di una determinata dimensione, viene utilizzata la successiva dimensione più grande, adattata al formato richiesto. Se non fornite alcuna icona, viene utilizzata un'icona predefinita del sistema.

Altri argomenti presenti nell' Aiuto

[“icon”](#) a pagina 233

[“imageNxN”](#) a pagina 235

Impostazioni ignorate

Le applicazioni sul desktop ignorano le impostazioni applicazione esclusive delle funzionalità dei profili per dispositivi mobili. Le impostazioni ignorate sono:

- android
- aspectRatio
- autoOrients
- fullScreen
- iPhone
- renderMode (precedente ad AIR 3)
- requestedDisplayResolution
- softKeyboardBehavior

Debug di un'applicazione AIR desktop

Se sviluppate l'applicazione in un ambiente IDE come Flash Builder, Flash Professional o Dreamweaver, i tool di debug sono generalmente incorporati nell'ambiente stesso e potete eseguire il debug dell'applicazione semplicemente avviandola in modalità debug. Se invece non disponete di un IDE che supporta direttamente il debug, potete utilizzare i tool ADL (AIR Debug Launcher) e FDB (Flash Debugger) per facilitare l'attività di debug dell'applicazione.

Altri argomenti presenti nell’Aiuto

[De Monsters: Monster Debugger](#)

“[Debug con AIR HTML Introspector](#)” a pagina 292

Esecuzione di un'applicazione mediante ADL

Mediante ADL potrete eseguire un'applicazione AIR senza prima effettuare la creazione del pacchetto e installarla. Passate il file descrittore dell'applicazione ad ADL sotto forma di parametro, come indicato nell'esempio seguente (prima è necessario compilare il codice ActionScript dell'applicazione):

```
adl myApplication-app.xml
```

ADL stampa le istruzioni trace, le eccezioni runtime e gli errori di analisi sintattica HTML direttamente nella finestra di terminale. Se un processo FDB è in attesa di una connessione in entrata, ADL si connette al debugger.

Potete anche utilizzare ADL per eseguire il debug di un'applicazione AIR che usa estensioni native. Ad esempio:

```
adl -extdir extensionDirs myApplication-app.xml
```

Altri argomenti presenti nell’Aiuto

“[ADL \(AIR Debug Launcher\)](#)” a pagina 165

Stampa di istruzioni trace

Per stampare istruzioni trace nella console utilizzata per l'esecuzione di ADL, aggiungete le istruzioni trace al codice mediante la funzione `trace()`.

Nota: Se le istruzioni `trace()` non vengono visualizzate sulla console, verificate di non aver specificato `ErrorReportingEnable` o `TraceOutputFileEnable` nel file `mm.cfg`. Per maggiori informazioni sulle posizioni di questo file a seconda della piattaforma, vedete [Editing the mm.cfg file](#).

Esempio ActionScript:

```
//ActionScript  
trace("debug message");
```

Esempio JavaScript:

```
//JavaScript  
air.trace("debug message");
```

In JavaScript, per visualizzare i messaggi di debug provenienti dall'applicazione, potete utilizzare le funzioni `alert()` e `confirm()`. Nella console vengono anche stampati i numeri di riga per gli errori di sintassi e le eccezioni JavaScript non rilevate.

Nota: per utilizzare il prefisso `air` mostrato nell'esempio JavaScript, dovete importare il file `AIRAliases.js` nella pagina. Il file si trova nella directory `frameworks` di AIR SDK.

Connessione a FDB (Flash Debugger)

Per eseguire il debug di un'applicazione AIR mediante il debugger Flash, avviate una sessione FDB, quindi avviate l'applicazione usando ADL.

Nota: nelle applicazioni AIR basate su SWF, i file di origine ActionScript devono essere compilati con il flag `-debug`. In Flash Professional selezionate l'opzione *Consenti debug* nella finestra di dialogo *Impostazioni pubblicazione*.

1 Avviate FDB. Il programma FDB si trova nella directory `bin` di Flex SDK.

Nella console viene visualizzato il prompt FDB: `<fdb>`

2 Eseguite il comando `run`: `<fdb>run` [Invio]

3 In un diverso comando o console della shell, avviate una versione di debug dell'applicazione:

```
adl myApp.xml
```

4 Utilizzando i comandi FDB, impostate i punti di interruzione come desiderato.

5 Digitate: `continue` [Invio]

Se un'applicazione AIR è basata su SWF, il debugger controlla solamente l'esecuzione del codice ActionScript. Se l'applicazione AIR è basata su HTML, il debugger controlla solamente l'esecuzione del codice JavaScript.

Per eseguire ADL senza connettersi al debugger, includete l'opzione `-nodebug`:

```
adl myApp.xml -nodebug
```

Per informazioni di base sui comandi FDB, eseguite il comando `help`:

```
<fdb>help [Enter]
```

Per informazioni dettagliate sui comandi FDB, vedete [Using the command-line debugger commands](#) (Uso dei comandi del debugger della riga di comando) nella documentazione di Flex.

Creazione del pacchetto di un file di installazione AIR desktop

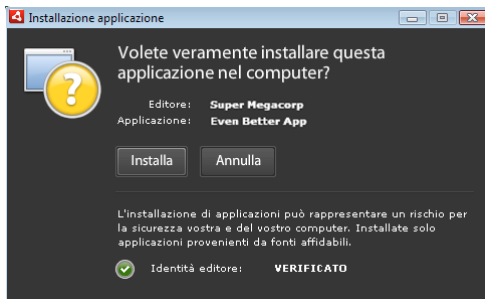
Per ciascuna applicazione AIR devono essere presenti, quanto meno, un file descrittore dell'applicazione e un file principale SWF o HTML. Tutte le altre risorse da installare con l'applicazione devono essere incluse nel pacchetto del file AIR.

Questo articolo descrive la creazione del pacchetto di un'applicazione AIR mediante i tool della riga di comando inclusi nel kit SDK. Per informazioni sulla creazione del pacchetto di un'applicazione mediante uno degli strumenti di authoring di Adobe, vedete:

- Adobe® Flex® Builder™: [Packaging AIR applications with Flex Builder](#).
- Adobe® Flash® Builder™: [Packaging AIR applications with Flash Builder](#).
- Adobe® Flash® Professional: [Pubblicazione per Adobe AIR](#).
- Adobe® Dreamweaver®: [Creazione di un'applicazione AIR in Dreamweaver](#).

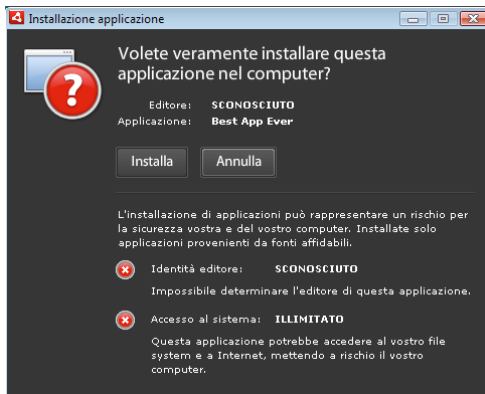
Tutti i file di installazione di AIR devono essere firmati mediante un certificato digitale. Il programma di installazione di AIR utilizza la firma per verificare che il file dell'applicazione non sia stato alterato dal momento in cui è stato firmato. Potete usare un certificato per la firma del codice rilasciato da un'autorità di certificazione oppure un certificato autofirmato.

Utilizzando un certificato rilasciato da un'autorità di certificazione attendibile, offrite garanzie sulla vostra identità di editore agli utenti dell'applicazione. Nella finestra di dialogo di installazione viene segnalato il fatto che la vostra identità è stata verificata dall'autorità di certificazione:



Finestra di conferma dell'installazione per un'applicazione firmata con un certificato attendibile

Se utilizzate un certificato autofirmato, gli utenti non possono verificare la vostra identità di firmatario. Un certificato autofirmato, inoltre, offre minori garanzie che il pacchetto non sia stato alterato poiché un file di installazione legittimo potrebbe essere stato sostituito da uno contraffatto prima di raggiungere l'utente. Nella finestra di dialogo di installazione viene segnalato il fatto che non è possibile verificare l'identità dell'editore. Installando l'applicazione, gli utenti si assumono un rischio di sicurezza maggiore:



Finestra di conferma dell'installazione per un'applicazione firmata con un certificato autofirmato

Per creare un pacchetto e firmare un file AIR in un unico passaggio, è possibile utilizzare il comando `-package` di ADT. È inoltre possibile creare un pacchetto intermedio, non firmato, mediante il comando `-prepare`, quindi firmarlo mediante il comando `-sign` in un passaggio separato.

Nota: le versioni Java dalla 1.5 in poi non accettano i caratteri high-ASCII nelle password utilizzate per proteggere i file di certificato PKCS12. Quando create o esportate il file del certificato per la firma del codice, utilizzate solo caratteri ASCII standard nella password.

Quando si appone la firma sul pacchetto di installazione, ADT contatta automaticamente il server di un'autorità di indicazione di data e ora per la verifica dell'ora. Le informazioni di indicazione di data e ora vengono incluse nel file AIR. Un file AIR che include un'indicazione di data e ora verificata può essere installato in qualsiasi momento. Se ADT non è in grado di connettersi al server di indicazione di data e ora, la creazione del pacchetto viene annullata. È possibile ignorare l'opzione di indicazione di data e ora, ma senza un indicatore di data e ora un'applicazione AIR non è più installabile dopo che il certificato utilizzato per la firma del file di installazione è scaduto.

Se create un pacchetto per aggiornare un'applicazione AIR esistente, il pacchetto deve essere firmato con lo stesso certificato dell'applicazione originale. Se il certificato originale è stato rinnovato o è scaduto negli ultimi 180 giorni, oppure desiderate passare a un nuovo certificato, potete applicare una firma di migrazione. La firma di migrazione prevede che il file dell'applicazione AIR venga firmato sia con il certificato nuovo che con quello vecchio. Utilizzate il comando `-migrate` per applicare la firma di migrazione, come descritto in “Comando ADT migrate” a pagina 179.

Importante: dopo la scadenza del certificato originale è previsto un tassativo periodo di tolleranza di 180 giorni per l'applicazione della firma di migrazione. Senza una firma di migrazione, gli utenti esistenti devono disinstallare la versione corrente dell'applicazione prima di installare la nuova versione. Il periodo di tolleranza vale solo per le applicazioni nelle quali è specificato AIR 1.5.3 (o una versione successiva) nello spazio dei nomi del descrittore dell'applicazione. Per le versioni precedenti del runtime AIR non è previsto alcun periodo di tolleranza.

Prima di AIR 1.1, le firme di migrazione non erano supportate. Per applicare una firma di migrazione dovete creare il pacchetto dell'applicazione con SDK 1.1 o versione successiva.

Le applicazioni distribuite mediante file AIR sono dette applicazioni con profilo desktop. Non è possibile utilizzare ADT per creare il pacchetto di un programma di installazione nativo per un'applicazione AIR se il file descrittore dell'applicazione non supporta il profilo desktop. Potete assegnare una restrizione a questo profilo utilizzando l'elemento `supportedProfiles` nel file descrittore dell'applicazione. Vedete [“Profili dispositivo”](#) a pagina 254 e [“supportedProfiles”](#) a pagina 248.

Nota: le impostazioni nel file descrittore dell'applicazione determinano l'identità di un'applicazione AIR e il relativo percorso di installazione predefinito. Vedete [“File descrittore delle applicazioni AIR”](#) a pagina 212.

ID editore

A partire da AIR 1.5.3, gli ID editore sono stati dichiarati obsoleti. Per le nuove applicazioni (originariamente pubblicate con AIR 1.5.3 o successivo) l'ID editore non è necessario e non deve essere specificato.

Quando aggiornate le applicazioni pubblicate con versioni precedenti di AIR, dovete specificare l'ID editore originale nel file descrittore dell'applicazione. In caso contrario, la versione installata e la versione di aggiornamento dell'applicazione verranno gestite come applicazioni distinte. Se utilizzate un ID differente oppure omettete il tag `publisherID`, l'utente dovrà disinstallare la versione precedente prima di installare la nuova versione.

Per determinare l'ID editore originale, individuate il file `publisherid` nella sottodirectory `META-INF/AIR` in cui è installata l'applicazione originale. La stringa contenuta in questo file è l'ID editore. Per specificare manualmente l'ID editore, dovete specificare il runtime AIR 1.5.3 (o successivo) nella dichiarazione namespace del file descrittore dell'applicazione.

Per le applicazioni pubblicate prima di AIR 1.5.3 (o le applicazioni pubblicate con l'SDK di AIR 1.5.3 specificando una versione precedente di AIR nello spazio dei nomi del descrittore dell'applicazione) viene calcolato un ID editore sulla base del certificato di firma. Questo ID viene utilizzato insieme all'ID applicazione per determinare l'identità dell'applicazione. L'ID editore, se presente, viene utilizzato per i seguenti scopi:

- Come indicazione del fatto che un file AIR è un aggiornamento e non una nuova applicazione da installare
- Come parte della chiave di crittografia per l'archivio locale crittografato
- Come parte del percorso della directory di archiviazione dell'applicazione
- Come parte della stringa di connessione per le connessioni locali
- Come parte della stringa di identità utilizzata per chiamare un'applicazione con l'API AIR interna al browser
- Come parte dell'identificatore OSID (utilizzato per la creazione di programmi di installazione/disinstallazione personalizzati)

Prima di AIR 1.5.3, l'ID editore di un'applicazione poteva cambiare se un aggiornamento dell'applicazione veniva firmato con una firma di migrazione utilizzando un certificato nuovo o rinnovato. Quando un ID editore viene modificato, cambia anche il comportamento delle funzioni AIR associate all'ID. Ad esempio, non è più possibile accedere ai dati presenti nell'archivio locale crittografato, ed eventuali istanze Flash o AIR che creano una connessione locale all'applicazione devono usare il nuovo ID nella stringa di connessione.

In AIR 1.5.3 o successivo, l'ID editore non si basa sul certificato di firma e viene assegnato solo se il descrittore dell'applicazione include il tag publisherID. Non è possibile aggiornare un'applicazione se l'ID editore specificato per il pacchetto AIR di aggiornamento non corrisponde all'ID editore corrente dell'applicazione.

Creazione del pacchetto con ADT

Potete usare il tool della riga di comando ADT di AIR per creare il pacchetto di un'applicazione AIR. Prima di creare il pacchetto è necessario effettuare la compilazione di tutto il codice ActionScript, MXML e di estensione (se presente). Dovete inoltre avere un certificato di firma del codice.

Per informazioni dettagliate sui comandi e le opzioni ADT, vedete “[ADT \(AIR Developer Tool\)](#)” a pagina 171.

Creazione di un pacchetto AIR

Per creare un pacchetto AIR, usate il comando ADT `package`, impostando il tipo di target su `air` per le build di rilascio.

```
adt -package -target air -storetype pkcs12 -keystore ../codesign.p12 myApp.air myApp-app.xml  
myApp.swf icons
```

Nell'esempio, si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316 per ulteriori informazioni.)

Dovete eseguire il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf` e una directory di icone.

Quando eseguite il comando come indicato, ADT richiede la password dell'archivio di chiavi. (I caratteri della password che digitate non vengono sempre visualizzati; quando avete immesso tutti i caratteri della password, premete Invio.)

Creazione di un pacchetto AIR da un file AIRI

Potete firmare un file AIRI per creare un pacchetto AIR installabile:

```
adt -sign -storetype pkcs12 -keystore ../codesign.p12 myApp.airi myApp.air
```

Creazione del pacchetto di un file di installazione nativo desktop

A partire da AIR 2 potete utilizzare ADT per creare programmi di installazione nativi per la distribuzione delle applicazioni AIR. Ad esempio, potete creare un file di installazione EXE per la distribuzione di un'applicazione AIR in Windows, un file di installazione DMG per la distribuzione di un'applicazione AIR in Mac OS In AIR 2.5 and AIR 2.6, è possibile compilare un file di installazione DEB o RPM per la distribuzione di un'applicazione AIR in Linux.

Le applicazioni installate mediante un programma di installazione nativo sono dette applicazioni con profilo desktop esteso. Non è possibile utilizzare ADT per creare il pacchetto di un programma di installazione nativo per un'applicazione AIR se il file descrittore dell'applicazione non supporta il profilo desktop esteso. Potete assegnare una restrizione a questo profilo utilizzando l'elemento `supportedProfiles` nel file descrittore dell'applicazione. Vedete “[Profili dispositivo](#)” a pagina 254 e “[supportedProfiles](#)” a pagina 248.

Per creare una versione dell'applicazione AIR dotata di un programma di installazione nativo, potete procedere in due modi:

- Potete creare il programma di installazione nativo in base al file descrittore dell'applicazione e ad altri file di origine, ad esempio file SWF, file HTML e altre risorse.
- Potete creare il programma di installazione nativo in base a un file AIR o AIRI.

ADT deve essere eseguito nel sistema operativo associato al file del programma di installazione nativo che volete generare. Per creare un file EXE per Windows, ad esempio, eseguite ADT in Windows. Per creare un file DMG per Mac OS, eseguite ADT in Mac OS. Per creare un file DEB o RPM per Linux, eseguite ADT da AIR 2.6 SDK in Linux.

Quando create un programma di installazione nativo per distribuire un'applicazione AIR, l'applicazione acquisisce le seguenti funzionalità:

- Può avviare i processi nativi e interagire con questi processi utilizzando la classe `NativeProcess`. Per ulteriori informazioni, vedete una delle seguenti risorse:
 - Sviluppatori ActionScript: [Comunicazione con processi nativi in AIR](#)
 - Sviluppatori HTML: [Communicating with native processes in AIR](#)
- Può utilizzare estensioni native.
- È possibile utilizzare il metodo `File.openWithDefaultApplication()` per aprire qualsiasi tipo di file utilizzando l'applicazione predefinita specificata nel sistema operativo per l'apertura del file, indipendentemente dal tipo di file. Per le applicazioni che *non* vengono installate mediante un programma di installazione nativo sono previste alcune restrizioni. Per ulteriori informazioni, vedete la voce relativa a `File.openWithDefaultApplication()` nella guida di riferimento del linguaggio.

Tuttavia, se un file viene impacchettato come file di installazione nativo, l'applicazione perde alcuni dei vantaggi del formato AIR. Un singolo file non può più essere distribuito su tutti i computer desktop. La funzione di aggiornamento incorporata (nonché il framework di aggiornamento) non funziona.

Quando l'utente fa doppio clic sul file del programma di installazione nativo, viene avviata l'installazione dell'applicazione AIR. Se la versione richiesta di Adobe AIR non è già installata sul computer, il programma di installazione la scarica dalla rete e la installa. Se non è disponibile una connessione di rete per scaricare l'eventuale versione corretta di Adobe AIR, l'installazione non viene eseguita. Lo stesso succede se il sistema operativo non è supportato in Adobe AIR 2.

Nota: se desiderate rendere il file eseguibile nell'applicazione installata, accertatevi che sia eseguibile nel file system quando create un pacchetto dell'applicazione. (In Mac e Linux, potete utilizzare `chmod` per impostare il flag dell'eseguibile, se necessario.)

Creazione di un programma di installazione nativo a partire dai file di origine dell'applicazione

Per creare un programma di installazione nativo a partire dai file di origine dell'applicazione, utilizzate il comando `-package` con la seguente sintassi (su un'unica riga di comando):

```
adt -package AIR_SIGNING_OPTIONS
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    app_xml
    [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

Questa sintassi è simile a quella utilizzata per creare il pacchetto di un file AIR (senza programma di installazione nativo). Esistono tuttavia alcune differenze:

- Dovete aggiungere l'opzione `-target native` al comando. Se specificate `-target air`, ADT genera un file AIR anziché un file del programma di installazione nativo.
- Dovete specificare il file DMG o EXE di destinazione come `installer_file`.

- In Windows potete aggiungere un secondo set di opzioni di firma, indicato come [WINDOWS_INSTALLER_SIGNING_OPTIONS] nella sintassi. In Windows, oltre a firmare il file AIR, potete firmare il file di Windows Installer. Utilizzate lo stesso tipo di certificato e la stessa sintassi delle opzioni di firma che utilizzereste per firmare il file AIR (vedete “Opzioni di firma codice ADT” a pagina 185). Potete utilizzare lo stesso certificato per firmare il file AIR e il file del programma di installazione, oppure potete specificare certificati differenti. Quando un utente scarica dal Web un file di Windows Installer firmato, Windows identifica l'origine del file in base al certificato.

Per informazioni dettagliate sulle opzioni ADT diverse da `-target`, vedete “ADT (AIR Developer Tool)” a pagina 171.

Nell'esempio seguente viene creato un file DMG (file del programma di installazione nativo per Mac OS):

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    index.html resources
```

Nell'esempio seguente viene creato un file EXE (file del programma di installazione nativo per Windows):

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.exe
    application.xml
    index.html resources
```

Nell'esempio seguente viene creato e firmato un file EXE:

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    -storetype pkcs12
    -keystore myCert.pfx
    myApp.exe
    application.xml
    index.html resources
```

Creazione di un programma di installazione nativo per un'applicazione che usa estensioni native

Potete costruire un programma di installazione nativo a partire dai file di origine dell'applicazione e dai pacchetti delle estensioni native richieste dall'applicazione. Utilizzate il comando `-package` con la sintassi seguente su un'unica riga di comando:

```
adt -package AIR_SIGNING_OPTIONS
    -migrate MIGRATION_SIGNING_OPTIONS
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    app_xml
    -extdir extension-directory
    [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

Tale sintassi è la stessa utilizzata per la creazione di pacchetti di un programma di installazione nativo, con due opzioni aggiuntive. Utilizzate l'opzione `-extdir extension-directory` per specificare la directory contenente i file ANE (estensioni native) utilizzati dall'applicazione. Utilizzate il flag opzionale `-migrate` e i parametri `MIGRATION_SIGNING_OPTIONS` per firmare un aggiornamento di un'applicazione con una firma di migrazione, quando il certificato di firma del codice principale è diverso da quello utilizzato dalla versione precedente. Per maggiori informazioni, consultate “[Firma di una versione aggiornata di un'applicazione AIR](#)” a pagina 206.

Per dettagli sulle opzioni ADT, consultate “[ADT \(AIR Developer Tool\)](#)” a pagina 171.

Nell'esempio seguente viene creato un file DMG (un file di installazione nativo per Mac OS) per un'applicazione che usa estensioni native:

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    -extdir extensionsDir
    index.html resources
```

Creazione di un programma di installazione nativo a partire da un file AIR o AIRI

Potete utilizzare ADT per generare un file del programma di installazione nativo basato su un file AIR o AIRI. Per creare un programma di installazione nativo basato su un file AIR, utilizzate il comando ADT `-package` con la seguente sintassi (su un'unica riga di comando):

```
adt -package
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    air_file
```

Questa sintassi è simile a quella utilizzata per creare un programma di installazione nativo a partire dai file di origine dell'applicazione AIR. Esistono tuttavia alcune differenze:

- Come origine dovete specificare un file AIR anziché un file descrittore dell'applicazione e altri file di origine dell'applicazione AIR.
- Non specificate le opzioni di firma per il file AIR in quanto il file è già firmato.

Per creare un programma di installazione nativo basato su un file AIRI, utilizzate il comando ADT `-package` con la seguente sintassi (su un'unica riga di comando):

```
adt AIR_SIGNING_OPTIONS
    -package
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    airi_file
```

Questa sintassi è simile a quella utilizzata per creare un programma di installazione nativo basato su un file AIR. Esistono tuttavia alcune differenze:

- Come origine dovete specificare un file AIRI.
- Dovete specificare le opzioni di firma per l'applicazione AIR di destinazione.

Nell'esempio seguente viene creato un file DMG (file del programma di installazione nativo per Mac OS) basato su un file AIR:

```
adt -package -target native myApp.dmg myApp.air
```

Nell'esempio seguente viene creato un file EXE (file del programma di installazione nativo per Windows) basato su un file AIR:

```
adt -package -target native myApp.exe myApp.air
```

Nell'esempio seguente viene creato e firmato un file EXE basato su un file AIR:

```
adt -package -target native -storetype pkcs12 -keystore myCert.pfx myApp.exe myApp.air
```

Nell'esempio seguente viene creato un file DMG (file del programma di installazione nativo per Mac OS) basato su un file AIR:

```
adt -storetype pkcs12 -keystore myCert.pfx -package -target native myApp.dmg myApp.airi
```

Nell'esempio seguente viene creato un file EXE (file del programma di installazione nativo per Windows) basato su un file AIR:

```
adt -storetype pkcs12 -keystore myCert.pfx -package -target native myApp.exe myApp.airi
```

Nell'esempio seguente viene creato un file EXE (basato su un file AIRI) al quale viene applicata sia una firma AIR che una firma nativa di Windows:

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native -storetype pkcs12 -keystore myCert.pfx myApp.exe myApp.airi
```

Creazione di un pacchetto runtime autonomo per computer desktop

Un pacchetto runtime autonomo è un pacchetto che include il codice della vostra applicazione insieme a una versione dedicata del runtime. Un'applicazione compilata in questo modo impiega il runtime impacchettato, anziché il runtime condiviso installato altrove sul computer dell'utente.

Il pacchetto prodotto è una cartella autonoma di file dell'applicazione su Windows oppure un pacchetto .app in Mac OS. Per produrre un pacchetto per un sistema operativo di destinazione dovete eseguire tale sistema operativo. (Per eseguire più sistemi operativi su un singolo computer, è possibile usare una macchina virtuale, quale VMWare.)

L'applicazione può essere eseguita da tale cartella o pacchetto senza necessità di installazione.

Vantaggi

- Produce un'applicazione "tutto in uno"
- Nessun accesso a Internet richiesto per l'installazione
- L'applicazione è isolata dagli aggiornamenti del runtime
- Le aziende possono certificare la specifica combinazione di applicazione e runtime
- Supporto del modello di implementazione software tradizionale
- Nessuna necessità di redistribuzione separata del runtime
- Possibilità di usare l'API NativeProcess
- Possibilità di utilizzare estensioni native
- Possibilità di utilizzare la funzione `File.OpenWithDefaultApplication()` senza limitazioni
- Eseguitabile da USB o disco ottico senza necessità di installazione

Svantaggi

- Patch di sicurezza critici non automaticamente disponibili ogni volta che Adobe pubblica un patch di sicurezza
- Non consente l'impiego del formato file .air
- Se necessario, dovete creare un vostro programma di installazione
- API e framework di aggiornamento AIR non supportati
- L'API AIR integrata nel browser per l'installazione e il lancio di applicazioni AIR da una pagina web non è supportata
- In Windows, la registrazione file deve essere gestita dal vostro programma di installazione
- Maggiore spazio su disco occupato dall'applicazione

Creazione di un pacchetto runtime autonomo in Windows

Per creare un pacchetto runtime autonomo per Windows, dovete compilare l'applicazione mentre il sistema operativo Windows è in esecuzione. Create il pacchetto dell'applicazione utilizzando il target ADT *bundle*.

```
adt -package
    -keystore ..\cert.p12 -storetype pkcs12
    -target bundle
    myApp
    myApp-app.xml
    myApp.swf icons resources
```

Questo comando crea il pacchetto in una directory denominata myApp. Tale directory contiene i file sia dell'applicazione che del runtime. Potete eseguire il programma direttamente dalla cartella. Tuttavia, per creare una voce di menu del programma, registrare i tipi di file o i gestori di schemi URI, dovete creare un programma di installazione che configuri le voci di registro richieste. L'SDK di AIR non include strumenti per creare tali programmi di installazione, tuttavia sono disponibili varie opzioni di terze parti, sia a pagamento che gratuite, quali i toolkit per programmi di installazione open-source.

Potete firmare l'eseguibile nativo in Windows, specificando un secondo set di opzioni di firma, dopo la voce `-target bundle` della riga di comando. Queste opzioni di firma identificano la chiave privata e il certificato associato da utilizzare quando si applica la firma nativa di Windows. (In genere, è possibile utilizzare un certificato di firma codice di AIR.) Solo l'eseguibile principale viene firmato. Eventuali eseguibili aggiuntivi impacchettati con l'applicazione non vengono firmati durante questa procedura.

Associazione dei tipi di file

Per associare alla vostra applicazione tipi di file pubblici o personalizzati su Windows, il programma di installazione che avete creato deve configurare le voci di registro appropriate. I tipi di file devono anche essere elencati nell'elemento `fileTypes` del file descrittore dell'applicazione.

Per maggiori informazioni sui tipi di file di Windows, consultate [MSDN Library: File Types and File Associations \(Libreria MSDN: tipi e associazioni di file\)](#)

Registrazione del gestore URI

Per fare in modo che l'applicazione possa gestire il lancio di un URL mediante un determinato schema URI, è necessario che il vostro programma di installazione configuri le voci di registro necessarie.

Per maggiori informazioni sulla registrazione di un'applicazione per la gestione di uno schema URI, consultate [MSDN Library: Registering an Application to a URL Protocol \(Libreria MSDN: registrazione di un'applicazione in un protocollo URL\)](#)

Creazione di un pacchetto runtime autonomo in Mac OS

Per creare un pacchetto runtime autonomo per Mac OS, dovete compilare l'applicazione mentre il sistema operativo Mac OS è in esecuzione. Create il pacchetto dell'applicazione utilizzando il target ADT *bundle*.

```
adt -package
    -keystore ../cert.p12 -storetype pkcs12
    -target bundle
    myApp.app
    myApp-app.xml
    myApp.swf icons resources
```

Questo comando crea il pacchetto dell'applicazione denominato myApp.app. Tale pacchetto contiene i file sia dell'applicazione che del runtime. Potete eseguire l'applicazione facendo doppio clic sull'icona myApp.app e installarla trascinandola in una posizione idonea, quale la cartella Applicazioni. Tuttavia, per registrare i tipi di file o i gestori degli schemi URI, dovete modificare il file dell'elenco delle proprietà all'interno del pacchetto dell'applicazione.

Per la distribuzione, potete creare un file di immagine del disco (.dmg). L'SDK di Adobe AIR non fornisce strumenti per la creazione di file dmg per un pacchetto runtime autonomo.

Associazione dei tipi di file

Per associare alla vostra applicazione tipi di file pubblici o personalizzati in Mac OS X, dovete modificare il file info.plist nel pacchetto per configurare la proprietà CFBundleDocumentTypes. Consultate [Mac OS X Developer Library: Information Property List Key Reference, CFBundleURLTypes](#).

Registrazione del gestore URI

Per fare in modo che l'applicazione possa gestire il lancio di un URL mediante un determinato schema URI, dovete modificare il file info.plist del pacchetto per configurare la proprietà CFBundleURLTypes. Consultate [Mac OS X Developer Library: Information Property List Key Reference, CFBundleDocumentTypes](#).

Distribuzione di pacchetti AIR per computer desktop

Le applicazioni AIR possono essere distribuite sotto forma di pacchetto AIR, che contiene il codice dell'applicazione e tutte le risorse. Potete distribuire questo pacchetto attraverso i canali tipici, quali download, e-mail o supporti fisici quali CD-ROM. Gli utenti possono installare l'applicazione facendo doppio clic sul file AIR. Potete usare l'API AIR interna al browser (una libreria ActionScript basata su Web) per consentire agli utenti di installare l'applicazione AIR (e Adobe® AIR®, se necessario) facendo clic su un singolo collegamento su una pagina Web.

Le applicazioni AIR possono anche essere impacchettate e distribuite come programmi di installazione nativi (ovvero come file EXE in Windows, DMG in Mac OS e DEB o RPM in Linux). I pacchetti di installazione nativi possono essere distribuiti e installati in base alle convenzioni specifiche della piattaforma di destinazione. Quando distribuite un'applicazione come pacchetto nativo, tuttavia, perdete alcuni dei vantaggi del formato di file AIR. In particolare, un singolo file di installazione non può più essere utilizzato sulla maggior parte delle piattaforme, il framework di aggiornamento AIR non è più disponibile e l'API interna al browser non può più essere utilizzata.

Installazione ed esecuzione di un'applicazione AIR sul desktop

È possibile inviare semplicemente il file AIR al destinatario. Ad esempio, è possibile inviare il file AIR come allegato di posta elettronica o come collegamento in una pagina Web.

Una volta scaricata l'applicazione AIR, seguite le istruzioni riportate di seguito per installarla:

1 Fate doppio clic sul file AIR.

Adobe AIR deve essere già installato nel computer.

2 Nella finestra Installazione, lasciate selezionate le impostazioni predefinite, quindi fate clic su Continua.

In Windows, AIR effettua automaticamente le seguenti operazioni:

- Installa l'applicazione nella directory Programmi
- Crea un collegamento al desktop per l'applicazione.
- Crea un collegamento al menu Start.
- Aggiunge una voce per l'applicazione in Installazione applicazioni di Pannello di controllo

In Mac OS, per impostazione predefinita l'applicazione viene aggiunta nella directory Applicazioni.

Se l'applicazione è già installata, il programma di installazione consente all'utente di scegliere tra aprire la versione esistente dell'applicazione o effettuare l'aggiornamento alla versione nel file AIR scaricato. L'applicazione viene identificata mediante l'ID applicazione e l'ID editore nel file AIR.

3 Al termine dell'installazione, scegliete Fine.

In Mac OS, per installare una versione aggiornata di un'applicazione sono necessari privilegi di sistema adeguati per effettuare l'installazione nella directory dell'applicazione. In Windows e Linux, sono richiesti privilegi di amministratore.

Un'applicazione è in grado di installare anche una nuova versione tramite ActionScript o JavaScript. Per ulteriori informazioni, vedete [“Aggiornamento di applicazioni AIR”](#) a pagina 267.

Dopo che l'applicazione AIR è stata installata, è sufficiente fare doppio clic sull'icona dell'applicazione per eseguirla, come per qualsiasi altra applicazione desktop.

- In Windows, fate doppio clic sull'icona dell'applicazione (installata sul desktop o in una cartella) oppure selezionate l'applicazione dal menu Start.
- In Linux, fate doppio clic sull'icona dell'applicazione (installata sul desktop o in una cartella) oppure selezionate l'applicazione dal menu delle applicazioni.
- In Mac OS, fate doppio clic sull'applicazione nella cartella in cui è stata installata. La directory di installazione predefinita è /Applications.

Nota: solo le applicazioni AIR sviluppate per AIR 2.6 o versioni precedenti possono essere installate in Linux.

La funzionalità *installazione invisibile* di AIR consente di installare un'applicazione AIR facendo clic su un collegamento in una pagina Web. Le funzionalità di *chiamata browser* consentono a un utente di eseguire un'applicazione AIR installata facendo clic su un collegamento in una pagina Web. Tali funzionalità vengono descritte nelle sezioni che seguono.

Installazione ed esecuzione di applicazioni AIR desktop da una pagina Web

L'API AIR interna al browser permette di installare ed eseguire un'applicazione AIR da una pagina Web. Questa API viene fornita in una libreria SWF, *air.swf*, ospitata da Adobe. AIR SDK include un'applicazione "badge" di esempio che utilizza questa libreria per installare, aggiornare o avviare un'applicazione AIR (e il runtime, se necessario). Potete modificare l'applicazione badge di esempio fornita oppure creare una vostra applicazione Web badge che utilizza direttamente la libreria *air.swf* online.

Qualsiasi applicazione AIR può essere installata tramite un'applicazione badge su pagina Web. Tuttavia, solo le applicazioni che includono l'elemento `<allowBrowserInvocation>true</allowBrowserInvocation>` nei file descrittivi dell'applicazione possono essere avviate tramite un badge Web.

Altri argomenti presenti nell' Aiuto

“[API interna al browser AIR.SWF](#)” a pagina 258

Distribuzione aziendale su computer desktop

Gli amministratori IT possono installare il runtime di Adobe AIR e le applicazioni AIR in modo invisibile all'utente utilizzando gli strumenti standard di distribuzione desktop. Gli amministratori IT possono effettuare le seguenti operazioni:

- Installare in modo invisibile all'utente il runtime di Adobe utilizzando strumenti quali Microsoft SMS, IBM Tivoli o qualsiasi altro strumento di distribuzione che consente l'utilizzo di bootstrap per questo tipo di installazioni.
- Installare in modo invisibile all'utente l'applicazione AIR utilizzando gli stessi strumenti impiegati per la distribuzione del runtime.

Per ulteriori informazioni, vedete il manuale [Adobe AIR Administrator's Guide \(Guida per l'amministratore di Adobe AIR\)](#) (http://www.adobe.com/go/learn_air_admin_guide_it).

Registri di installazione su computer desktop

I registri di installazione vengono creati quando si installa il runtime AIR o un'applicazione AIR. Potete consultare i file di registro per individuare la causa di eventuali problemi di installazione o aggiornamento.

I file di registro vengono creati nei seguenti percorsi:

- Mac: registro di sistema standard (`/private/var/log/system.log`)
Potete visualizzare il registro di sistema Mac aprendo l'applicazione Console, che solitamente si trova nella cartella Utilità.
- Windows XP: `C:\Documents and Settings\\Impostazioni locali\Dati applicazioni\Adobe\AIR\logs\Install.log`
- Windows Vista, Windows 7: `C:\Utenti\\AppData\Local\Adobe\AIR\logs\Install.log`
- Linux: `/home/<nome utente>/.appdata/Adobe/AIR/Logs/Install.log`

Nota: questi file di registro non venivano creati nelle versioni di AIR precedenti ad AIR 2.

Capitolo 7: Sviluppo di applicazioni AIR per dispositivi mobili

Le applicazioni AIR per dispositivi mobili vengono distribuite come applicazioni native. In altri termini, usano il formato di applicazione del dispositivo, non il formato di file AIR. Attualmente AIR supporta i pacchetti Android APK e i pacchetti iOS IPA. Una volta creata la versione release del pacchetto dell'applicazione, potete distribuirla tramite il meccanismo standard della piattaforma di destinazione. Per Android, solitamente si tratta di Android Market; per iOS, di Apple App Store.

Potete usare [AIR SDK](#) e Flash Professional, Flash Builder o un altro strumento di sviluppo ActionScript per realizzare applicazioni AIR per dispositivi mobili. Le applicazioni AIR per dispositivi mobili basate su HTML attualmente non sono supportate.

***Nota:** il dispositivo BlackBerry Playbook di Research In Motion (RIM) dispone di un proprio kit SDK per lo sviluppo AIR. Per informazioni sullo sviluppo per il prodotto Playbook, vedete [RIM: BlackBerry Tablet OS Development](#).*

***Nota:** questo documento spiega come sviluppare applicazioni iOS utilizzando il kit SDK AIR 2.6 o successivo. Le applicazioni create con AIR 2.6+ possono essere installate su dispositivi iPhone 3Gs, iPhone 4 e iPad che eseguono iOS 4 o una versione successiva. Per sviluppare applicazioni AIR per versioni precedenti di iOS, dovete usare AIR 3 Packager for iPhone, come descritto in [Creazione di applicazioni iPhone](#).*

Per ulteriori informazioni sulle procedure consigliate per la privacy, vedete la [guida sulla privacy di Adobe AIR SDK](#).

Per consultare i requisiti di sistema completi per l'esecuzione di applicazioni AIR, vedete [Adobe AIR: requisiti di sistema](#).

Impostazione dell'ambiente di sviluppo

Le piattaforme mobili non prevedono requisiti di installazione aggiuntivi oltre a quelli della normale configurazione dell'ambiente di sviluppo AIR, Flex e Flash. (Per ulteriori informazioni sull'impostazione di base dell'ambiente di sviluppo AIR, vedete “[Strumenti della piattaforma Adobe Flash per lo sviluppo AIR](#)” a pagina 18.)

Configurazione per Android

Non sono richieste impostazioni di configurazione speciali per Android in AIR 2.6+. Il tool ADB di Android è incluso in AIR SDK (nella cartella lib/android/bin). AIR SDK usa il tool ADB per installare, disinstallare ed eseguire i pacchetti delle applicazioni sul dispositivo. Potete utilizzare ADB anche per visualizzare i registri di sistema. Per creare ed eseguire un emulatore Android dovete scaricare il kit SDK Android separato.

Se la vostra applicazione aggiunge elementi all'elemento `<manifestAdditions>` del descrittore dell'applicazione che non sono riconosciuti come validi dalla versione corrente di AIR, dovete installare una versione più recente di Android SDK. Impostate la variabile di ambiente AIR_ANDROID_SDK_HOME o il parametro della riga di comando `-platformsdk` sul percorso di file del kit SDK. Il tool di compilazione pacchetti di AIR, ADT, usa questo SDK per convalidare le voci dell'elemento `<manifestAdditions>`.

In AIR 2.5, è necessario scaricare una copia distinta di Android SDK da Google. Potete impostare la variabile d'ambiente AIR_ANDROID_SDK_HOME in modo che faccia riferimento alla cartella di Android SDK. Se non effettuate questa impostazione, dovete specificare il percorso di Android SDK nell'argomento `-platformsdk` sulla riga di comando ADT.

Altri argomenti presenti nell' Aiuto

“Variabili d'ambiente ADT” a pagina 195

“Variabili d'ambiente dei percorsi” a pagina 316

Configurazione per iOS

Per installare e provare un'applicazione iOS su un dispositivo e distribuirla, dovete aderire al programma Apple iOS Developer, un servizio gratuito. Una volta effettuata l'iscrizione al programma iOS Developer, potete accedere al sito iOS Provisioning Portal, dove è possibile procurarsi le risorse e i file indicati di seguito, necessarie per installare un'applicazione su un dispositivo a scopo di test e successiva distribuzione. Tali risorse e file sono:

- Certificati di sviluppo e di distribuzione
- ID applicazione
- File di provisioning di sviluppo e di distribuzione

Considerazioni sulla progettazione di applicazioni per dispositivi mobili

Il contesto operativo e le caratteristiche fisiche dei dispositivi mobili impongono un'estrema cura nelle fasi di scrittura del codice e progettazione. Ad esempio, è fondamentale ottimizzare il codice per garantire la massima velocità di esecuzione. L'ottimizzazione del codice può arrivare fino a un certo punto, ovviamente; una progettazione intelligente che si adatta ai limiti del dispositivo può anche aiutare a evitare di sottoporre il sistema di rendering a un carico di lavoro eccessivo.

Codice

Velocizzare l'esecuzione del codice è sempre positivo, ovviamente, e la più lenta velocità del processore della maggior parte dei dispositivi mobili rende più gratificante il tempo e l'impegno dedicati alla scrittura di un codice "snello". Inoltre, i dispositivi mobili funzionano quasi sempre a batteria. Ottenere lo stesso risultato con meno lavoro permette di consumare meno carica della batteria.

Progettazione

Fattori come le dimensioni ridotte dello schermo, l'interazione tramite touch screen e anche l'ambiente in costante cambiamento dell'utente di un dispositivo mobile devono essere presi in debita considerazione quando si progetta la user experience dell'applicazione.

Codice e progettazione in combinazione

Se l'applicazione usa l'animazione, l'ottimizzazione del rendering è molto importante. Tuttavia, l'ottimizzazione del codice da sola spesso non basta. Dovete progettare gli aspetti visuali dell'applicazione in maniera tale che il codice possa eseguirne il rendering in modo efficiente.

Importanti tecniche di ottimizzazione sono trattate nella guida [Ottimizzazione delle prestazioni per la piattaforma Adobe Flash](#). Le tecniche descritte nella guida valgono per i contenuti Flash e AIR di qualsiasi tipo, ma sono essenziali per lo sviluppo di applicazioni che funzionino bene sui dispositivi mobili.

- [Paul Trani: Tips and Tricks for Mobile Flash Development](#)
- [roguish: GPU Test App AIR for Mobile](#)
- [Jonathan Campos: Optimization Techniques for AIR for Android apps](#)
- [Charles Schulze: AIR 2.6 Game Development: iOS included](#)

Ciclo di vita dell'applicazione

Quando l'applicazione perde lo stato di attivazione in favore di un'altra applicazione, AIR riduce la frequenza dei fotogrammi a 4 fotogrammi al secondo e cessa di eseguire il rendering della grafica. Al di sotto di questa frequenza, lo streaming di rete e le connessioni socket tendono a interrompersi. Se l'applicazione non fa uso di tali connessioni, potete ridurre la frequenza dei fotogrammi anche a un valore inferiore.

Quando è appropriato, interrompete la riproduzione audio e rimuovete i listener dei sensori di localizzazione geografica (geolocation) e accelerometro (accelerometer). L'oggetto AIR NativeApplication invia gli eventi di attivazione (activate) e disattivazione (deactivate). Usate questi eventi per gestire la transizione tra lo stato attivo e lo stato di background.

La maggior parte dei sistemi operativi per dispositivi mobili chiude le applicazioni in background senza avvisare l'utente. Se lo stato dell'applicazione viene salvato frequentemente, dovrebbe essere possibile ripristinare l'applicazione a uno stato di funzionalità sia che venga riportata allo stato di attività dal background o venga avviata di nuovo.

Densità di informazione

Le dimensioni fisiche dello schermo dei dispositivi mobili sono inferiori a quelle del desktop, sebbene la densità dei pixel (pixel per pollice) sia maggiore. La stessa dimensione carattere produce lettere fisicamente più piccole sullo schermo di un dispositivo mobile rispetto a un computer desktop. Spesso occorre adottare un carattere più grande per garantire la leggibilità. In generale, la dimensione di 14 punti è la più piccola che risulti facilmente leggibile.

I dispositivi mobili sono spesso utilizzati durante gli spostamenti e in condizioni di scarsa illuminazione. Considerate la quantità di informazioni che è possibile visualizzare realisticamente sullo schermo in maniera leggibile. Potrebbe essere inferiore a quella visualizzata su uno schermo desktop avente le stesse dimensioni in pixel.

Valutate anche che quando un utente tocca lo schermo, il dito e la mano occludono alla vista parte del display. Inserite gli elementi interattivi ai lati e in fondo allo schermo quando l'utente deve interagire con essi per un intervallo di tempo più lungo rispetto a un singolo tocco istantaneo.

Immissione del testo

Molti dispositivi usano una tastiera virtuale per l'immissione del testo. Le tastiere virtuali oscurano parte dello schermo e spesso non sono facilissime da usare. Evitate di fare un uso eccessivo degli eventi da tastiera (tranne che per i tasti virtuali)

Valutate la possibilità di implementare delle alternative all'utilizzo dei campi di testo di input. Ad esempio, per fare in modo che l'utente immetta un valore numerico, non è necessario un campo di testo. Potete fornire due pulsanti per aumentare o diminuire il valore.

Tasti virtuali

I dispositivi mobili includono un numero variabile di tasti virtuali. I tasti virtuali sono pulsanti programmabili con svariate funzioni. Seguite le convenzioni della piattaforma specifica per adottare questi tasti nella vostra applicazione.

Modifica dell'orientamento dello schermo

Il contenuto destinato ai dispositivi mobili può essere visualizzato con orientamento verticale o orizzontale. Valutate il modo in cui l'applicazione reagisce alle variazioni dell'orientamento dello schermo. Per maggiori informazioni, consultate [Orientamento dello stage](#).

Oscuramento dello schermo

AIR non disabilita automaticamente l'oscuramento dello schermo durante la riproduzione di un video. Potete usare la proprietà `systemIdleMode` dell'oggetto AIR NativeApplication per controllare l'attivazione o meno della modalità di risparmio energetico del dispositivo. (Su alcune piattaforme, è necessario richiedere le autorizzazioni appropriate per poter utilizzare questa funzione.)

Chiamate telefoniche in arrivo

Il runtime AIR disattiva automaticamente l'audio quando l'utente esegue o riceve una chiamata telefonica. Su Android, dovete impostare l'autorizzazione Android `READ_PHONE_STATE` nel descrittore dell'applicazione se l'applicazione riproduce l'audio mentre è in background. In caso contrario, Android impedisce al runtime di rilevare le chiamate in arrivo e di disattivare automaticamente l'audio. Vedete "[Autorizzazioni Android](#)" a pagina 80.

Destinazioni di tocco

Valutate le dimensioni delle destinazioni di tocco durante la progettazione di pulsanti e altri elementi dell'interfaccia utente che vengono toccati dall'utente. Si consiglia di rendere questi elementi sufficientemente larghi da poter essere comodamente attivati con un dito su uno schermo sensibile. Inoltre, accertarsi che lo spazio tra le destinazioni sia sufficiente. L'area "target" del tocco deve essere di circa 44 x 57 pixel su ciascun lato, nel caso di un display telefonico tipico con valore dpi elevato.

Dimensioni del pacchetto di installazione dell'applicazione

I dispositivi mobili solitamente hanno una quantità di memoria molto inferiore ai computer desktop per l'installazione delle applicazioni e l'archiviazione dei dati. Riducete le dimensioni del pacchetto rimuovendo le risorse e le librerie inutilizzate.

Su Android, il pacchetto dell'applicazione non viene estratto in un file separato quando un'applicazione viene installata. Al contrario, le risorse vengono decomprese in un'area di memoria temporanea nel momento in cui vengono utilizzate. Per ridurre al minimo lo spazio occupato in memoria da queste risorse decomprese, chiudete i flussi dati di file e URL quando le risorse sono state completamente caricate.

Accesso al file system

I diversi sistemi operativi per dispositivi mobili impongono restrizioni differenti sul file system e tali restrizioni tendono anche a differenziarsi da quelle previste nei sistemi operativi desktop. La posizione corretta per collocare e salvare file e dati può, di conseguenza, variare da una piattaforma all'altra.

Una delle conseguenze dell'uso di file system differenti consiste nel fatto che le scorciatoie alla directory più comune fornite dalla classe File di AIR non sono sempre disponibili. La tabella seguente mostra le scorciatoie che possono essere utilizzate in Android e iOS:

	Android	iOS
File.applicationDirectory	Sola lettura tramite URL (non percorso nativo)	Sola lettura
File.applicationStorageDirectory	Disponibile	Disponibile
File.cacheDirectory	Disponibile	Disponibile
File.desktopDirectory	Root scheda SD	Non disponibile
File.documentsDirectory	Root scheda SD	Disponibile
File.userDirectory	Root scheda SD	Non disponibile
File.createTempDirectory()	Disponibile	Disponibile
File.createTempFile()	Disponibile	Disponibile

Le linee guida Apple per le applicazioni iOS forniscono regole specifiche su quali posizioni di archiviazione utilizzare per i file in diversi casi. Ad esempio, una linea guida indica che solo i file che contengono i dati immessi dall'utente o i dati che non possono essere rigenerati o riscaricati dovrebbero essere archiviati in una directory specifica per il backup remoto. Per informazioni su come rispettare le linee guida Apple per il backup e la memorizzazione nella cache del file, consultate [Controllo del backup dei file e memorizzazione nella cache](#).

Componenti dell'interfaccia utente

Adobe ha sviluppato una versione del framework Flex ottimizzata per i dispositivi mobili. Per maggiori informazioni, consultate l'articolo [Developing Mobile Applications with Flex and Flash Builder](#).

Sono anche disponibili progetti di componenti per applicazioni mobili proposti dalla comunità degli utenti. Ad esempio:

- Josh Tynjala: [Feathers UI controls for Starling](#)
- Derrick Grigg: [skinnable version of Minimal Comps](#)
- Todd Anderson: [as3flobile components](#)

Rendering con grafica accelerata Stage3D

A partire da AIR 3.2, AIR per dispositivi mobili supporta il rendering con grafica accelerata Stage 3D. Le API ActionScript [Stage3D](#) sono una serie di API di basso livello con accelerazione grafica che abilitano funzionalità 2D e 3D avanzate. Queste API di basso livello danno agli sviluppatori la flessibilità necessaria per sfruttare l'accelerazione hardware tramite GPU al fine di ottenere sensibili miglioramenti delle prestazioni. Potete anche usare motori di gaming che supportano le API ActionScript Stage3D.

Per ulteriori informazioni, vedete [Gaming engines, 3D, and Stage 3D](#) (Motori di gaming, 3D e Stage3D).

Smoothing video

Per migliorare le prestazioni, l'opzione di smoothing video è disattivata su AIR.

Funzioni native

AIR 3.0+

Molte piattaforme mobili forniscono funzioni che non sono ancora accessibili tramite l'API standard di AIR. A partire da AIR 3, potete estendere AIR con le vostre librerie di codice. Queste librerie di estensione native possono accedere alle funzioni disponibili nel sistema operativo così come alle funzioni specifiche di un determinato dispositivo. Le estensioni native possono essere scritte in C per iOS e in Java o C per Android. Per informazioni sullo sviluppo delle estensioni native, vedete [Introducing native extensions for Adobe AIR](#) (Introduzione alle estensioni native per Adobe AIR).

Flusso di lavoro per la creazione di applicazioni AIR per dispositivi mobili

Il flusso di lavoro per la creazione di un'applicazione AIR per dispositivi mobili (o di altro tipo) è, in generale, molto simile a quello per la creazione di applicazioni desktop. Le differenze principali rispetto al flusso di lavoro principale si presentano quando è il momento di creare il pacchetto, eseguire il debug e installare l'applicazione. Ad esempio, le applicazioni AIR for Android utilizzano il formato di pacchetto nativo Android APK anziché il formato di pacchetto AIR. Di conseguenza, usano anche i meccanismi di installazione e aggiornamento standard di Android.

AIR for Android

I seguenti passaggi sono tipici del processo di sviluppo di un'applicazione AIR for Android:

- Scrivete il codice ActionScript o MXML.
- Create il file descrittore di un'applicazione AIR (utilizzando lo spazio dei nomi 2.5 o successivo)
- Compilate l'applicazione.
- Create il pacchetto dell'applicazione nel formato Android (.apk).
- Installate il runtime di AIR sul dispositivo o sull'emulatore Android (se utilizzate un runtime esterno; il runtime autonomo è quello predefinito in AIR 3.7 e versioni successive).
- Installate l'applicazione nel dispositivo (o nell'emulatore Android).
- Avviate l'applicazione sul dispositivo.

Per effettuare questi passaggi potete usare Adobe Flash Builder, Adobe Flash Professional CS5 o i tool della riga di comando.

Una volta completata l'applicazione AIR e creato il pacchetto APK, potete inviare quest'ultimo ad Android Market o distribuirlo tramite altri canali.

AIR for iOS

I seguenti passaggi sono tipici del processo di sviluppo di un'applicazione AIR for iOS:

- Installate iTunes.
- Generate i file sviluppatore e gli ID necessari nel sito Apple iOS Provisioning Portal. Questi elementi sono:
 - Certificato per sviluppatori
 - App ID (ID applicazione)
 - Profilo di provisioning

Quando create il profilo di provisioning, dovete elencare gli ID di tutti i dispositivi di prova sui quali intendete installare l'applicazione.

- Convertite il certificato di sviluppo e la chiave privata in un file di archivio di chiavi P12.
- Scrivete il codice ActionScript o MXML dell'applicazione.
- Compilate l'applicazione con un compilatore ActionScript o MXML.
- Create un disegno di icona e un disegno di schermata iniziale per l'applicazione.
- Create il descrittore dell'applicazione (utilizzando lo spazio dei nomi 2.6 o successivo).
- Create il pacchetto del file IPA utilizzando ADT.
- Utilizzate iTunes per inserire il profilo di provisioning nel dispositivo di prova.
- Installate e provate l'applicazione sul dispositivo iOS. Per installare il file IPA, potete utilizzare iTunes o ADT via USB (supportato in AIR 3.4 e versioni successive).

Una volta completata l'applicazione AIR, potete crearne un nuovo pacchetto utilizzando un certificato di distribuzione e il profilo di provisioning. A questo punto potete inviarla a Apple App Store.

Impostazione delle proprietà delle applicazioni per dispositivi mobili

Come per le altre applicazioni AIR, dovete impostare le proprietà di base dell'applicazione nel file descrittore dell'applicazione. Le applicazioni per dispositivi mobili ignorano alcune delle proprietà specifiche delle applicazioni desktop, quali le dimensioni della finestra e la trasparenza. Inoltre, le applicazioni per dispositivi mobili possono utilizzare proprietà specifiche della piattaforma di destinazione. Ad esempio, potete includere un elemento `android` per le applicazioni Android e un elemento `iPhone` per le applicazioni iOS.

Impostazioni comuni

Varie impostazioni del descrittore dell'applicazione sono importanti per tutte le applicazioni per dispositivi mobili.

Versione del runtime AIR necessaria

Specificate la versione del runtime AIR necessaria per l'applicazione utilizzando lo spazio dei nomi del file descrittore dell'applicazione.

Lo spazio dei nomi, assegnato nell'elemento `application`, determina in larga parte quali funzioni può utilizzare la vostra applicazione. Ad esempio, se l'applicazione usa lo spazio dei nomi AIR 2.7 e l'utente ha installato una versione successiva, l'applicazione vede comunque il comportamento AIR 2.7 (anche se è cambiato nella versione successiva). Solo quando cambiate lo spazio dei nomi e pubblicate un aggiornamento l'applicazione avrà accesso al nuovo comportamento e alle nuove funzioni. Gli aggiornamenti correttivi relativi alla sicurezza rappresentano un'importante eccezione a questa regola.

Nei dispositivi che utilizzano un runtime separato dall'applicazione, come Android in AIR 3.6 e versioni precedenti, all'utente viene richiesto di installare o aggiornare AIR se non dispone della versione richiesta. Nei dispositivi che incorporano il runtime, come iPhone, questa situazione non si verifica (poiché la versione richiesta è già inclusa nel pacchetto dell'applicazione).

Nota: (AIR 3.7 e versioni successive) Per impostazione predefinita, ADT include il runtime con le applicazioni Android.

Specificate lo spazio dei nomi utilizzando l'attributo `xmlns` dell'elemento root `application`. I seguenti spazi dei nomi devono essere utilizzati per le applicazioni per dispositivi mobili (a seconda della piattaforma mobile di destinazione):

iOS 4+ and iPhone 3Gs+ or Android:

```
<application xmlns="http://ns.adobe.com/air/application/2.7">
  iOS only:
  <application xmlns="http://ns.adobe.com/air/application/2.0">
```

Nota: il supporto per i dispositivi iOS 3 è fornito dal kit SDK Packager for iPhone, basato sul kit SDK AIR 2.0. Per informazioni sullo sviluppo di applicazioni AIR for iOS 3, vedete [Creazione di applicazioni iPhone](#). AIR 2.6 SDK (e versioni successive) supporta iOS 4 e versioni successive su dispositivi iPhone 3Gs, iPhone 4 e iPad.

Altri argomenti presenti nell' Aiuto

“[application](#)” a pagina 217

Identità dell'applicazione

Diverse impostazioni devono essere univoche per ciascuna applicazione che pubblicate. Tali impostazioni includono l'ID, il nome e il nome del file.

ID applicazione Android

In Android, l'ID viene convertito nel nome del pacchetto Android antepoendo il prefisso "air.". all'ID AIR. Quindi, se ad esempio l'ID AIR è `com.example.MyApp`, il nome del pacchetto Android è `air.com.example.MyApp`.

```
<id>com.example.MyApp</id>
      <name>My Application</name>
      <filename>MyApplication</filename>
```

Inoltre, se l'ID non è un nome di pacchetto valido per il sistema operativo Android, viene convertito in un nome valido. I trattini vengono convertiti in trattini di sottolineatura e le cifre iniziali di qualsiasi componente dell'ID vengono precedute da una "A" maiuscola. Ad esempio, l'ID `3-goats.1-boat` viene trasformato nel nome di pacchetto `air.A3_goats.A1_boat`.

Nota: il prefisso aggiunto all'ID applicazione può essere utilizzato per identificare le applicazioni AIR in Android Market. Se non volete che l'applicazione possa essere identificata come applicazione AIR in virtù del prefisso, dovete "spacchettare" il file APK, cambiare l'ID applicazione e re-impacchettarlo come descritto in [Opt-out of AIR application analytics for Android](#).

ID applicazione iOS

Impostate l'ID applicazione AIR in modo che corrisponda all'ID applicazione creato nel sito Apple iOS Provisioning Portal.

Gli ID applicazione iOS contengono un identificatore chiamato "bundle seed ID" (ID inizializzazione pacchetto) seguito da un altro identificatore denominato "bundle ID" (identificatore pacchetto). L'ID inizializzazione pacchetto è una stringa di caratteri, ad esempio `5RM86Z4DJM`, assegnata da Apple all'ID applicazione. L'identificatore pacchetto contiene un nome in stile nome di dominio inverso a vostra scelta. L'identificatore pacchetto può terminare con un carattere asterisco (*), che indica un ID applicazione jolly. Se l'ID pacchetto termina con il carattere jolly, potete sostituire tale carattere con qualsiasi stringa valida.

Ad esempio:

- Se l'ID applicazione Apple è `5RM86Z4DJM.com.example.helloWorld`, dovete usare `com.example.helloWorld` nel descrittore dell'applicazione.

- Se l'ID applicazione Apple è `96LPVWEASL.com.example.*` (notate il carattere jolly alla fine), potreste usare `com.example.helloWorld` o `com.example.anotherApp`, oppure qualsiasi altro ID che inizi con `com.example`.
- Infine, se l'ID applicazione Apple è composto semplicemente dal bundle seed ID e da un carattere jolly, come in `38JE93KJL.*`, potete usare qualsiasi ID applicazione in AIR.

Quando specificate l'ID applicazione, non includete la parte corrispondente all'ID inizializzazione pacchetto (bundle seed ID).

Altri argomenti presenti nell' Aiuto

[“id”](#) a pagina 234

[“filename”](#) a pagina 229

[“name”](#) a pagina 242

Versione dell'applicazione

In AIR 2.5 e versioni successive, specificate la versione dell'applicazione nell'elemento `versionNumber`. L'uso dell'elemento `version` non è più valido. Quando specificate un valore per `versionNumber`, dovete usare una sequenza composta da un massimo di tre numeri separati da punti, come in "0.1.2". Ogni segmento del numero di versione può contenere fino a tre cifre. (In altre parole, "999.999.999" è il numero di versione più alto possibile.) Non dovete includere necessariamente tutti e tre i segmenti nel numero: "1" e "1.0" sono numeri di versione perfettamente validi.

Potete anche specificare un'etichetta per la versione utilizzando l'elemento `versionLabel`. Quando aggiungete un'etichetta di versione, questa viene visualizzata al posto del numero di versione, ad esempio nelle finestre di installazione dell'applicazione Android. È necessario specificare un'etichetta di versione per le applicazioni che vengono distribuite tramite Android Market. Se non specificate il valore `versionLabel` nel descrittore dell'applicazione AIR, al campo dell'etichetta di versione Android viene assegnato il valore `versionNumber`.

```
<!-- AIR 2.5 and later -->
    <versionNumber>1.23.7</versionNumber>
    <versionLabel>1.23 Beta 7</versionLabel>
```

In Android, il dato AIR `versionNumber` viene convertito nel numero intero Android `versionCode` utilizzando la formula $a*1000000 + b*1000 + c$, dove a, b e c sono i componenti del numero di versione AIR: a.b.c.

Altri argomenti presenti nell' Aiuto

[“version”](#) a pagina 250

[“versionLabel”](#) a pagina 251

[“versionNumber”](#) a pagina 251

SWF principale dell'applicazione

Specificate il file SWF principale dell'applicazione nell'elemento `content` dell'elemento `initialWindow`. Quando un'applicazione ha come target dispositivi nel profilo mobile, dovete utilizzare un file SWF (le applicazioni basate su HTML non sono supportate).

```
<initialWindow>
    <content>MyApplication.swf</content>
</initialWindow>
```

Dovete includere il file nel pacchetto AIR (usando ADT o il vostro ambiente IDE). Un semplice riferimento al nome nel descrittore dell'applicazione non determina l'inclusione automatica del file nel pacchetto.

Proprietà della schermata principale

Vari elementi secondari dell'elemento `initialWindow` controllano l'aspetto iniziale e il comportamento della finestra principale dell'applicazione.

- **aspectRatio** - Specifica se l'applicazione deve essere inizialmente visualizzata in formato verticale (*portrait*, altezza superiore alla larghezza) o orizzontale (*landscape*, altezza inferiore alla larghezza) oppure in un formato qualsiasi (*any*, stage orientato automaticamente in tutte le direzioni).

```
<aspectRatio>landscape</aspectRatio>
```

- **autoOrients** - Specifica se lo stage deve cambiare orientamento automaticamente quando l'utente ruota il dispositivo o esegue un altro gesto relativo all'orientamento, come l'apertura o la chiusura di una tastiera scorrevole. Se il valore è *false* (impostazione predefinita), lo stage non cambia orientamento con il dispositivo.

```
<autoOrients>true</autoOrients>
```

- **depthAndStencil** - Specifica di utilizzare il buffer di profondità o di stencil. Solitamente questi buffer vengono utilizzati quando si lavora con contenuti 3D.

```
<depthAndStencil>true</depthAndStencil>
```

- **fullScreen** - Specifica se l'applicazione deve occupare interamente il display del dispositivo oppure condividere il display con il chrome standard del sistema operativo, ad esempio una barra di stato.

```
<fullScreen>true</fullScreen>
```

- **renderMode** - Specifica se il runtime deve eseguire il rendering dell'applicazione con la GPU (unità di elaborazione grafica) o con la CPU (unità di elaborazione centrale) principale. In generale, il rendering tramite GPU viene eseguito a una velocità superiore, ma alcune funzionalità, quali i metodi di fusione e i filtri `PixelBender`, non sono disponibili in modalità GPU. Inoltre, le funzionalità e i limiti della GPU variano da un dispositivo o driver di dispositivo all'altro. Dovete sempre provare l'applicazione con la gamma di dispositivi più ampia possibile, specialmente quando scegliete la modalità GPU.

Potete impostare la modalità di rendering su *gpu*, *cpu*, *direct* o *auto*. Il valore predefinito è *auto*, che attualmente corrisponde per impostazione predefinita alla modalità CPU.

Nota: per poter sfruttare l'accelerazione GPU dei contenuti Flash con AIR per le piattaforme mobili, Adobe consiglia di utilizzare `renderMode="direct"` (ovvero, `Stage3D`) anziché `renderMode="gpu"`. Adobe supporta e consiglia ufficialmente i seguenti framework basati su `Stage3D`: `Starling (2D)` e `Away3D (3D)`. Per maggiori dettagli su `Stage3D` e `Starling/Away3D`, visitate <http://gaming.adobe.com/getstarted/>.

```
<renderMode>direct</renderMode>
```

Nota: Non è possibile utilizzare `renderMode="direct"` per le applicazioni eseguite in background.

I limiti della modalità GPU sono i seguenti:

- Il framework `Flex` non supporta tale modalità.
- I filtri non sono supportati.
- Le fusioni `PixelBender` e i riempimenti non sono supportati.
- I seguenti metodi di fusione non sono supportati: `Livello`, `Alfa`, `Cancella`, `Sovrapponi`, `Luce intensa`, `Schiarisci` e `Scurisci`.
- L'uso della modalità di rendering GPU in un'applicazione che riproduce video è sconsigliato.
- Nella modalità di rendering tramite GPU, i campi di testo non vengono spostati correttamente in un'area visibile quando si apre la tastiera virtuale. Per assicurare che il campo di testo sia visibile quando l'utente immette il testo, usate la proprietà `softKeyboardRect` degli eventi dello stage e della tastiera virtuale per spostare il campo di testo nell'area visibile.

- Un oggetto di visualizzazione del quale non può essere eseguito il rendering tramite GPU non viene visualizzato del tutto. Ad esempio, se viene applicato un filtro a un oggetto di visualizzazione, l'oggetto non risulta visibile.

Nota: l'implementazione GPU per iOS in AIR 2.6+ è molto diversa da quelle adottate nella versione precedente, AIR 2.0, relativamente alle tecniche di ottimizzazione.

Altri argomenti presenti nell' Aiuto

["aspectRatio"](#) a pagina 221

["autoOrients"](#) a pagina 221

["depthAndStencil"](#) a pagina 225

["fullScreen"](#) a pagina 233

["renderMode"](#) a pagina 244

Profili supportati

Potete aggiungere l'elemento `supportedProfiles` per specificare quali profili di dispositivo sono supportati dall'applicazione. Utilizzate il profilo `mobileDevice` per i dispositivi mobili. Quando eseguite l'applicazione con ADL (Adobe Debug Launcher), questo tool usa il primo profilo nell'elenco come profilo attivo. Potete anche usare il flag `-profile` quando eseguite ADL per selezionare un profilo particolare nell'elenco di quelli supportati. Se l'applicazione può essere eseguita in tutti i profili, potete omettere del tutto l'elemento `supportedProfiles`. In questo caso, ADL usa il profilo `desktop` come profilo attivo predefinito.

Per specificare che l'applicazione supporta sia il profilo per dispositivi mobili che il profilo `desktop`, e che in genere desiderate testarla nel profilo mobile, aggiungete il seguente elemento:

```
<supportedProfiles>mobileDevice desktop</supportedProfiles>
```

Altri argomenti presenti nell' Aiuto

["supportedProfiles"](#) a pagina 248

["Profili dispositivo"](#) a pagina 254

["ADL \(AIR Debug Launcher\)"](#) a pagina 165

Estensioni native obbligatorie

Le applicazioni che supportano il profilo `mobileDevice` possono utilizzare le estensioni native.

Dichiarate tutte le estensioni native utilizzate dall'applicazione AIR nel descrittore dell'applicazione. L'esempio seguente illustra la sintassi da adottare per specificare due estensioni native richieste:

```
<extensions>
    <extensionID>com.example.extendedFeature</extensionID>
    <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

L'elemento `extensionID` ha lo stesso valore dell'elemento `id` nel file descrittore dell'estensione. Quest'ultimo è un file XML denominato `extension.xml`, impacchettato nel file ANE fornito dallo sviluppatore dell'estensione nativa.

Comportamento della tastiera virtuale

Impostate l'elemento `softKeyboardBehavior` su `none` per disattivare il comportamento automatico di scorrimento e ridimensionamento utilizzato dal runtime per fare in modo che il campo di immissione testo attivo sia visibile dopo che è stata aperta la tastiera virtuale. Se disattivate il comportamento automatico, spetta all'applicazione il compito di fare in modo che l'area di immissione del testo, o qualsiasi contenuto rilevante, sia visibile dopo l'attivazione della tastiera. Potete usare la proprietà `softKeyboardRect` dello stage in combinazione con `SoftKeyboardEvent` per rilevare il momento in cui la tastiera si apre e determinare l'area che va a coprire.

Per abilitare il comportamento automatico, impostate il valore dell'elemento su `pan`:

```
<softKeyboardBehavior>pan</softKeyboardBehavior>
```

Poiché `pan` è il valore predefinito, anche l'omissione dell'elemento `softKeyboardBehavior` ha l'effetto di abilitare il comportamento automatico della tastiera.

Nota: quando si usa il rendering tramite GPU, il comportamento di scorrimento ("`pan`") non è supportato.

Altri argomenti presenti nell' Aiuto

[“softKeyboardBehavior”](#) a pagina 247

[Stage.softKeyboardRect](#)

[SoftKeyboardEvent](#)

Impostazioni Android

Nella piattaforma Android, potete usare l'elemento `android` del descrittore dell'applicazione per aggiungere informazioni al file manifest dell'applicazione Android, che è un file di proprietà dell'applicazione utilizzato dal sistema operativo Android. ADT genera automaticamente il file `AndroidManifest.xml` quando create il pacchetto APK. AIR imposta alcune proprietà sui valori richiesti per garantire il funzionamento di determinate funzioni. Eventuali altre proprietà impostate nella sezione `android` del descrittore dell'applicazione AIR vengono aggiunte alla sezione corrispondente del file `Manifest.xml`.

Nota: per la maggior parte delle applicazioni AIR, dovete impostare le autorizzazioni Android necessarie per l'applicazione nell'elemento `android`, ma in genere non occorre impostare altre proprietà.

Potete impostare solo gli attributi che accettano stringhe, numeri interi o valori booleani. L'impostazione di riferimenti a risorse interne al pacchetto dell'applicazione non è supportata.

Nota: Il runtime richiede una versione SDK minima uguale o maggiore di 14. Se desiderate creare un'applicazione solo per versioni successive, assicuratevi che il file manifest includa `<uses-sdk android:minSdkVersion=" "></uses-sdk>` con la versione corretta.

Impostazioni Android manifest riservate

AIR imposta varie voci del file manifest nel documento Android manifest generato, al fine di garantire che le funzioni dell'applicazione e del runtime funzionino correttamente. Non è possibile definire le seguenti impostazioni:

Elemento manifest

Non è possibile impostare gli attributi seguenti dell'elemento manifest:

- `package`
- `android:versionCode`
- `android:versionName`

- xmlns:android

Elemento activity

Non è possibile impostare gli attributi seguenti dell'elemento activity principale:

- android:label
- android:icon

Elemento application

Non è possibile impostare gli attributi seguenti dell'elemento application:

- android:theme
- android:name
- android:label
- android:windowSoftInputMode
- android:configChanges
- android:screenOrientation
- android:launchMode

Autorizzazioni Android

Il modello di sicurezza Android richiede che ogni applicazione richieda l'autorizzazione prima di utilizzare funzioni che hanno un impatto sulla sicurezza o sulla privacy. Queste autorizzazioni devono essere specificate quando si crea il pacchetto dell'applicazione e non possono essere modificate in fase di runtime. Quando un utente installa un'applicazione, il sistema operativo Android segnala quali autorizzazioni sono richieste per tale applicazione. Se un'autorizzazione necessaria per una particolare funzione non viene richiesta, il sistema operativo Android potrebbe generare un'eccezione quando l'applicazione tenta di accedere a tale funzione, ma non è sicuro che ciò avvenga. Le eccezioni vengono trasmesse dal runtime all'applicazione. In caso di esito negativo senza eccezione, un messaggio di mancata autorizzazione viene aggiunto al registro di sistema Android.

In AIR, le autorizzazioni Android vengono specificate nell'elemento `android` del descrittore dell'applicazione. Le autorizzazioni vanno aggiunte nel formato seguente (`PERMISSION_NAME` è il nome di un'autorizzazione Android).

```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest>
    <uses-permission
android:name="android.permission.PERMISSION_NAME" />
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Le istruzioni `uses-permissions` all'interno dell'elemento `manifest` vengono aggiunte direttamente al documento Android `manifest`.

Le seguenti autorizzazioni sono richieste per l'uso di varie funzioni AIR:

ACCESS_COARSE_LOCATION Consente all'applicazione di accedere ai dati di posizione della rete WiFi e cellulare tramite la classe `Geolocation`.

ACCESS_FINE_LOCATION Consente all'applicazione di accedere ai dati GPS tramite la classe `Geolocation`.

ACCESS_NETWORK_STATE e ACCESS_WIFI_STATE Consente all'applicazione di accedere alle informazioni di rete tramite la classe NetworkInfo.

CAMERA Consente all'applicazione di accedere alla fotocamera.

Nota: quando chiedete l'autorizzazione a utilizzare la funzione fotocamera, Android presume che anche l'applicazione richieda l'uso della fotocamera. Se la fotocamera è una funzione opzionale dell'applicazione, dovete aggiungere un elemento `uses-feature` al file manifest per la fotocamera, impostando l'attributo richiesto su `false`. Vedete ["Filtro di compatibilità Android"](#) a pagina 82.

INTERNET Consente all'applicazione di effettuare richieste di rete e di eseguire il debug remoto.

READ_PHONE_STATE Consente al runtime AIR di disattivare l'audio durante le chiamate telefoniche. Dovete impostare questa autorizzazione se l'applicazione riproduce l'audio quando è in background.

RECORD_AUDIO Consente all'applicazione di accedere al microfono.

WAKE_LOCK e DISABLE_KEYGUARD Consente all'applicazione di impedire che il dispositivo entri in modalità sleep utilizzando le impostazioni della classe SystemIdleMode.

WRITE_EXTERNAL_STORAGE Consente all'applicazione di scrivere sulla scheda di memoria esterna del dispositivo.

Ad esempio, per impostare le autorizzazioni per un'applicazione che richiede ogni tipo di autorizzazione, potreste aggiungere il codice seguente al descrittore dell'applicazione:

```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest>
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission
android:name="android.permission.DISABLE_KEYGUARD" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO"
/>
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Altri argomenti presenti nell' Aiuto

[Sicurezza e autorizzazioni Android](#)

[Classe Android Manifest.permission](#)

Schemi URI personalizzati per Android

Potete usare uno schema URI personalizzato per avviare un'applicazione AIR da una pagina Web o un'applicazione Android nativa. Il supporto degli URI personalizzati dipende dai filtri di intento specificati nel file manifest di Android, quindi non è possibile sfruttare questa tecnica su altre piattaforme.

Per utilizzare un URI personalizzato, aggiungete una voce `intent-filter` al descrittore dell'applicazione all'interno del blocco `<android>`. Nell'esempio seguente, entrambi gli elementi `intent-filter` devono essere specificati. Modificate l'istruzione `<data android:scheme="my-customuri"/>` in modo che corrisponda alla stringa URI del vostro schema personalizzato.

```
<android>
    <manifestAdditions>
    <![CDATA[
    <manifest>
    <application>
    <activity>
    <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="my-customuri"/>
    </intent-filter>
    </activity>
    </application>
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Un filtro di intento indica al sistema operativo Android che l'applicazione è disponibile per l'esecuzione di una determinata azione. Nel caso di un URI personalizzato, questo significa che l'utente ha fatto clic su un collegamento che utilizza quello schema URI (e il browser non sa come gestirlo).

Quando l'applicazione viene attivata da un URI personalizzato, l'oggetto `NativeApplication` invia un evento `invoke`. L'URL del collegamento, inclusi i parametri di query, viene inserito nell'array `arguments` dell'oggetto `InvokeEvent`. Potete usare un numero illimitato di elementi `intent-filter`.

Nota: i collegamenti in un'istanza `StageWebView` non possono aprire URL che impiegano schemi URI personalizzati.

Altri argomenti presenti nell' Aiuto

[Filtri di intento Android](#)

[Azioni e categorie Android](#)

Filtro di compatibilità Android

Il sistema operativo Android usa vari elementi nel file manifest di un'applicazione per determinare se l'applicazione è compatibile con un particolare dispositivo. L'aggiunta di queste informazioni al file manifest è opzionale. Se non includete questi elementi, l'applicazione può essere installata su qualsiasi dispositivo Android. Tuttavia, non vi è garanzia che possa funzionare correttamente su qualunque dispositivo Android. Ad esempio, un'applicazione per fotocamera non sarà molto utile su un telefono che non dispone di una fotocamera.

I tag manifest Android che potete usare per filtrare sono:

- supports-screens
- uses-configuration
- uses-feature
- uses-sdk (in AIR 3+)

Applicazioni per fotocamera

Se richiedete l'autorizzazione camera per l'applicazione, Android presuppone che l'applicazione necessiti di tutte le funzioni della fotocamera disponibili, incluso la messa a fuoco automatica e il flash. Se l'applicazione non richiede tutte le funzioni della fotocamera, oppure se la fotocamera è una funzione opzionale, dovreste impostare i vari elementi `uses-feature` della fotocamera in modo da indicare che sono opzionali. In caso contrario, gli utenti con dispositivi in cui manca una particolare funzione o che non dispongono di una fotocamera, non saranno in grado di trovare la vostra applicazione in Android Market.

L'esempio seguente illustra come richiedere l'autorizzazione per la fotocamera e rendere opzionali tutte le funzioni della fotocamera:

```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest>
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera"
android:required="false"/>
    <uses-feature
android:name="android.hardware.camera.autofocus" android:required="false"/>
    <uses-feature android:name="android.hardware.camera.flash"
android:required="false"/>
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Applicazioni di registrazione audio

Se richiedete l'autorizzazione per la registrazione audio, Android presume anche che l'applicazione necessiti di un microfono. Se la registrazione audio è una funzione opzionale dell'applicazione, potete aggiungere un tag `uses-feature` per specificare che il microfono non è richiesto. In caso contrario, gli utenti con dispositivi privi di microfono non saranno in grado di trovare la vostra applicazione in Android Market.

L'esempio seguente illustra come richiedere l'autorizzazione per l'uso del microfono mantenendo comunque opzionale la presenza di un microfono hardware:


```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest>
    <uses-permission
android:name="android.permission.RECORD_AUDIO" />
    <uses-feature android:name="android.hardware.microphone"
android:required="false"/>
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Altri argomenti presenti nell' Aiuto

[Sviluppatori Android: compatibilità Android](#)

[Sviluppatori Android: costanti dei nomi di funzioni Android](#)

Posizione di installazione

Potete consentire che l'applicazione venga installata o spostata sulla scheda di memoria esterna impostando l'attributo `installLocation` dell'elemento Android `manifest` su `auto` oppure su `preferExternal`:

```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest android:installLocation="preferExternal"/>
    ]]>
    </manifestAdditions>
</android>
```

Il sistema operativo Android non garantisce che l'applicazione sarà installata sulla memoria esterna. Un utente può anche spostare l'applicazione tra la memoria interna e quella esterna utilizzando le impostazioni del sistema.

Anche quando un'applicazione è installata nella memoria esterna, la cache e i dati utente dell'applicazione, quali il contenuto della directory `app-storage`, gli oggetti condivisi e i file temporanei vengono comunque salvati nella memoria interna. Per evitare di usare una quantità eccessiva di memoria interna, siate selettivi nella scelta dei dati da salvare nella directory di archiviazione dell'applicazione. Eventuali grandi quantità di dati vanno salvate sulla scheda SD utilizzando la posizione `File.userDirectory` o `File.documentsDirectory` (entrambe sono mappate alla directory principale della scheda SD in Android).

Abilitazione di Flash Player e altri plug-in in un oggetto StageWebView

In Android 3.0+, un'applicazione deve abilitare l'accelerazione hardware nell'elemento Android `application` per consentire la visualizzazione del contenuto plug-in in un oggetto `StageWebView`. Per abilitare il rendering tramite plugin, impostate l'attributo `android.hardwareAccelerated` dell'elemento `application` su `true`:

```
<android>
    <manifestAdditions>
    <![CDATA [
    <manifest>
    <application android:hardwareAccelerated="true"/>
    </manifest>
    ]]>
    </manifestAdditions>
</android>
```

Profondità del colore

AIR 3+

In AIR 3 e versioni successive, il runtime imposta la visualizzazione per il rendering con colori a 32 bit. Nelle versioni precedenti di AIR, il runtime usa il colore a 16 bit. Potete istruire il runtime a utilizzare il colore a 16 bit mediante l'elemento `<colorDepth>` del descrittore dell'applicazione:

```
<android>
    <colorDepth>16bit</colorDepth>
    <manifestAdditions>...</manifestAdditions>
</android>
```

L'uso della profondità colore a 16 bit può migliorare le prestazioni di rendering, ma con una riduzione della fedeltà dei colori.

Impostazioni iOS

Le impostazioni che riguardano esclusivamente i dispositivi iOS vengono inserite nell'elemento `<iPhone>` del descrittore dell'applicazione. L'elemento `iPhone` può avere come elementi secondari un elemento `InfoAdditions`, un elemento `requestedDisplayResolution`, un elemento `Entitlements`, un elemento `externalSwfs` e un elemento `forceCpuRenderModeForDevices`.

L'elemento `InfoAdditions` permette di specificare coppie chiave-valore che vengono aggiunte al file di impostazioni `Info.plist` dell'applicazione. Ad esempio, i valori seguenti impostano lo stile della barra di stato dell'applicazione e indicano che l'applicazione non richiede un accesso WiFi continuo.

```
<InfoAdditions>
    <![CDATA [
        <key>UIStatusBarStyle</key>
        <string>UIStatusBarStyleBlackOpaque</string>
        <key>UIRequiresPersistentWiFi</key>
        <string>NO</string>
    ]]>
</InfoAdditions>
```

Le impostazioni `InfoAdditions` sono racchiuse in un tag `CDATA`.

L'elemento `Entitlements` permette di specificare coppie chiave-valore che vengono aggiunte al file di impostazioni `Entitlements.plist` dell'applicazione. Le impostazioni `Entitlements.plist` consentono all'applicazione di accedere a determinate funzioni iOS, quali le notifiche push.

Per informazioni più dettagliate sulle impostazioni `Info.plist` e `Entitlements.plist`, consultate la documentazione per gli sviluppatori Apple.

Supporto delle attività in background in iOS

AIR 3.3

Adobe AIR 3.3 e versioni successive supporta il multitasking in iOS mediante l'abilitazione di determinati comportamenti di background:

- Audio
- Aggiornamenti della posizione
- Connettività di rete
- Possibilità di non eseguire l'app in background

Nota: Con SWF versione 21 e precedenti, AIR non supporta l'esecuzione in background su iOS e Android se l'opzione `renderMode` è impostata su `direct`. A causa di tale restrizione, le applicazioni basate su Stage3D non possono eseguire attività in background come la riproduzione audio, gli aggiornamenti della posizione, upload o download tramite rete e così via. iOS non consente OpenGL o il rendering delle chiamate in background. Le applicazioni che cercano di effettuare chiamate OpenGL in background vengono chiuse da iOS. Android non impedisce alle applicazioni di effettuare chiamate OpenGL in background o di eseguire altre attività in background, come la riproduzione audio. Con SWF versione 22 e successive, le applicazioni AIR per dispositivi mobili possono essere eseguite in background se l'opzione `renderMode` è impostata su `direct`. Il runtime di iOS AIR risulta in un errore ActionScript (3768: L'API Stage3D non può essere utilizzata durante l'esecuzione in background) se le chiamate OpenGL vengono effettuate in background. Tuttavia in Android non si verificano errori in quanto le applicazioni native possono effettuare chiamate OpenGL in background. Per un utilizzo ottimale delle risorse per dispositivi mobili, non effettuate chiamate di rendering mentre c'è un'applicazione in esecuzione in background.

Audio in background

Per abilitare la riproduzione e la registrazione in background, includete la seguente coppia chiave-valore nell'elemento `InfoAdditions`:

```
<InfoAdditions>
    <![CDATA [
      <key>UIBackgroundModes</key>
      <array>
        <string>audio</string>
      </array>
    ]]>
</InfoAdditions>
```

Aggiornamenti della posizione in background

Per abilitare gli aggiornamenti della posizione in background, includete la seguente coppia chiave-valore nell'elemento `InfoAdditions`:

```
<InfoAdditions>
    <![CDATA [
      <key>UIBackgroundModes</key>
      <array>
        <string>location</string>
      </array>
    ]]>
</InfoAdditions>
```

Nota: utilizzate questa funzione solo se necessario, poiché le API di localizzazione determinano un consumo notevole della batteria.

Connettività in background

Per eseguire brevi attività in background, l'applicazione imposta la proprietà `NativeApplication.nativeApplication.executeInBackground` su `true`.

Ad esempio, l'applicazione potrebbe avviare un'operazione di caricamento file dopo la quale l'utente porta in primo piano un'altra applicazione. Quando l'applicazione riceve un evento di completamento upload, può impostare `NativeApplication.nativeApplication.executeInBackground` su `false`.

L'impostazione della proprietà `NativeApplication.nativeApplication.executeInBackground` su `true` non garantisce che l'applicazione venga eseguita a tempo indefinito, poiché iOS impone un limite di tempo per le attività in background. Quando iOS arresta l'elaborazione in background, AIR invia l'evento `NativeApplication.suspend`.

Possibilità di non eseguire l'app in background

È possibile impedire esplicitamente l'esecuzione in background dell'applicazione includendo la seguente coppia chiave-valore nell'elemento `InfoAdditions`:

```
<InfoAdditions>
    <![CDATA [
        <key>UIApplicationExitsOnSuspend</key>
        <true/>
    ]]>
</InfoAdditions>
```

Impostazioni iOS InfoAdditions riservate

AIR imposta varie voci del file nel file `Info.plist` generato, al fine di garantire che le funzioni dell'applicazione e del runtime funzionino correttamente. Non è possibile definire le seguenti impostazioni:

<code>CFBundleDisplayName</code>	<code>CTInitialWindowTitle</code>
<code>CFBundleExecutable</code>	<code>CTInitialWindowVisible</code>
<code>CFBundleIconFiles</code>	<code>CTIosSdkVersion</code>
<code>CFBundleIdentifier</code>	<code>CTMaxSWFMajorVersion</code>
<code>CFBundleInfoDictionaryVersion</code>	<code>DTPlatformName</code>
<code>CFBundlePackageType</code>	<code>DTSDKName</code>
<code>CFBundleResourceSpecification</code>	<code>MinimumOSVersion</code> (riservato fino a 3.2)
<code>CFBundleShortVersionString</code>	<code>NSMainNibFile</code>
<code>CFBundleSupportedPlatforms</code>	<code>UIInterfaceOrientation</code>
<code>CFBundleVersion</code>	<code>UIStatusBarHidden</code>
<code>CTAutoOrients</code>	<code>UISupportedInterfaceOrientations</code>

Nota: È possibile definire `MinimumOSVersion`. La definizione di `MinimumOSVersion` viene eseguita in Air 3.3 e versioni successive.

Supporto di modelli di dispositivo iOS differenti

Per il supporto iPad, includete le impostazioni chiave-valore appropriate per `UIDeviceFamily` nell'elemento `InfoAdditions`. L'impostazione `UIDeviceFamily` è un array di stringhe. Ogni stringa definisce i dispositivi supportati. L'impostazione `<string>1</string>` definisce il supporto per l'iPhone e l'iPod Touch. L'impostazione `<string>2</string>` definisce il supporto per l'iPad. L'impostazione `<string>3</string>` definisce il supporto per tvOS. Se specificate una sola di queste stringhe, solo quella famiglia di dispositivi è supportata. Ad esempio, l'impostazione seguente limita il supporto all'iPad:

```
<key>UIDeviceFamily</key>
    <array>
        <string>2</string>
    </array>>
```

L'impostazione seguente supporta entrambe le famiglie di dispositivi (iPhone/iPod Touch e iPad):

```
<key>UIDeviceFamily</key>
    <array>
        <string>1</string>
        <string>2</string>
    </array>
```

Inoltre, in AIR 3.7 e versioni successive, potete utilizzare il tag `forceCPURenderModeForDevices` per richiedere la modalità di rendering CPU per un insieme di dispositivi specificato e abilitare la modalità di rendering GPU per i dispositivi iOS rimanenti.

Aggiungete questo tag come tag secondario del tag `iPhone` e specificate un elenco separato da spazi contenente i nomi di modello dei dispositivi. Per un elenco di nomi di modello dei dispositivi validi, consultate [“forceCPURenderModeForDevices”](#) a pagina 232.

Ad esempio, per utilizzare la modalità CPU per iPod, iPhone e iPad meno recenti e attivare la modalità GPU per tutti gli altri dispositivi, specificate quanto segue nel descrittore dell'applicazione:

```
...
                                <renderMode>GPU</renderMode>
                                ...
                                <iPhone>
                                ...
                                <forceCPURenderModeForDevices>iPad1,1 iPhone1,1 iPhone1,2
iPod1,1
                                </forceCPURenderModeForDevices>
                                </iPhone>
```

Display ad alta risoluzione

L'elemento `requestedDisplayResolution` specifica se l'applicazione deve utilizzare la modalità di risoluzione *standard* o *elevata* nei dispositivi iOS con schermi ad alta risoluzione.

```
<requestedDisplayResolution>high</requestedDisplayResolution>
```

Nella modalità ad alta risoluzione, è possibile indirizzare individualmente ogni singolo pixel di un display ad alta risoluzione. Nella modalità standard, lo schermo del dispositivo apparirà all'applicazione come schermo a risoluzione standard. L'operazione di disegno di un pixel singolo in questa modalità imposta il colore di quattro pixel sullo schermo ad alta risoluzione.

L'impostazione predefinita è `standard`. Considerate che per i dispositivi iOS di destinazione, dovete utilizzare l'elemento `requestedDisplayResolution` come elemento secondario dell'elemento `iPhone` (non l'elemento `InfoAdditions` o `initialWindow`).

Se desiderate utilizzare impostazioni diverse in vari dispositivi, specificate il valore predefinito come valore dell'elemento `requestedDisplayResolution`. Utilizzate l'attributo `excludeDevices` per specificare dispositivi che dovrebbero utilizzare il valore opposto. Ad esempio, con il seguente codice, la modalità ad alta risoluzione è utilizzata per tutti i dispositivi che la supportano, eccetto gli iPad di terza generazione, che utilizzano la modalità standard:

```
<requestedDisplayResolution excludeDevices="iPad3">high</requestedDisplayResolution>
```

L'attributo `excludeDevices` è disponibile in AIR 3.6 e versioni successive.

Altri argomenti presenti nell' Aiuto

[“requestedDisplayResolution”](#) a pagina 245

[Renaun Erickson: Developing for both retina and non-retina iOS screens using AIR 2.6](#)

Schemi URI personalizzati per iOS

Potete registrare uno schema URI personalizzato per consentire all'applicazione di essere chiamata da un collegamento inserito in una pagina Web o in un'altra applicazione, nativa, del dispositivo. Per registrare uno schema URI, aggiungete una chiave `CFBundleURLTypes` all'elemento `InfoAdditions`. L'esempio seguente registra uno schema URI denominato `com.example.app` per consentire a un'applicazione di essere chiamata dagli URL con il formato `example://foo`.

```
<key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>example</string>
      </array>
      <key>CFBundleURLName</key>
      <string>com.example.app</string>
    </dict>
  </array>
```

Quando l'applicazione viene attivata da un URI personalizzato, l'oggetto `NativeApplication` invia un evento `invoke`. L'URL del collegamento, inclusi i parametri di query, viene inserito nell'array `arguments` dell'oggetto `InvokeEvent`. Potete usare un numero illimitato di schemi URI personalizzati.

Nota: i collegamenti in un'istanza `StageWebView` non possono aprire URL che impiegano schemi URI personalizzati.

Nota: se un'altra applicazione ha già registrato uno schema, la vostra applicazione non può sostituirla come applicazione registrata per quello schema URI.

Filtro di compatibilità iOS

Aggiungete delle voci a un array `UIRequiredDeviceCapabilities` nell'elemento `InfoAdditions` se l'applicazione deve essere utilizzata solo con dispositivi provvisti di specifiche funzionalità hardware o software. Ad esempio, la voce seguente indica che un'applicazione richiede una fotocamera e un microfono:

```
<key>UIRequiredDeviceCapabilities</key>
  <array>
    <string>microphone</string>
    <string>still-camera</string>
  </array>
```

Se un dispositivo non è dotato della funzionalità corrispondente, l'applicazione non può essere installata. Le impostazioni relative alle funzionalità rilevanti per le applicazioni AIR sono:

telephony	camera-flash
wifi	video-camera
sms	accelerometer
still-camera	location-services
auto-focus-camera	gps
front-facing-camera	microphone

AIR 2.6+ aggiunge automaticamente `armv7` e `opengles-2` all'elenco delle funzionalità richieste.

Nota: non dovete obbligatoriamente includere queste funzionalità nel descrittore dell'applicazione affinché l'applicazione possa utilizzarle. Utilizzate le impostazioni `UIRequiredDeviceCapabilities` solo per impedire agli utenti di installare l'applicazione nei dispositivi sui quali non può funzionare correttamente.

Uscita anziché pausa

Quando un utente abbandona un'applicazione AIR, questa passa in background ed entra in pausa. Se volete chiudere completamente un'applicazione anziché metterla in pausa, impostate la proprietà `UIApplicationExitsOnSuspend` su YES:

```
<key>UIApplicationExitsOnSuspend</key>  
    <true/>
```

Riduzione delle dimensioni di download tramite il caricamento di SWF esterni di sole risorse AIR 3.7

Potete ridurre le dimensioni iniziali di download dell'applicazione creando un pacchetto con un sottoinsieme di SWF utilizzati dall'applicazione, quindi caricando i rimanenti SWF esterni (di sole risorse) in fase di runtime mediante il metodo `Loader.load()`. Per utilizzare questa funzione, create l'applicazione in modo che ADT sposti tutti i componenti ActionScript ByteCode (ABC) dai file SWF caricati esternamente all'SWF dell'applicazione principale, dando origine a un file SWF che contiene soltanto risorse. In tal modo viene rispettata la regola di Apple Store che non consente il download di codice aggiuntivo dopo l'installazione di un'applicazione.

Per supportare i file SWF caricati esternamente (detti anche file SWF stripped) ADT effettua le seguenti operazioni:

- Legge il file di testo specificato nell'elemento `<iPhone>`, sottoelemento `<externalSwfs>`, per accedere all'elenco delimitato da linee dei file SWF da caricare al momento dell'esecuzione:

```
<iPhone>  
    . . .  
    <externalSwfs>FilewithPathsOfSWFsthatAreToNotToBePackaged.txt</externalSwfs>  
</iPhone>
```

- Trasferisce il codice ABC da ciascun file SWF caricato esternamente al file eseguibile principale.
- Omette i file SWF caricati esternamente dal file `.ipa`.
- Copia i file SWF stripped nella directory `.remoteStrippedSWFs`. Potete salvare i file SWF su un server Web host. L'applicazione caricherà tali file quando necessario durante la fase di runtime.

Indicate i file SWF da caricare in fase di runtime specificandone i nomi in un file di testo, uno per riga, come mostrato nel seguente esempio:

```
assets/Level1/Level1.swf  
assets/Level2/Level2.swf  
assets/Level3/Level3.swf  
assets/Level4/Level4.swf
```

Il percorso del file specificato è relativo al file descrittore dell'applicazione. Inoltre è necessario specificare questi file SWF come risorse nel comando `adt`.

Nota: Questa funzione è valida soltanto per la creazione di *package standard*. Per la creazione di *package rapida* (ad esempio con utilizzo di *interpreter*, *simulator* o *debug*), ADT non crea file SWF stripped.



Per ulteriori informazioni su questa funzione e un esempio di codice, consultate [External hosting of secondary SWFs for AIR apps on iOS](#) (Hosting esterno di SWF secondari per applicazioni AIR su iOS), un post del blog creato dall'ingegnere di Adobe Abhinav Dhandh.

Supporto di localizzazione geografica

Per il supporto di localizzazione geografica, aggiungete una delle seguenti coppie chiave-valore all'elemento `InfoAdditions`:

```
<InfoAdditions>
    <![CDATA [
    <key>NSLocationAlwaysUsageDescription</key>
    <string>Sample description to allow geolocation always</string>
    <key>NSLocationWhenInUseUsageDescription</key>
    <string>Sample description to allow geolocation when application
is in foreground</string>
    ]]>
</InfoAdditions>
```

Icone dell'applicazione

La tabella seguente elenca le dimensioni delle icone utilizzate su ciascuna piattaforma mobile:

Dimensioni icona	Piattaforma
29 x 29	iOS
36 x 36	Android
40 x 40	iOS
48 x 48	Android, iOS
50 x 50	iOS
57 x 57	iOS
58 x 58	iOS
60 x 60	iOS
72 x 72	Android, iOS
75 x 75	iOS
76 x 76	iOS
80 x 80	iOS
87 x 87	iOS
96 x 96	Android
100 x 100	iOS
114 x 114	iOS
120 x 120	iOS
144 x 144	Android, iOS
152 x 152	iOS
167 x 167	iOS
180 x 180	iOS
192 x 192	Android
512 x 512	Android, iOS
1024 x 1024	iOS

Specificate il percorso dei file di icone nell'elemento icon del file descrittore dell'applicazione:


```
<icon>
    <image36x36>assets/icon36.png</image36x36>
    <image48x48>assets/icon48.png</image48x48>
    <image72x72>assets/icon72.png</image72x72>
</icon>
```

Se non fornite un'icona di una determinata dimensione, viene utilizzata la successiva dimensione più grande, adattata al formato richiesto.

Icone in Android

In Android, le icone specificate nel descrittore dell'applicazione vengono utilizzate per l'icona di avvio dell'applicazione. L'icona di avvio dell'applicazione deve essere fornita come set di immagini PNG di 36 x 36, 48 x 48, 72 x 72, 96 x 96, 144 x 144 e 192 x 192 pixel. Queste dimensioni di icona vengono usate rispettivamente per gli schermi a bassa, media e alta densità.

Al momento dell'invio dell'app a Google Play Store, gli sviluppatori devono inviare l'icona di 512 x 512 pixel.

Icone in iOS

Le icone definite nel descrittore dell'applicazione vengono utilizzate nelle seguenti aree di un'applicazione iOS:

- Un'icona di 29 x 29 pixel - Icona di ricerca Spotlight per iPhone/iPod a bassa risoluzione e icona Impostazioni per iPad a bassa risoluzione.
- Un'icona di 40 x 40 pixel - Icona di ricerca Spotlight per iPad a bassa risoluzione.
- Un'icona di 48 x 48 pixel - AIR aggiunge un bordo a questa immagine e la utilizza come icona 50 x 50 per la ricerca Spotlight sugli iPad a risoluzione più bassa.
- Un'icona di 50 x 50 pixel - Icona di ricerca Spotlight per iPad a bassa risoluzione.
- Un'icona di 57 x 57 pixel - Icona di applicazione per iPhone/iPod a bassa risoluzione.
- Un'icona di 58 x 58 pixel - Icona di ricerca Spotlight per iPhone/iPod con display Retina e icona Impostazioni per iPad con display Retina.
- Un'icona di 60 x 60 pixel - Icona di applicazione per iPhone/iPod a bassa risoluzione.
- Un'icona di 72 x 72 pixel (opzionale) - Icona di applicazione per iPad a bassa risoluzione.
- Un'icona di 76 x 76 pixel (opzionale) - Icona di applicazione per iPad a bassa risoluzione.
- Un'icona di 80 x 80 pixel - Ricerca Spotlight per iPhone/iPod/iPad ad alta risoluzione.
- Un'icona di 100 x 100 pixel - Icona di ricerca Spotlight per iPad con display Retina.
- Un'icona di 114 x 114 pixel - Icona di applicazione per iPhone/iPod con display Retina.
- Un'icona di 120 x 120 pixel - Icona di applicazione per iPhone/iPod ad alta risoluzione.
- Un'icona di 152 x 152 pixel - Icona di applicazione per iPad ad alta risoluzione.
- Un'icona da 167 x 167 pixel - Icona di applicazione per iPad Pro ad alta risoluzione.
- Un'icona di 512 x 512 pixel - Icona di applicazione per iPhone/iPod/iPad a bassa risoluzione. iTunes visualizza questa icona. Il file da 512 pixel è utilizzato solo per la prova delle versioni di sviluppo dell'applicazione. Quando inviate l'applicazione finale a Apple App Store, l'immagine da 512 pixel viene inviata separatamente come un file JPG che non fa parte dell'IPA.
- Un'icona di 1024 x 1024 pixel - Icona di applicazione per iPhone/iPod/iPad con display Retina.

iOS aggiunge un effetto bagliore all'icona che non è necessario applicare all'immagine sorgente. Per rimuovere questo effetto bagliore predefinito, aggiungete le istruzioni seguenti all'elemento `InfoAdditions` nel file descrittore dell'applicazione:

```
<InfoAdditions>
    <![CDATA [
    <key>UIPrerenderedIcon</key>
    <true/>
    ]]>
</InfoAdditions>
```

Nota: in iOS, i metadati dell'applicazione vengono inseriti come metadati png nelle icone dell'applicazione in modo da consentire ad Adobe di registrare il numero di applicazioni AIR disponibili nell'app store Apple iOS. Se non volete che l'applicazione sia identificata come applicazione AIR a causa di questi metadati di icone, dovete spaccettare il file IPA, rimuovere i metadati delle icone e quindi ricreare il pacchetto. Questa procedura è descritta nell'articolo [Opt-out of AIR application analytics for iOS](#).

Altri argomenti presenti nell' Aiuto

“`icon`” a pagina 233

“`imageNxN`” a pagina 235

[Sviluppatori Android: linee guida per la progettazione delle icone](#)

[Linee guida per l'interfaccia di iOS: linee guida per la creazione di immagini e icone personalizzate](#)

Immagini di avvio per iOS

Oltre alle icone dell'applicazione, dovete fornire almeno un'immagine di avvio denominata *Default.png*. Facoltativamente, potete includere immagini di avvio distinte per orientamenti iniziali differenti, risoluzioni diverse (comprese quelle dei display Retina ad alta risoluzione e di quelli in formato 16:9) e dispositivi differenti. Potete anche includere più immagini di avvio da utilizzare quando l'applicazione viene attivata tramite un URL.

I file delle immagini di avvio, per i quali non esiste un riferimento nel descrittore dell'applicazione, devono essere inseriti nella directory principale dell'applicazione. (*Non* collocate questi file in una sottodirectory.)

Schema di attribuzione nome

Denominate l'immagine in base allo schema seguente:

```
basename + screen size modifier + urischeme + orientation + scale + device + .png
```

La porzione *basename* del nome del file è l'unica parte obbligatoria. Corrisponde a *Default* (con la D maiuscola) o al nome che specificate utilizzando la chiave `UILaunchImageFile` nell'elemento `InfoAdditions` del descrittore dell'applicazione.

La porzione *screen size modifier* specifica la dimensione dello schermo quando non è una di quelle standard. Questo modificatore va utilizzato solo per i modelli di iPhone e iPod touch con schermo in formato 16:9, come l'iPhone o 5 l'iPod touch di quinta generazione. L'unico valore supportato per questo modificatore è `-568h`. Poiché questi dispositivi supportano i display ad alta risoluzione (Retina), il modificatore della dimensione dello schermo viene sempre utilizzato con un'immagine che include anche il modificatore di scala `@2x`. Il nome completo dell'immagine di avvio predefinita per questi dispositivi è `Default-568h@2x.png`.

La porzione *urischeme* è la stringa utilizzata per identificare lo schema URI. Questa porzione viene utilizzata solo se l'app supporta uno o più schemi URL personalizzati. Ad esempio, se l'applicazione può essere attivata tramite un collegamento, come `example://foo`, utilizzate `-example` come porzione di schema del nome file dell'immagine di avvio.

La porzione *orientation* fornisce un modo per specificare più immagini di avvio da utilizzare a seconda dell'orientamento del dispositivo al momento dell'avvio dell'applicazione. Questa porzione viene utilizzata solo per le app per iPad. Può essere uno dei valori seguenti, che indicano l'orientamento del dispositivo al momento dell'avvio dell'applicazione:

- -Portrait
- -PortraitUpsideDown
- -Landscape
- -LandscapeLeft
- -LandscapeRight

La porzione *scale* è @2x (per iPhone 4, iPhone 5 e iPhone 6) o @3x (per iPhone 6 plus) per le immagini di avvio utilizzate per i display ad alta risoluzione (Retina). (Omettete completamente la porzione *scale* per le immagini utilizzate per i display ad alta risoluzione.) Per le immagini di avvio per dispositivi più alti, quali l'iPhone 5 e l'iPod touch di quinta generazione, è necessario specificare anche il modificatore della dimensione dello schermo -528h dopo la porzione *basename* e prima di qualsiasi altra porzione.

La porzione *device* viene utilizzata per specificare le immagini di avvio per i dispositivi palmari e i telefoni. Questa porzione viene utilizzata per le app universali che supportano sia i dispositivi palmari che i tablet con un unico codice app binario. Il valore deve essere ~ipad o ~iphone (sia per iPhone che per iPod Touch).

Per iPhone, è possibile includere solo immagini con orientamento verticale. Tuttavia, nel caso di iPhone 6 plus, possono inoltre essere aggiunte le immagini orizzontali. Usate immagini a 320x480 pixel per i dispositivi a risoluzione standard, a 640x960 pixel per i dispositivi ad alta risoluzione e a 640x1136 pixel per i dispositivi con display in formato 16:9 come l'iPhone 5 e l'iPod touch di quinta generazione.

Per iPad, includete le immagini nel modo seguente:

- AIR 3.3 e versioni precedenti - Immagini non a schermo intero: potete includere immagini con orientamento sia orizzontale (1024 x 748 per risoluzione normale, 2048 x 1496 per alta risoluzione) che verticale (768 x 1004 per risoluzione normale, 1536 x 2008 per alta risoluzione).
- AIR 3.4 e versioni successive - Immagini a schermo intero: potete includere immagini con orientamento sia orizzontale (1024 x 768 per risoluzione normale, 2048 x 1536 per alta risoluzione) che verticale (768 x 1024 per risoluzione normale, 1536 x 2048 per alta risoluzione). Tenete presente che quando inserite un'immagine a schermo intero in un'applicazione non a schermo intero, i primi 20 pixel in alto (primi 40 pixel per le immagini ad alta risoluzione) vengono coperti dalla barra di stato. Evitate di visualizzare informazioni importanti in quest'area.

Esempi

La tabella seguente mostra set di immagini di avvio di esempio che potreste includere per un'applicazione ipotetica che supporta la più ampia gamma possibile di dispositivi e orientamenti e che può essere avviata tramite URL utilizzando lo schema `example://`:

Nome file	Dimensione immagine	Uso
Default.png	320 x 480	iPhone, risoluzione standard
Default@2x.png	640 x 960	iPhone, alta risoluzione
Default-568h@2x.png	640 x 1136	iPhone, alta risoluzione, formato 16:9

Nome file	Dimensione immagine	Uso
Default-Portrait.png	768 x 1004 (AIR 3.3 e versioni precedenti) 768 x 1024 (AIR 3.4 e versioni successive)	iPad, orientamento verticale
Default-Portrait@2x.png	1536 x 2008 (AIR 3.3 e versioni precedenti) 1536 x 2048 (AIR 3.4 e versioni successive)	iPad, alta risoluzione, orientamento verticale
Default-PortraitUpsideDown.png	768 x 1004 (AIR 3.3 e versioni precedenti) 768 x 1024 (AIR 3.4 e versioni successive)	iPad, orientamento verticale capovolto
Default-PortraitUpsideDown@2x.png	1536 x 2008 (AIR 3.3 e versioni precedenti) 1536 x 2048 (AIR 3.4 e versioni successive)	iPad, alta risoluzione, orientamento verticale capovolto
Default-Landscape.png	1024 x 768	iPad, orientamento orizzontale a sinistra
Default-LandscapeLeft@2x.png	1536 x 2048	iPad, alta risoluzione, orientamento verticale a sinistra
Default-LandscapeRight.png	1024 x 768	iPad, orientamento orizzontale a destra
Default-LandscapeRight@2x.png	1536 x 2048	iPad, alta risoluzione, orientamento verticale a destra
Default-example.png	320 x 480	example:// URL su iPhone standard
Default-example@2x.png	640 x 960	example:// URL su iPhone ad alta risoluzione
Default-example~ipad.png	768 x 1004	example:// URL su iPad con orientamento verticale
Default-example-Landscape.png	1024 x 768	example:// URL su iPad con orientamento orizzontale

Questo esempio illustra solo uno degli approcci. Potete, ad esempio, utilizzare l'immagine `Default.png` per l'iPad e specificare immagini di avvio specifiche per l'iPhone e l'iPod con `Default~iphone.png` e `Default@2x~iphone.png`.

Vedete anche

[iOS Application Programming Guide: Application Launch Images](#)

Immagini di avvio da includere nel pacchetto per dispositivi iOS

Dispositivi	Risoluzione (pixel)	Nome immagine di avvio	Orientamento
iPhone			
iPhone 4 (non retina)	640 x 960	Default~iphone.png	Verticale
iPhone 4, 4s	640 x 960	Default@2x~iphone.png	Verticale
iPhone 5, 5c, 5s	640 x 1136	Default-568h@2x~iphone.png	Verticale
iPhone6, iPhone 7	750 x 1334	Default-375w-667h@2x~iphone.png	Verticale

iPhone6+, iPhone 7+	1242 x 2208	Default-414w-736h@3x~iphone.png	Verticale
iPhone 6+, iPhone 7+	2208 x 1242	Default-Landscape-414w-736h@3x~iphone.png	Orizzontale
iPad			
iPad 1, 2	768 x 1024	Default-Portrait~ipad.png	Verticale
iPad 1, 2	768 x 1024	Default-PortraitUpsideDown~ipad.png	Verticale capovolto
iPad 1, 2	1024 x 768	Default-Landscape~ipad.png	Orizzontale a sinistra
iPad 1, 2	1024 x 768	Default-LandscapeRight~ipad.png	Orizzontale a destra
iPad 3, Air	1536 x 2048	Default-Portrait@2x~ipad.png	Verticale
iPad 3, Air	1536 x 2048	Default-PortraitUpsideDown@2x~ipad.png	Verticale capovolto
iPad 3, Air	1536 x 2048	Default-LandscapeLeft@2x~ipad.png	Orizzontale a sinistra
iPad 3, Air	1536 x 2048	Default-LandscapeRight@2x~ipad.png	Orizzontale a destra
iPad Pro	2732 x 2048	Default-Portrait@2x.png	Verticale
iPad Pro	2732 x 2048	Default-Landscape@2x.png	Orizzontale

Linee guida grafiche

Potete creare un disegno a scelta per un'immagine di avvio, purché sia delle dimensioni corrette. Tuttavia, spesso è preferibile che l'immagine corrisponda allo stato iniziale dell'applicazione. Per creare un'immagine di avvio, potete acquisire un'istantanea della schermata di avvio dell'applicazione:

- 1 Aprite l'applicazione sul dispositivo iOS. Quando appare la prima schermata dell'interfaccia utente, tenete premuto il pulsante Home (sotto la schermata). Tenendo premuto il pulsante Home, premete il pulsante Power/Sleep (nella parte superiore del dispositivo). Questa operazione consente di scattare un'istantanea e di inviarla all'album Rullino fotografico.
- 2 Caricate l'immagine nel computer di sviluppo trasferendo le fotografie da iPhoto o da un'altra applicazione di trasferimento foto.

Non includete testo nell'immagine di avvio se l'applicazione viene localizzata in più lingue. L'immagine di avvio è statica e il testo non verrebbe adattato alle altre lingue.

Vedete anche

[iOS Human Interface Guidelines: Launch images \(Linee guida per l'interfaccia di iOS: immagini di avvio\)](#)

Impostazioni ignorate

Le applicazioni per dispositivi mobili ignorano le impostazioni applicazione valide solo per le finestre native o le funzioni del sistema operativo. Le impostazioni ignorate sono:

- allowBrowserInvocation
- customUpdateUI
- fileTypes
- height
- installFolder

- maximizable
- maxSize
- minimizable
- minSize
- programMenuFolder
- resizable
- systemChrome
- title
- transparent
- visible
- width
- x
- y

Creazione del pacchetto di un'applicazione AIR per dispositivi mobili

Utilizzate il comando `ADT -package` per creare il pacchetto per un'applicazione destinata ai dispositivi mobili. Il parametro `-target` specifica la piattaforma mobile di destinazione del pacchetto.

Pacchetti Android

Le applicazioni AIR per Android utilizzano il formato di pacchetto Android APK anziché il formato di pacchetto AIR.

I pacchetti prodotti da ADT con il tipo di target `APK` sono codificati in un formato che può essere presentato su Android Market. Android Market prevede dei requisiti che le applicazioni inviate devono rispettare per essere accettate. Informatevi sui requisiti più recenti prima di creare il pacchetto finale. Vedete [Android Developers: Publishing on the Market](#).

A differenza di quanto avviene con le applicazioni iOS, potete usare un normale certificato AIR di firma del codice per firmare l'applicazione Android; tuttavia, affinché sia possibile inviarla ad Android Market, il certificato deve essere conforme alle regole del Market, in base alle quali il certificato deve essere valido almeno fino al 2033. Potete creare un certificato di questo tipo con il comando `ADT -certificate`.

Per distribuire un'applicazione in uno spazio commerciale che non consente all'applicazione di richiedere un download AIR da Google Market, potete specificare un URL di download alternativo usando il parametro `-airDownloadURL` di ADT. Quando un utente che non dispone della versione richiesta del runtime AIR avvia l'applicazione, viene indirizzato all'URL specificato. Per ulteriori informazioni, vedete “[Comando ADT package](#)” a pagina 172.

Per impostazione predefinita, ADT crea pacchetti per l'applicazione Android con runtime condiviso. Quindi, per eseguire l'applicazione, l'utente dovrebbe installare il runtime AIR sul dispositivo separatamente.

Nota: Per forzare la creazione da parte di ADT di un APK che utilizza il runtime autonomo, utilizzare `target apk-captive-runtime`.

Pacchetti iOS

Le applicazioni AIR per iOS utilizzano il formato di pacchetto iOS (IPA) anziché il formato AIR nativo.

I pacchetti prodotti da ADT con il tipo di target *ipa-app-store* e con il certificato di firma del codice e il profilo di provisioning corretti sono codificati in un formato che può essere presentato su Apple App Store. Usate il tipo di target *ipa-ad-hoc* per creare il pacchetto di un'applicazione per la distribuzione ad hoc.

Dovete utilizzare un certificato per sviluppatori corretto, emesso da Apple, per firmare l'applicazione. I certificati impiegati per la creazione del pacchetto finale prima dell'invio dell'applicazione sono differenti da quelli utilizzati per la creazione delle build di prova.

Per un esempio di come impacchettare un'applicazione iOS mediante Ant, vedete [Piotr Walczyszyn: Packaging AIR application for iOS devices with ADT command and ANT script](#)

Creazione del pacchetto con ADT

Il kit SDK AIR versione 2.6 e successive supporta la creazione di pacchetti sia per iOS che per Android. Prima di creare il pacchetto è necessario effettuare la compilazione di tutto il codice ActionScript, MXML e di estensione (se presente). Dovete inoltre avere un certificato di firma del codice.

Per informazioni dettagliate sui comandi e le opzioni ADT, vedete “[ADT \(AIR Developer Tool\)](#)” a pagina 171.

Pacchetti Android APK

Creazione di un pacchetto APK

Per creare un pacchetto APK, usate il comando `ADT package`, impostando il tipo di target su *apk* per le build di rilascio, *apk-debug* per le build di debug o *apk-emulator* per le build in modalità rilascio da eseguire su un emulatore.

```
adt -package  
  
-target apk  
-storetype pkcs12 -keystore ../codesign.p12  
myApp.apk  
myApp-app.xml  
myApp.swf icons
```

Digitate l'intero comando su una sola riga; le interruzioni di riga nell'esempio qui sopra sono presenti solo per facilitare la leggibilità. Inoltre, nell'esempio si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316 per ulteriori informazioni.)

Dovete eseguire il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf` e una directory di icone.

Quando eseguite il comando come indicato, ADT richiede la password dell'archivio di chiavi. (I caratteri della password che digitate non vengono visualizzati; quando avete immesso tutti i caratteri della password, premete Invio.)

Nota: Per impostazione predefinita, tutte le applicazioni Android AIR dispongono di *air.* prefisso nel nome del pacchetto. Per escludere questo comportamento predefinito, impostare la variabile di ambiente `AIR_NOANDROIDFLAIR` su *true*, sul computer.

Creazione di un pacchetto APK per un'applicazione che usa estensioni native

Per creare un pacchetto APK per un'applicazione che usa estensioni native, aggiungete l'opzione `-extdir` alle normali opzioni di compilazione. Nel caso di più di ANE che condividono risorse/librerie, ADT sceglie solo una singola risorsa/libreria e ignora altre voci doppie prima della pubblicazione dell'avviso. Questa opzione consente di specificare la directory contenente i file ANE utilizzati dall'applicazione. Ad esempio:

```
adt -package
    -target apk
    -storetype pkcs12 -keystore ../codesign.p12
    MyApp.apk
    MyApp-app.xml
    -extdir extensionsDir
    MyApp.swf icons
```

Creazione di un pacchetto APK che include una propria versione del runtime AIR

Per creare un pacchetto APK che contiene sia l'applicazione che una versione autonoma del runtime AIR, specificate il target `apk-captive-runtime`. Questa opzione consente di specificare la directory contenente i file ANE utilizzati dall'applicazione. Ad esempio:

```
adt -package
    -target apk-captive-runtime
    -storetype pkcs12 -keystore ../codesign.p12
    MyApp.apk
    MyApp-app.xml
    MyApp.swf icons
```

I possibili svantaggi di questa tecnica sono:

- Patch di sicurezza critici non automaticamente disponibili ogni volta che Adobe pubblica un patch di sicurezza
- Le applicazioni richiedono più memoria RAM

Nota: quando impacchettate il runtime, ADT aggiunge le autorizzazioni `INTERNET` e `BROADCAST_STICKY` all'applicazione. Queste autorizzazioni sono richieste dal runtime AIR.

Creazione di un pacchetto APK di debug

Per creare una versione dell'applicazione utilizzabile con un debugger, usate `apk-debug` come target e specificate le opzioni di connessione:

```
adt -package
    -target apk-debug
    -connect 192.168.43.45
    -storetype pkcs12 -keystore ../codesign.p12
    MyApp.apk
    MyApp-app.xml
    MyApp.swf icons
```

Il flag `-connect` indica al runtime AIR sul dispositivo dove connettersi a un debugger remoto tramite la rete. Per eseguire il debug via USB, dovete specificare invece il flag `-listen`, indicando la porta TCP da usare per la connessione di debug:

```
adt -package
    -target apk-debug
    -listen 7936
    -storetype pkcs12 -keystore ../codesign.p12
    MyApp.apk
    MyApp-app.xml
    MyApp.swf icons
```

Per garantire il corretto funzionamento della maggior parte delle opzioni di debug, dovete inoltre compilare i file SWF e SWC dell'applicazione abilitando esplicitamente il debug. Vedete [“Opzioni per la connessione del debugger”](#) a pagina 189 per una descrizione completa dei flag `-connect` e `-listen`.

Nota: Per impostazione predefinita, ADT include una copia autonoma del runtime AIR con l'app Android durante la creazione di un'app Android con il valore target `APK-debug`. Per forzare ADT a creare un APK che utilizza un runtime esterno, impostate la variabile di ambiente `AIR_ANDROID_SHARED_RUNTIME` su `true`.

In Android, l'applicazione deve anche disporre dell'autorizzazione ad accedere a Internet per potersi connettere al computer che esegue il debugger tramite la rete. Vedete “[Autorizzazioni Android](#)” a pagina 80.

Creazione di un pacchetto Android da utilizzare su un emulatore Android

Su un emulatore Android potete usare un pacchetto APK di debug ma non un pacchetto in modalità rilascio. Per creare un pacchetto APK di rilascio da usare su un emulatore, usate il comando ADT `package`, impostando il tipo di target su `apk-emulator`:

```
adt -package -target apk-emulator -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp-  
app.xml myApp.swf icons
```

Nell'esempio, si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316 per ulteriori informazioni.)

Creazione di un pacchetto APK da un file AIR o AIRI

Potete creare un pacchetto APK direttamente da un file AIR o AIRI:

```
adt -target apk -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp.air
```

Il file AIR deve usare lo spazio dei nomi AIR 2.5 (o successivo) nel file descrittore dell'applicazione.

Creazione di un pacchetto APK per la piattaforma Android x86

A partire da AIR 14, l'argomento `-arch` può essere utilizzato per creare un pacchetto APK per la piattaforma Android x86. Ad esempio:

```
adt -package -target apk-debug -listen 7936 -arch x86 -storetype pkcs12 -keystore ../codesign.p12  
myApp.apk myApp-app.xml myApp.swf icons
```

Pacchetti iOS

In iOS, ADT converte il codice byte del file SWF e altri file di origine in un'applicazione iOS nativa.

- 1 Create il file SWF con Flash Builder, Flash Professional o un compilatore della riga di comando.
- 2 Aprite una shell dei comandi o un terminale e passate alla cartella del progetto dell'applicazione iPhone.
- 3 Quindi, utilizzate il tool ADT per creare il file IPA, utilizzando la sintassi seguente:

```
adt -package
                                     -target [ipa-test | ipa-debug | ipa-app-store | ipa-ad-
hoc |
                                     ipa-debug-interpreter | ipa-debug-interpreter-simulator
                                     ipa-test-interpreter | ipa-test-interpreter-simulator]
                                     -provisioning-profile PROFILE_PATH
                                     SIGNING_OPTIONS
                                     TARGET_IPA_FILE
                                     APP_DESCRIPTOR
                                     SOURCE_FILES
                                     -extdir extension-directory
                                     -platformsdk path-to-iOSSDK or path-to-ios-simulator-
sdk
```

Modificate il riferimento `adt` includendo il percorso completo all'applicazione `adt`. L'applicazione `adt` è installata nella directory `bin` di AIR SDK.

Selezionate l'opzione `-target` che corrisponde al tipo di applicazione iPhone che desiderate creare:

- `-target ipa-test`: scegliete questa opzione per compilare rapidamente una versione dell'applicazione per la prova sull'iPhone per sviluppatori. potete anche usare `ipa-test-interpreter` per ottenere una compilazione più veloce o `ipa-test-interpreter-simulator` per l'esecuzione in iOS Simulator.
- `-target ipa-debug`: scegliete questa opzione per compilare una versione di debug dell'applicazione per la prova sull'iPhone per sviluppatori. Con questa opzione, potete utilizzare una sessione di debug per ricevere `output trace()` dall'applicazione iPhone.

Potete includere una delle opzioni `-connect` seguenti (`CONNECT_OPTIONS`) per specificare l'indirizzo IP del computer di sviluppo che esegue il debugger:

- `-connect` - l'applicazione tenta di connettersi via wi-fi a una sessione di debug sul computer di sviluppo utilizzato per compilare l'applicazione.
- `-connect IP_ADDRESS` - l'applicazione tenta di connettersi via wi-fi a una sessione di debug sul computer con l'indirizzo IP specificato. Ad esempio:

```
-target ipa-debug -connect 192.0.32.10
```
- `-connect HOST_NAME` - l'applicazione tenta di connettersi via wi-fi a una sessione di debug sul computer con il nome host specificato. Ad esempio:

```
-target ipa-debug -connect bobroberts-mac.example.com
```

L'opzione `-connect` è opzionale. Se non specificato, l'applicazione di debug risultante non tenterà di connettersi a un debugger residente. In alternativa, potete specificare `-listen` anziché `-connect` per abilitare il debugging USB, come descritto in [“Debug remoto con FDB via USB”](#) a pagina 110.

Se una connessione di debug non riesce, l'applicazione visualizza una finestra di dialogo in cui si chiede all'utente di inserire l'indirizzo IP della macchina di debug. Un tentativo di connessione può non riuscire se il dispositivo non è connesso a una rete wifi. Inoltre, questa evenienza si può verificare se il dispositivo è connesso ma non dietro il firewall della macchina host di debug.

potete anche usare `ipa-debug-interpreter` per ottenere una compilazione più veloce o `ipa-debug-interpreter-simulator` per l'esecuzione in iOS Simulator.

Per ulteriori informazioni, vedete [“Debug di un'applicazione AIR per dispositivi mobili”](#) a pagina 104.

- `-target ipa-ad-hoc`: scegliete questa opzione per creare un'applicazione per la distribuzione ad hoc. Visitate il centro per sviluppatori Apple iPhone.

- `-target ipa-app-store`: scegliete questa opzione per creare una versione finale del file IPA per la distribuzione all'App Store di Apple.

Sostituite `PROFILE_PATH` con il percorso al file del profilo di provisioning per l'applicazione. Per ulteriori informazioni sui profili di provisioning, vedete “[Configurazione per iOS](#)” a pagina 69.

Usate l'opzione `-platformsdk` per puntare a iOS Simulator SDK quando realizzate un'applicazione da eseguire in iOS Simulator.

Sostituite `SIGNING_OPTIONS` per fare riferimento al certificato e alla password per sviluppatori iPhone. Utilizzate la sintassi seguente:

```
-storetype pkcs12 -keystore P12_FILE_PATH -storepass PASSWORD
```

Sostituite `P12_FILE_PATH` con il percorso al file di certificato P12. Sostituite `PASSWORD` con la password del certificato. (Vedete l'esempio riportato di seguito.) Per ulteriori informazioni sul file del certificato P12, vedete “[Conversione di un certificato per sviluppatori in un file di archivio chiavi P12](#)” a pagina 204.

Nota: potete utilizzare un certificato autofirmato quando compilate per iOS Simulator.

Sostituite `APP_DESCRIPTOR` per fare riferimento al file descrittore dell'applicazione.

Sostituite `SOURCE_FILES` per fare riferimento al file SWF principale del progetto seguito dalle eventuali altre risorse da includere. Includete i percorsi a tutti i file delle icone definiti nella finestra di dialogo delle impostazioni dell'applicazione in Flash Professional o in un file descrittore dell'applicazione personalizzato. Inoltre, aggiungete il file grafico della schermata iniziale, `Default.png`.

Utilizzate l'opzione `-extdir extension-directory` per specificare la directory contenente i file ANE (estensioni native) utilizzati dall'applicazione. Se l'applicazione non usa estensioni native, non includete questa opzione.

Importante: non create una sottodirectory `Resources` nella directory dell'applicazione. Il runtime crea automaticamente una cartella con questo nome per assicurare la conformità con la struttura di pacchetto IPA. La creazione di una vostra cartella `Resources` determinerebbe un conflitto non risolvibile.

Creazione di un pacchetto iOS per il debug

Per creare un pacchetto iOS da installare su dispositivi di prova, usate il comando `ADT package`, impostando il tipo di target su `ios-debug`. Prima di eseguire questo comando, dovete essere già in possesso di un certificato di firma del codice per sviluppatori e di un profilo di provisioning forniti da Apple.

```
adt -package  
  
-target ipa-debug  
-storetype pkcs12 -keystore ../AppleDevelopment.p12  
-provisioning-profile AppleDevelopment.mobileprofile  
-connect 192.168.0.12 | -listen  
myApp.ipa  
myApp-app.xml  
myApp.swf icons Default.png
```

Nota: potete anche usare `ipa-debug-interpreter` per ottenere una compilazione più veloce o `ipa-debug-interpreter-simulator` per eseguire in iOS Simulator.

Digitate l'intero comando su una sola riga; le interruzioni di riga nell'esempio qui sopra sono presenti solo per facilitare la leggibilità. Inoltre, nell'esempio si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316 per ulteriori informazioni.)

Dovete eseguire il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf`, una directory di icone e il file `Default.png`.

Dovete firmare l'applicazione usando il corretto certificato di distribuzione emesso da Apple; altri certificati di firma del codice non sono validi.

Utilizzate l'opzione `-connect` per il debug via wi-fi. L'applicazione tenta di avviare una sessione di debug con l'istanza di Flash Debugger (FDB) eseguita sull'IP o nome host specificato. Utilizzate l'opzione `-listen` per il debug via USB. Prima avviate l'applicazione e poi FDB, che apre una sessione di debug per l'applicazione in esecuzione. Vedete [“Connessione al debugger di Flash”](#) a pagina 108 per ulteriori informazioni.

Creazione di un pacchetto iOS per l'invio a Apple App Store

Per creare un pacchetto iOS da inviare ad Apple App Store, usate il comando ADT `package`, impostando il tipo di target su `ios-app-store`. Prima di eseguire questo comando, dovete essere già in possesso di un certificato di firma del codice per la distribuzione e di un profilo di provisioning forniti da Apple.

```
adt -package
                                     -target ipa-app-store
                                     -storetype pkcs12 -keystore ../AppleDistribution.p12
                                     -provisioning-profile AppleDistribution.mobileprofile
                                     myApp.ipa
                                     myApp-app.xml
                                     myApp.swf icons Default.png
```

Digitate l'intero comando su una sola riga; le interruzioni di riga nell'esempio qui sopra sono presenti solo per facilitare la leggibilità. Inoltre, nell'esempio si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete [“Variabili d'ambiente dei percorsi”](#) a pagina 316 per ulteriori informazioni.)

Dovete eseguire il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf`, una directory di icone e il file `Default.png`.

Dovete firmare l'applicazione usando il corretto certificato di distribuzione emesso da Apple; altri certificati di firma del codice non sono validi.

Importante: Apple richiede l'uso del programma *Apple Application Loader* per caricare applicazioni su App Store. Tuttavia, Apple pubblica *Application Loader* solo per Mac OS X e quindi, anche se potete sviluppare un'applicazione AIR per iPhone usando un computer Windows, dovete avere accesso a un computer con sistema operativo OS X (versione 10.5.3 o successiva) per inviarla ad App Store. Il programma *Application Loader* può essere scaricato da Apple iOS Developer Center.

Creazione di un pacchetto iOS per la distribuzione ad hoc

Per creare un pacchetto iOS per la distribuzione ad hoc, usate il comando ADT `package`, impostando il tipo di target su `ios-ad-hoc`. Prima di eseguire questo comando, dovete essere già in possesso del certificato di firma del codice per la distribuzione ad hoc e di un profilo di provisioning appropriati forniti da Apple.

```
adt -package
                                     -target ipa-ad-hoc
                                     -storetype pkcs12 -keystore ../AppleDistribution.p12
                                     -provisioning-profile AppleDistribution.mobileprofile
                                     myApp.ipa
                                     myApp-app.xml
                                     myApp.swf icons Default.png
```

Digitate l'intero comando su una sola riga; le interruzioni di riga nell'esempio qui sopra sono presenti solo per facilitare la leggibilità. Inoltre, nell'esempio si presume che il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete [“Variabili d'ambiente dei percorsi”](#) a pagina 316 per ulteriori informazioni.)

Dovete eseguire il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf`, una directory di icone e il file `Default.png`.

Dovete firmare l'applicazione usando il corretto certificato di distribuzione emesso da Apple; altri certificati di firma del codice non sono validi.

Creazione di un pacchetto iOS per un'applicazione che usa estensioni native

Per creare un pacchetto iOS per un'applicazione che usa estensioni native, usate il comando del pacchetto ADT con l'opzione `-extdir`. Usate il comando ADT in base alle destinazioni (`ipa-app-store`, `ipa-debug`, `ipa-ad-hoc`, `ipa-test`). Ad esempio:

```
adt -package  
  
-target ipa-ad-hoc  
-storetype pkcs12 -keystore ../AppleDistribution.p12  
-provisioning-profile AppleDistribution.mobileprofile  
myApp.ipa  
myApp-app.xml  
-extdir extensionsDir  
myApp.swf icons Default.png
```

Digitate l'intero comando su una sola riga; le interruzioni di riga nell'esempio qui sopra sono presenti solo per facilitare la leggibilità.

Per quanto riguarda le estensioni native, nell'esempio si presume che la directory denominata `extensionsDir` sia la directory nella quale eseguite il comando. La directory `extensionsDir` contiene i file ANE utilizzati dall'applicazione.

Debug di un'applicazione AIR per dispositivi mobili

Potete eseguire il debug di un'applicazione AIR per dispositivi mobili in diversi modi. Il modo più semplice per individuare eventuali problemi nella logica dell'applicazione è quello di eseguire il debug sul vostro computer di sviluppo utilizzando ADL o iOS Simulator. Potete anche installare l'applicazione su un dispositivo ed eseguire il debug in remoto eseguendo il debugger Flash su un computer desktop.

Simulazione del dispositivo con ADL

Il modo più veloce e facile di eseguire il test e il debug della maggior parte delle funzioni di un'applicazione per dispositivi mobili consiste nell'eseguire l'applicazione sul computer di sviluppo utilizzando l'utilità ADL (Adobe Debug Launcher). ADL usa l'elemento `supportedProfiles` nel descrittore dell'applicazione per scegliere quale profilo utilizzare. Se sono elencati più profili, ADL sceglie il primo dell'elenco. Potete anche usare il parametro `-profile` di ADL per selezionare uno degli altri profili dell'elenco `supportedProfiles`. (Se non includete l'elemento `supportedProfiles` nel descrittore dell'applicazione, potete specificare qualsiasi profilo per l'argomento `-profile`.) Ad esempio, utilizzate il comando seguente per avviare un'applicazione simulando il profilo dispositivo mobile:

```
adl -profile mobileDevice myApp-app.xml
```

Quando si simula il profilo mobile in ambiente desktop in questo modo, l'applicazione viene eseguita in un contesto il più simile possibile a quello di un dispositivo mobile di destinazione. Le API ActionScript che non fanno parte del profilo mobile non sono disponibili. Tuttavia, ADL non fa distinzione tra le funzionalità di dispositivi mobili differenti. Ad esempio, potete inviare pressioni di tasti virtuali simulate all'applicazione, anche se il dispositivo di destinazione vero e proprio non usa i tasti virtuali.

ADL supporta le simulazioni dei cambi di orientamento e dell'input dei tasti virtuali tramite i comandi di menu. Quando eseguite ADL nel profilo per dispositivi mobili, ADL visualizza un menu (nella finestra dell'applicazione o nella barra dei menu desktop) che consente di specificare la rotazione del dispositivo o l'input tramite tasti virtuali.

Input dei tasti virtuali

ADL simula i pulsanti dei tasti virtuali Indietro, Menu e Cerca di un dispositivo mobile. Potete inviare questi tasti al dispositivo simulato utilizzando il menu visualizzato nel momento in cui ADL viene avviato utilizzando il profilo mobile.

Rotazione del dispositivo

ADL consente di simulare la rotazione del dispositivo utilizzando il menu visualizzato nel momento in cui ADL viene avviato utilizzando il profilo mobile. Potete ruotare il dispositivo simulato a destra o a sinistra.

La simulazione della rotazione ha effetto solo sulle applicazioni nella quali l'orientamento automatico è abilitato. Per abilitare questa funzione, impostate l'elemento `autoOrients` su `true` nel descrittore dell'applicazione.

Dimensione dello schermo

È possibile provare l'applicazione su schermi di dimensioni differenti impostando il parametro ADL `-screensize`. Potete passare il codice per uno dei tipi di schermo predefiniti oppure una stringa contenente i quattro valori che rappresentano le dimensioni in pixel degli schermi in formato normale e ingrandito.

Specificate sempre le dimensioni in pixel per il layout verticale, ovvero specificate una larghezza di valore inferiore a quello dell'altezza. Ad esempio, il comando seguente aprirebbe ADL in modo da simulare lo schermo di Motorola Droid:

```
adl -screensize 480x816:480x854 myApp-app.xml
```

Per un elenco dei tipi di schermo predefiniti, vedete “[Utilizzo di ADL](#)” a pagina 165.

Limitazioni

Alcune API che non sono supportate nel profilo desktop non possono essere simulate da ADL. Le API non simulate sono:

- Accelerometer
- `cacheAsBitmapMatrix`
- CameraRoll
- CameraUI
- Geolocation
- Multitouch e i gesti nei sistemi operativi che non supportano questa funzioni
- `SystemIdleMode`

Se l'applicazione fa uso di queste classi, dovete provare le funzioni su un dispositivo effettivo o su un emulatore.

Analogamente, vi sono API che funzionano se eseguite con ADL in ambiente desktop, mentre non funzionano con tutti i tipi di dispositivi mobili. Ad esempio:

- Codec audio Speex e AAC
- Supporto accessibilità e screen
- RTMPE
- Caricamento di file SWF contenenti codice byte ActionScript
- Shader PixelBender

Provate sempre le applicazioni che usano queste funzioni sui dispositivi di destinazione, poiché ADL non replica interamente l'ambiente di esecuzione.

Simulazione del dispositivo con iOS Simulator

iOS Simulator (disponibile solo per Mac) offre un modo rapido per eseguire applicazioni iOS ed effettuare il debug. Quando testate un'applicazione con iOS Simulator, non avete bisogno di un certificato per sviluppatori né di un profilo di provisioning. Dovete comunque creare un certificato p12, ma può essere autofirmato.

Per impostazione predefinita, ADT avvia sempre il simulatore di iPhone. Per modificare il dispositivo di simulazione, procedete come segue:

- Usate il comando seguente per visualizzare i simulatori disponibili.

```
xcrun simctl list devices
```

L'output avrà un aspetto simile a quello illustrato di seguito.

```
== Devices ==
-iOS 10.0 -
iPhone 5 (F6378129-A67E-41EA-AAF9-D99810F6BCE8) (Shutdown)
iPhone 5s (5F640166-4110-4F6B-AC18-47BC61A47749) (Shutdown)
iPhone 6 (E2ED9D38-C73E-4FF2-A7DD-70C55A021000) (Shutdown)
iPhone 6 Plus (B4DE58C7-80EB-4454-909A-C38C4106C01B) (Shutdown)
iPhone 6s (9662CB8A-2E88-403E-AE50-01FB49E4662B) (Shutdown)
iPhone 6s Plus (BED503F3-E70C-47E1-BE1C-A2B7F6B7B63E) (Shutdown)
iPhone 7 (71880D88-74C5-4637-AC58-1F9DB43BA471) (Shutdown)
iPhone 7 Plus (2F411EA1-EE8B-486B-B495-EFC421E0A494) (Shutdown)
iPhone SE (DF52B451-ACA2-47FD-84D9-292707F9F0E3) (Shutdown)
iPad Retina (C4EF8741-3982-481F-87D4-700ACD0DA6E1) (Shutdown)
....
```

- Potete scegliere un simulatore specifico impostando la variabile di ambiente `AIR_IOS_SIMULATOR_DEVICE`, come indicato di seguito:

```
export AIR_IOS_SIMULATOR_DEVICE = 'iPad Retina'
```

Riavviate il processo dopo aver impostato la variabile di ambiente ed eseguite l'applicazione sul dispositivo di simulazione di vostra scelta.

Nota: quando usate ADT con iOS Simulator, dovete sempre includere l'opzione `-platformsdk`, specificando il percorso di iOS Simulator SDK.

Per eseguire un'applicazione in iOS Simulator:

- 1 Utilizzate il comando `adt -package` con `-target ipa-test-interpreter-simulator` oppure con `-target ipa-debug-interpreter-simulator`, come nell'esempio seguente:

```
adt -package
                                     -target ipa-test-interpreter-simulator
                                     -storetype pkcs12 -keystore Certificates.p12
                                     -storepass password
                                     myApp.ipa
                                     myApp-app.xml
                                     myApp.swf
                                     -platformsdk
/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator5.0.sdk
```

Nota: Le opzioni di firma non sono ora più richieste per i simulatori, pertanto qualsiasi valore può essere fornito in flag `-keystore` poiché non sarà rispettato da ADT.

- 2 Utilizzate il comando `-installApp` per installare l'applicazione in iOS Simulator, come nell'esempio seguente:

```
adt -installApp
    -platform ios
    -platformsdk
    /Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator5.0.sdk
    -device ios-simulator
    -package sample_ipa_name.ipa
```

3 Utilizzate il comando `adt -launchApp` per eseguire l'applicazione in iOS Simulator, come nell'esempio seguente:

Nota: Per impostazione predefinita, il comando `adt -launchApp` esegue l'applicazione nel simulatore per iPhone. Per eseguire l'applicazione in un simulatore per iPad, esportare la variabile di ambiente `AIR_IOS_SIMULATOR_DEVICE = "iPad"` e quindi utilizzare il comando `adt -launchApp`.

```
adt -launchApp
    -platform ios
    -platformsdk
    /Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator5.0.sdk
    -device ios-simulator
    -appid sample_ipa_name
```

Per testare un'estensione nativa in iOS Simulator, utilizzate il nome piattaforma `iPhone-x86` nel file `extension.xml` e specificate `library.a` (libreria statica) nell'elemento `nativeLibrary`, come nell'esempio seguente:

```
<extension xmlns="http://ns.adobe.com/air/extension/3.1">
  <id>com.cnative.extensions</id>
  <versionNumber>1</versionNumber>
  <platforms>
    <platform name="iPhone-x86">
      <applicationDeployment>
        <nativeLibrary>library.a</nativeLibrary>
        <initializer>TestNativeExtensionsInitializer </initializer>
        <finalizer>TestNativeExtensionsFinalizer </finalizer>
      </applicationDeployment>
    </platform>
  </platforms>
</extension>
```

Nota: quando testate un'estensione nativa in iOS Simulator, non utilizzate la libreria statica (file `.a`) compilata per il dispositivo, bensì quella compilata per il simulatore.

Istruzioni trace

Quando eseguite un'applicazione per dispositivi mobili sul desktop, l'output trace viene stampato sul debugger oppure nella finestra di terminale usata per avviare ADL. Quando invece eseguite l'applicazione su un dispositivo o un emulatore potete impostare una sessione di debug remota per la visualizzazione dell'output trace. È anche possibile, se tale operazione è supportata, visualizzare l'output trace utilizzando i tool di sviluppo software forniti dal produttore del dispositivo o del sistema operativo.

In ogni caso, i file SWF dell'applicazione devono essere compilati con l'opzione di debug abilitata affinché il runtime produca l'output delle eventuali istruzioni trace.

Istruzioni trace remote su Android

Quando eseguite un'applicazione su un dispositivo o emulatore Android, potete visualizzare l'output delle istruzioni trace nel registro di sistema di Android utilizzando l'utilità ADB (Android Debug Bridge) inclusa in Android SDK. Per visualizzare l'output dell'applicazione, eseguite il comando seguente da un prompt dei comandi o da una finestra di terminale sul computer di sviluppo:


```
tools/adb logcat air.MyApp:I *:S
```

MyApp è l'ID applicazione AIR della vostra applicazione. L'argomento `*:S` disattiva l'output in tutti gli altri processi. Per visualizzare le informazioni di sistema sull'applicazione oltre all'output trace, potete includere `ActivityManager` nella specifica di filtro logcat:

```
tools/adb logcat air.MyApp:I ActivityManager:I *:S
```

In questi esempi di comandi si presume che ADB sia eseguito dalla cartella di Android SDK o che la cartella `SDK` sia stata aggiunta alla variabile d'ambiente dei percorsi.

Nota: in AIR 2.6+, il tool ADB è incluso in AIR SDK (nella cartella `lib/android/bin`).

Istruzioni trace remote in iOS

Per visualizzare l'output delle istruzioni trace da un'applicazione eseguita su un dispositivo iOS, dovete definire una sessione di debug remota utilizzando FDB (Flash Debugger).

Altri argomenti presenti nell' Aiuto

[Android Debug Bridge: abilitare la registrazione logcat](#)

["Variabili d'ambiente dei percorsi"](#) a pagina 316

Connessione al debugger di Flash

Per eseguire il debug di un'applicazione eseguita su un dispositivo mobile, potete avviare il debugger Flash sul vostro computer di sviluppo e aprire una connessione con tale computer tramite la rete. Per abilitare il debug remoto, effettuate le operazioni seguenti:

- In Android, specificate l'autorizzazione `android.permission.INTERNET` nel descrittore dell'applicazione.
- Compilate i file SWF dell'applicazione con il debug abilitato.
- Create il pacchetto dell'applicazione con il flag `-target apk-debug`, per Android, o `-target ipa-debug`, per iOS, e con `-connect` (debug via wi-fi) o `-listen` (debug via USB).

Per il debug remoto via wi-fi, il dispositivo deve poter accedere alla porta TCP 7935 del computer che esegue il debugger Flash tramite indirizzo IP o nome di dominio completo. Per il debug remoto via USB, il dispositivo deve poter accedere alla porta TCP 7936 o alla porta specificata nel flag `-listen`.

Per iOS, potete anche specificare `-target ipa-debug-interpreter` o `-target ipa-debug-interpreter-simulator`.

Debug remoto con Flash Professional

Quando l'applicazione è pronta per il debug e le autorizzazioni sono state impostate nel descrittore dell'applicazione, procedete nel modo seguente:

- 1 Aprite la finestra di dialogo Impostazioni AIR for Android.
- 2 Nella scheda Distribuzione:
 - Selezionate "Debug dispositivo" come tipo di distribuzione.
 - Selezionate "Installa l'applicazione sul dispositivo Android connesso" in Dopo la pubblicazione.
 - Deselezionate "Avvia l'applicazione sul dispositivo Android connesso" in Dopo la pubblicazione.
 - Impostate il percorso di Android SDK, se necessario.
- 3 Fate clic su Pubblica.

A questo punto l'applicazione è installata e avviata sul dispositivo.

- 4 Chiudete la finestra di dialogo Impostazioni AIR for Android.
- 5 Selezionate Debug > Inizia sessione di debug remota > ActionScript 3 nel menu di Flash Professional.
Viene visualizzato il messaggio "Attesa connessione del lettore" nel pannello Output di Flash Professional.
- 6 Avviate l'applicazione sul dispositivo.
- 7 Immettete l'indirizzo IP o il nome host del computer che esegue il debugger Flash nella finestra della connessione di Adobe AIR, quindi fate clic su OK.

Debug remoto con FDB su una connessione di rete

Per eseguire il debug di un'applicazione eseguita su un dispositivo con il debugger Flash della riga di comando (FDB), eseguite innanzi tutto il debugger sul computer di sviluppo, quindi avviate l'applicazione sul dispositivo. Nella procedura seguente vengono utilizzati i tool AMXMLC, FDB e ADT per eseguire la compilazione, la creazione del pacchetto e il debug dell'applicazione sul dispositivo. Negli esempi si presuppone che venga utilizzato un ambiente combinato con Flex e AIR SDK e che la directory bin sia inclusa nella variabile d'ambiente dei percorsi. (Tale scenario viene ipotizzato unicamente ai fini degli esempi relativi ai comandi.)

- 1 Aprite una finestra di terminale o di prompt dei comandi e accedete alla directory che contiene il codice di origine dell'applicazione.

- 2 Compilate l'applicazione con amxmlc, abilitando il debug:

```
amxmlc -debug DebugExample.as
```

- 3 Create il pacchetto dell'applicazione usando il target apk-debug o ipa-debug:

Android

```
adobe adt -package -target apk-debug -connect -storetype  
pkcs12 -keystore ../../AndroidCert.p12 DebugExample.apk DebugExample-app.xml  
DebugExample.swf
```

iOS

```
adobe adt -package -target ipa-debug -connect -storetype  
pkcs12 -keystore ../../AppleDeveloperCert.p12 -provisioning-profile test.mobileprovision  
DebugExample.apk DebugExample-app.xml DebugExample.swf
```

Se utilizzate sempre lo stesso nome host o indirizzo IP per il debug, potete aggiungere tale valore dopo il flag -connect. L'applicazione tenterà di connettersi a quell'indirizzo IP o nome host automaticamente. In caso contrario, dovete immettere le informazioni sul dispositivo ogni volta che avviate il debug.

- 4 Installate l'applicazione.

In Android, potete usare il comando ADT -installApp:

```
adobe adt -installApp -platform android -package DebugExample.apk
```

In iOS, potete installare l'applicazione con il comando ADT -installApp oppure mediante iTunes.

- 5 In una seconda finestra di terminale o di comando, eseguite FDB:

```
adobe fdb
```

- 6 Nella finestra di FDB, digitate il comando run:

```
Adobe fdb (Flash Player Debugger) [build 14159]  
Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.  
(fdb) run  
Waiting for Player to connect
```

- 7 Avviate l'applicazione sul dispositivo.

- 8 Una volta avviata l'applicazione sul dispositivo o sull'emulatore, si apre la finestra di connessione di Adobe AIR. (Se avete specificato un nome host o indirizzo IP con l'opzione `-connect` quando avete creato il pacchetto dell'applicazione, verrà effettuato un tentativo di connessione automatica a tale indirizzo.) Immettete l'indirizzo appropriato e selezionate OK.

Per stabilire la connessione con il debugger in questa modalità, il dispositivo deve poter risolvere l'indirizzo o nome host e connettersi alla porta TCP 7935. La connessione di rete è indispensabile.

- 9 Quando il runtime remoto si connette al debugger, potete impostare dei punti di interruzione con il comando `FDB break` e quindi avviare l'esecuzione con il comando `continue`:

```
(fdb) run

                                     Waiting for Player to connect
                                     Player connected; session starting.
                                     Set breakpoints and then type 'continue' to resume the
session.

                                     [SWF]
Users:juser:Documents:FlashProjects:DebugExample:DebugExample.swf - 32,235 bytes after
decompression

(fdb) break clickHandler
Breakpoint 1 at 0x5993: file DebugExample.as, line 14
(fdb) continue
```

Debug remoto con FDB via USB

AIR 2.6 (Android) e AIR 3.3 (iOS)

Per eseguire il debug di un'applicazione tramite una connessione USB, potete creare il pacchetto dell'applicazione usando l'opzione `-listen` anziché `-connect`. Quando specificate l'opzione `-listen`, il runtime resta in attesa di una connessione in entrata dal debugger Flash (FDB) sulla porta TCP 7936 nel momento in cui avviate l'applicazione. Eseguite quindi FDB con l'opzione `-p` per avviare la connessione tramite FDB.

Procedura di debug via USB per Android

Affinché il debugger Flash in esecuzione sul computer desktop possa connettersi al runtime AIR in esecuzione sul dispositivo o l'emulatore, dovete usare l'utilità ADB (Android Debug Bridge), inclusa in Android SDK, oppure l'utilità IDB (iOS Debug Bridge), inclusa in AIR SDK, per instradare la porta del dispositivo alla porta del desktop.

- 1 Aprite una finestra di terminale o di prompt dei comandi e accedete alla directory che contiene il codice di origine dell'applicazione.

- 2 Compilate l'applicazione con `amxmlc`, abilitando il debug:

```
amxmlc -debug DebugExample.as
```

- 3 Create il pacchetto dell'applicazione usando il target di debug appropriate (ad esempio `apk-debug`) e specificate l'opzione `-listen`:

```
adt -package -target apk-debug -listen -storetype pkcs12 -keystore ../../AndroidCert.p12
DebugExample.apk DebugExample-app.xml DebugExample.swf
```

- 4 Connettete il dispositivo al computer di debug con un cavo USB. (Potete seguire questa procedura anche per eseguire il debug di un'applicazione in esecuzione su un emulatore, nel qual caso una connessione USB non è necessaria, né possibile.)

- 5 Installate l'applicazione.

Potete usare il comando ADT `-installApp`:

```
adt -installApp -platform android -package DebugExample.apk
```

- 6 Intradate la porta TCP 7936 dal dispositivo o emulator al computer desktop utilizzando l'utilità Android ADB:

```
adb forward tcp:7936 tcp:7936
```

- 7 Avviate l'applicazione sul dispositivo.

- 8 In una finestra di terminale o di comando, eseguite FDB con l'opzione -p:

```
fdb -p 7936
```

- 9 Nella finestra di FDB, digitate il comando run:

```
Adobe fdb (Flash Player Debugger) [build 14159]
Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
(fdb) run
```

- 10 L'utilità FDB tenta di connettersi all'applicazione.

- 11 Quando viene stabilita la connessione remota, potete impostare dei punti di interruzione con il comando FDB `break` e quindi avviare l'esecuzione con il comando `continue`:

```
(fdb) run
Player connected; session starting.
Set breakpoints and then type 'continue' to resume the
session.
[SWF]
Users:juser:Documents:FlashProjects:DebugExample:DebugExample.swf - 32,235 bytes after
decompression
(fdb) break clickHandler
Breakpoint 1 at 0x5993: file DebugExample.as, line 14
(fdb) continue
```

Nota: la porta numero 7936 viene utilizzata come porta predefinita per il debug via USB sia dal runtime AIR che da FDB. Potete specificare porte differenti con il parametro di porta ADT `-listen` e il parametro di porta FDB `-p`. In questo caso dovete usare l'utilità Android Debug Bridge per instradare il numero di porta specificato in ADT alla porta specificata in FDB: `adb forward tcp:adt_listen_port# tcp:fdb_port#`

Procedura di debug via USB per iOS

Affinché il debugger Flash in esecuzione sul computer desktop possa connettersi al runtime AIR in esecuzione sul dispositivo o l'emulatore, dovete usare l'utilità IDB (iOS Debug Bridge), inclusa in AIR SDK, per instradare la porta del dispositivo alla porta del desktop.

- 1 Aprite una finestra di terminale o di prompt dei comandi e accedete alla directory che contiene il codice di origine dell'applicazione.

- 2 Compilate l'applicazione con `amxmlc`, abilitando il debug:

```
amxmlc -debug DebugExample.as
```

- 3 Create il pacchetto dell'applicazione usando il target di debug appropriate (ad esempio `apk-debug` o `ipa-debug-interpretter`) e specificate l'opzione `-listen`:

```
adt -package -target ipa-debug-interpretter -listen 16000
xyz.mobileprovision -storetype pkcs12 -keystore
Certificates.p12
-storepass pass123 OutputFile.ipa InputFile-app.xml
InputFile.swf
```

- 4 Connettete il dispositivo al computer di debug con un cavo USB. (Potete seguire questa procedura anche per eseguire il debug di un'applicazione in esecuzione su un emulatore, nel qual caso una connessione USB non è necessaria, né possibile.)

5 Installate e avviate l'applicazione sul dispositivo iOS. In AIR 3.4 e versioni successive, potete usare `adt -installApp` per installare l'applicazione via USB.

6 Determinate l'handle del dispositivo con il comando `idb -devices` (IDB si trova in `air_sdk_root/lib/aot/bin/iOSBin/idb`):

```
./idb -devices  
  
List of attached devices  
Handle    UUID  
1         91770d8381d12644df91fbcee1c5bbdacb735500
```

Nota: (AIR 3.4 e versioni successive) Potete usare `adt -devices` anziché `idb -devices` per determinare l'handle del dispositivo.

7 Intradata una porta del desktop alla porta specificata nel parametro `adt -listen` (in questo caso, 16000; la porta predefinita è 7936) utilizzando l'utilità IDB e l'ID dispositivo trovato nel passaggio precedente:

```
idb -forward 7936 16000 1
```

In questo esempio, 7936 è la porta desktop, 16000 è la porta sulla quale è in ascolto il dispositivo connesso e 1 è l'ID dispositivo del dispositivo connesso.

8 In una finestra di terminale o di comando, eseguite FDB con l'opzione `-p`:

```
fdb -p 7936
```

9 Nella finestra di FDB, digitate il comando `run`:

```
Adobe fdb (Flash Player Debugger) [build 23201]  
Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.  
(fdb) run
```

10 L'utilità FDB tenta di connettersi all'applicazione.

11 Quando viene stabilita la connessione remota, potete impostare dei punti di interruzione con il comando `FDB break` e quindi avviare l'esecuzione con il comando `continue`:

Nota: la porta numero 7936 viene utilizzata come porta predefinita per il debug via USB sia dal runtime AIR che da FDB. Potete specificare porte differenti con il parametro di porta IDB `-listen` e il parametro di porta FDB `-p`.

Installazione di AIR e di applicazioni AIR sui dispositivi mobili

Gli utenti finali dell'applicazione possono installare il runtime AIR e le applicazioni AIR utilizzando il normale meccanismo di installazione e distribuzione delle applicazioni del proprio dispositivo.

In Android, ad esempio, gli utenti possono installare le applicazioni da Android Market. Oppure, se sono autorizzati a installare applicazioni da fonti sconosciute (nelle impostazioni Applicazione), possono installare un'applicazione facendo clic su un collegamento in una pagina Web o copiando il pacchetto dell'applicazione sul dispositivo e quindi aprendolo. Se un utente tenta di installare un'applicazione Android ma non ha ancora installato il runtime AIR, verrà automaticamente indirizzato ad Android Market, dove potrà installare il runtime.

iOS prevede due modi per distribuire le applicazioni agli utenti finali. Il canale di distribuzione principale è Apple App Store. In alternativa, potete usare la distribuzione ad hoc per consenti a un numero ristretto di utenti di installare l'applicazione senza dover accedere ad App Store.

Installare il runtime AIR e le applicazioni per fini di sviluppo

Poiché sui dispositivi mobili le applicazioni AIR vengono installate come pacchetti nativi, potete usare le normali funzionalità di installazione applicazioni della piattaforma per svolgere le attività di test. Se sono supportati, potete usare i comandi ADT per installare il runtime AIR e le applicazioni AIR. Attualmente questo tipo di approccio è supportato solo in Android.

In iOS, potete utilizzare iTunes per installare le applicazioni da testare. Le applicazioni di prova devono essere firmate con un certificato di firma del codice Apple emesso specificamente per lo sviluppo di applicazioni; inoltre, il pacchetto dell'applicazione deve essere creato con un profilo di provisioning di sviluppo. Un'applicazione AIR è un pacchetto autonomo in iOS. Non viene utilizzato un runtime separato.

Installazione di applicazioni AIR mediante ADT

Quando sviluppate applicazioni AIR, potete utilizzare ADT per installare e disinstallare sia il runtime che le applicazioni. (Il vostro ambiente IDE potrebbe anche già includere questi comandi, nel qual caso non dovete eseguire ADT manualmente.)

Potete installare il runtime AIR su un dispositivo o un emulatore utilizzando l'utilità AIR ADT. Deve essere installato anche il kit SDK fornito per il dispositivo. Utilizzate il comando `-installRuntime`:

```
adt -installRuntime -platform android -device deviceID -package path-to-runtime
```

Se non viene specificato il parametro `-package`, viene scelto il pacchetto runtime appropriato per il dispositivo o l'emulatore tra quelli disponibili nel kit AIR SDK installato.

Per installare un'applicazione AIR in Android o iOS (AIR 3.4 e versioni successive), utilizzate il comando analogo `-installApp`:

```
adt -installApp -platform android -device deviceID -package path-to-app
```

Il valore impostato per l'argomento `-platform` deve corrispondere al dispositivo sul quale effettuate l'installazione.

Nota: eventuali versioni già installate del runtime AIR o dell'applicazione AIR devono essere rimosse prima della reinstallazione.

Installazione di applicazioni AIR sui dispositivi mobili con iTunes

Per installare un'applicazione AIR su un dispositivo iOS a scopo di test:

- 1 Aprite l'applicazione iTunes.
- 2 Se non l'avete già fatto, aggiungete il profilo di provisioning per questa applicazione a iTunes. In iTunes, selezionate File > Aggiungi alla Libreria, quindi scegliete il file del profilo di provisioning (il cui tipo è `mobileprovision`).
- 3 Alcune versioni di iTunes non sostituiscono l'applicazione se la stessa versione dell'applicazione è già installata. In questo caso, cancellate l'applicazione dal dispositivo e dall'elenco delle applicazioni in iTunes.
- 4 Fate doppio clic sul file IPA per l'applicazione. Il file dovrebbe essere visualizzato nell'elenco delle applicazioni in iTunes.
- 5 Collegate il dispositivo alla porta USB del computer.
- 6 In iTunes, controllate la scheda Applicazione per il dispositivo e verificate che l'applicazione sia selezionata nell'elenco delle applicazioni da installare.
- 7 Selezionate il dispositivo nell'elenco a sinistra dell'applicazione iTunes. Quindi fate clic sul pulsante Sincronizza. Al termine della sincronizzazione, l'applicazione Hello World viene visualizzata sull'iPhone.

Se la nuova versione non è installata, eliminatela dal dispositivo e dall'elenco delle applicazioni in iTunes, quindi ripetete questa procedura. Questa situazione può verificarsi se la versione installata utilizza lo stesso ID applicazione e la stessa versione.

Altri argomenti presenti nell'Aiuto

[“Comando ADT installRuntime”](#) a pagina 183

[“Comando ADT installApp”](#) a pagina 180

Esecuzione di applicazioni AIR su un dispositivo

Potete avviare le applicazioni AIR utilizzando l'interfaccia utente del dispositivo. Se questa opzione è supportata, potete anche avviare le applicazioni in modalità remota mediante l'utilità AIR ADT:

```
adt -launchApp -platform android -device deviceID -appid applicationID
```

Il valore dell'argomento `-appid` deve corrispondere all'ID dell'applicazione AIR da avviare. Usate il valore specificato nel descrittore dell'applicazione AIR (senza il prefisso *air.* aggiunto durante la creazione del pacchetto).

Se un solo dispositivo o emulatore è collegato e in esecuzione, potete omettere il flag `-device`. Il valore impostato per l'argomento `-platform` deve corrispondere al dispositivo sul quale effettuate l'installazione. Attualmente, l'unico valore supportato è *android*.

Rimozione del runtime AIR e delle applicazioni

Per rimuovere le applicazioni, potete utilizzare le normali modalità previste dal sistema operativo. Se è supportata, potete anche usare l'utilità AIR ADT per eliminare il runtime AIR e le applicazioni. Per rimuovere il runtime, usate il comando `-uninstallRuntime`:

```
adt -uninstallRuntime -platform android -device deviceID
```

Per disinstallare un'applicazione, utilizzate il comando `-uninstallApp`:

```
adt -uninstallApp -platform android -device deviceID -appid applicationID
```

Se un solo dispositivo o emulatore è collegato e in esecuzione, potete omettere il flag `-device`. Il valore impostato per l'argomento `-platform` deve corrispondere al dispositivo sul quale effettuate l'installazione. Attualmente, l'unico valore supportato è *android*.

Configurazione di un emulatore

Per eseguire l'applicazione AIR su un emulatore di dispositivo, in genere occorre utilizzare il kit SDK del dispositivo per creare ed eseguire un'istanza dell'emulatore sul computer di sviluppo. Quindi potete installare la versione per emulatore del runtime AIR e l'applicazione AIR sull'emulatore. Tenete presente che le applicazioni eseguite su un emulatore sono generalmente più lente rispetto a quanto avviene sul dispositivo reale.

Creare un emulatore Android

- 1 Avviate Android SDK e l'applicazione AVD Manager:
 - In Windows, eseguite il file Setup.exe del kit SDK, al livello principale della directory Android SDK.
 - In Mac OS, eseguite l'applicazione android, nella sottodirectory tools della directory Android SDK.
- 2 Selezionate l'opzione Settings, quindi selezionate l'opzione "Force https://".
- 3 Selezionate l'opzione Available Packages. Dovrebbe apparire un elenco di SDK Android disponibili.

- 4 Selezionate un Android SDK compatibile (Android 2.3 o successivo) e fate clic sul pulsante Install Selected.
- 5 Selezionate l'opzione Virtual Devices e fate clic sul pulsante New.
- 6 Effettuate le seguenti impostazioni:
 - Un nome per il dispositivo virtuale
 - L'API di destinazione, ad esempio Android 2.3, API level 8
 - Una dimensione per la scheda SD (ad esempio 1024)
 - Uno skin (ad esempio Default HVGA)
- 7 Fate clic sul pulsante Create AVD.

La creazione del dispositivo virtuale potrebbe richiedere un po' di tempo, a seconda della configurazione del sistema.

Ora potete avviare il nuovo dispositivo virtuale.

- 1 Selezionate Virtuale Device nell'applicazione AVD Manager. Il dispositivo virtuale precedentemente creato dovrebbe comparire nell'elenco.
- 2 Selezionate il dispositivo virtuale e fate clic sul pulsante Start.
- 3 Fate clic sul pulsante Launch nella schermata successiva.

Sul desktop dovrebbe apparire una finestra di emulatore. Potrebbe essere necessario attendere qualche secondo. Anche l'inizializzazione del sistema operativo Android potrebbe richiedere qualche istante. In un emulatore potete installare le applicazioni il cui pacchetto è stato creato con il target *apk-debug* o *apk-emulator*, mentre le applicazioni create con il target *apk* non funzionano in un emulatore.

Altri argomenti presenti nell'Aiuto

<http://developer.android.com/guide/developing/tools/othertools.html#android>

<http://developer.android.com/guide/developing/tools/emulator.html>

Aggiornamento di applicazioni AIR per dispositivi mobili

Le applicazioni AIR per dispositivi mobili vengono distribuite come pacchetti nativi e di conseguenza utilizzano gli stessi meccanismi di aggiornamento standard delle altre applicazioni della piattaforma. In genere, ciò comporta l'invio dell'applicazione allo stesso spazio commerciale (Android Market, Apple App Store, ecc.) utilizzato per distribuire l'applicazione originale.

Le applicazioni AIR per dispositivi mobili non possono usare il framework o la classe AIR Updater.

Aggiornamento di applicazioni AIR in Android

Nel caso di applicazioni distribuite su Android Market, potete aggiornare un'applicazione inserendo una nuova versione in Android Market, a condizione che siano rispettate tutte le seguenti condizioni (imposte da Android Market, non da AIR):

- Il pacchetto APK è firmato con lo stesso certificato.
- L'ID AIR è lo stesso.

- Il valore `versionNumber` nel descrittore dell'applicazione è maggiore. (Dovete anche incrementare il valore `versionLabel`, se utilizzato.)

Gli utenti che hanno scaricato l'applicazione da Android Market ricevono una notifica dal software del dispositivo che segnala la disponibilità dell'aggiornamento.

Altri argomenti presenti nell' Aiuto

[Sviluppatori Android: pubblicazione di aggiornamenti su Android Market](#)

Aggiornamento di applicazioni AIR in iOS

Nel caso di applicazioni distribuite su iTunes App Store, potete aggiornare un'applicazione inviando l'aggiornamento all'App Store, a condizione che siano rispettate tutte le seguenti condizioni (imposte da Apple App Store, non da AIR):

- Il certificato di firma del codice e i profili di provisioning devono essere emessi per lo stesso Apple ID.
- Il pacchetto IPA deve usare lo stesso Apple Bundle ID.
- L'aggiornamento non restringe la gamma di dispositivi supportati (in altre parole, se l'applicazione originale supporta dispositivi con sistema operativo iOS 3, non potete creare un aggiornamento che non supporta iOS 3).

Importante: poiché AIR 2.6 (e versioni successive) non supporta iOS 3, al contrario di AIR 2, non potete aggiornare applicazioni iOS pubblicate che sono state sviluppate usando AIR 2 con una versione sviluppata con AIR 2.6+.

Uso delle notifiche push

Le notifiche push consentono a provider di notifiche remote di inviare notifiche ad applicazioni eseguite su dispositivi mobili. AIR 3.4 supporta le notifiche push per dispositivi iOS che impiegano il servizio APN (Apple Push Notification).

Nota: per abilitare le notifiche push in un'applicazione AIR for Android, utilizzate un'estensione nativa, quale [as3c2dm](#), sviluppata dall'esperto Adobe Piotr Walczyszyn.

La restante parte di questa sezione spiega come abilitare le notifiche push in applicazioni AIR for iOS.

Nota: questa discussione presuppone che disponiate di un ID di sviluppatore Apple, che abbiate familiarità con il flusso di lavoro di sviluppo iOS e che abbiate implementato almeno un'applicazione su un dispositivo iOS.

Panoramica delle notifiche push

Il servizio APN (Apple Push Notification service) consente a provider di notifiche remote di inviare notifiche ad applicazioni eseguite su dispositivi iOS. Il servizio APN supporta i seguenti tipi di notifiche:

- Avvisi
- Badge
- Suoni

Per informazioni complete sul servizio APN, andate alla pagina developer.apple.com.

L'uso delle notifiche push in un'applicazione comporta una serie di attività correlate:

- **Applicazione client** - Registrazione alle notifiche push, comunicazione con i provider di notifiche remote e ricezione delle notifiche push.
- **iOS** - Gestione dell'interazione tra applicazione client e servizio APN.

- **Servizio APN** - Fornitura di un tokenID durante la registrazione del client e trasferimento delle notifiche dai provider di notifiche remote a iOS.
- **Provider di notifiche remote** - Memorizzazione delle informazioni relative a tokenID e applicazione client e invio delle notifiche push al servizio APN.

Flusso di lavoro di registrazione

Il flusso di lavoro per la registrazione delle notifiche push su un servizio lato server è il seguente:

- 1 L'applicazione client richiede a iOS di abilitare le notifiche push.
- 2 iOS inoltra la richiesta al servizio APN.
- 3 Il server APN restituisce un tokenId a iOS.
- 4 iOS invia il tokenId all'applicazione client.
- 5 L'applicazione client (utilizzando un meccanismo specifico dell'applicazione) fornisce il tokenId al provider di notifiche remote, che lo memorizza per utilizzarlo per l'invio delle notifiche push.

Flusso di lavoro di notifica

Il flusso di lavoro di notifica è il seguente:

- 1 Il provider di notifiche remoto genera una notifica e trasmette il payload della notifica al servizio APN, insieme al tokenId.
- 2 Il servizio APN inoltra la notifica a iOS sul dispositivo.
- 3 iOS trasmette il payload della notifica all'applicazione.

API delle notifiche push

In AIR 3.4 è stata introdotta una serie di API che supportano le notifiche push iOS. Queste API si trovano nel pacchetto `flash.notifications` e includono le classi seguenti:

- `NotificationStyle` - Definisce le costanti per i tipi di notifiche: `ALERT`, `BADGE` e `SOUND`.
- `RemoteNotifier` - Consente di sottoscrivere le notifiche push e di annullare la sottoscrizione.
- `RemoteNotifierSubscribeOptions` - Consente di scegliere i tipi di notifiche che si desidera ricevere. Utilizzate la proprietà `notificationStyles` per definire un vettore di stringhe che consenta di registrarsi a più tipi di notifiche.

AIR 3.4 include anche la proprietà `flash.events.RemoteNotificationEvent`, che viene inviata da `RemoteNotifier` nei seguenti casi:

- Quando la sottoscrizione di un'applicazione va a buon fine e un nuovo tokenId viene ricevuto dal servizio APN.
- Al ricevimento di una nuova notifica remota.

Inoltre, `RemoteNotifier` trasmette `flash.events.StatusEvent` se riscontra un errore durante la procedura di sottoscrizione.

Gestione delle notifiche push in un'applicazione

Per registrare un'applicazione per le notifiche push, dovete procedere nel modo seguente:

- Create un codice per la sottoscrizione alle notifiche push nella vostra applicazione.
- Abilitate le notifiche push nel file XML dell'applicazione.

- Create un profilo di provisioning e un certificato che abiliti i servizi push di iOS.

Il codice di esempio riportato di seguito consente di sottoscrivere le notifiche push e di gestire gli eventi di notifica push:

```
package

    {
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.*;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.MouseEvent;
    import flash.net.*;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.ui.Multitouch;
    import flash.ui.MultitouchInputMode;
    // Required packages for push notifications
    import flash.notifications.NotificationStyle;
    import flash.notifications.RemoteNotifier;
    import flash.notifications.RemoteNotifierSubscribeOptions;
    import flash.events.RemoteNotificationEvent;
    import flash.events.StatusEvent;
    [SWF(width="1280", height="752", frameRate="60")]
    public class TestPushNotifications extends Sprite
    {
    private var notiStyles:Vector.<String> = new Vector.<String>;;
    private var tt:TextField = new TextField();
    private var tf:TextFormat = new TextFormat();
    // Contains the notification styles that your app wants to receive
    private var preferredStyles:Vector.<String> = new Vector.<String>();
    private var subscribeOptions:RemoteNotifierSubscribeOptions = new
RemoteNotifierSubscribeOptions();
    private var remoteNot:RemoteNotifier = new RemoteNotifier();
    private var subsButton:CustomButton = new CustomButton("Subscribe");
    private var unSubsButton:CustomButton = new
CustomButton("UnSubscribe");

    private var clearButton:CustomButton = new CustomButton("clearText");
    private var urlreq:URLRequest;
    private var urlLoad:URLLoader = new URLLoader();
    private var urlString:String;
    public function TestPushNotifications()
    {
    super();
    Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
    stage.align = StageAlign.TOP_LEFT;
    stage.scaleMode = StageScaleMode.NO_SCALE;
    tf.size = 20;
    tf.bold = true;
    tt.x=0;
    tt.y =150;
    tt.height = stage.stageHeight;
    tt.width = stage.stageWidth;
    tt.border = true;
    tt.defaultTextFormat = tf;
    addChild(tt);
```

```
        subsButton.x = 150;
        subsButton.y=10;
        subsButton.addEventListener(MouseEvent.CLICK,subsButtonHandler);
        stage.addChild(subsButton);
        unSubsButton.x = 300;
        unSubsButton.y=10;
        unSubsButton.addEventListener(MouseEvent.CLICK,unSubsButtonHandler);
        stage.addChild(unSubsButton);
        clearButton.x = 450;
        clearButton.y=10;
        clearButton.addEventListener(MouseEvent.CLICK,clearButtonHandler);
        stage.addChild(clearButton);
        //
        tt.text += "\n SupportedNotification Styles: " +
RemoteNotifier.supportedNotificationStyles.toString() + "\n";
        tt.text += "\n Before Preferred notificationStyles: " +
subscribeOptions.notificationStyles.toString() + "\n";
        // Subscribe to all three styles of push notifications:
        // ALERT, BADGE, and SOUND.
        preferredStyles.push(NotificationStyle.ALERT
,NotificationStyle.BADGE,NotificationStyle.SOUND );
        subscribeOptions.notificationStyles= preferredStyles;
        tt.text += "\n After Preferred notificationStyles:" +
subscribeOptions.notificationStyles.toString() + "\n";

remoteNot.addEventListener(RemoteNotificationEvent.TOKEN,tokenHandler);

remoteNot.addEventListener(RemoteNotificationEvent.NOTIFICATION,notificationHandler);
        remoteNot.addEventListener(StatusEvent.STATUS,statusHandler);
        this.stage.addEventListener(Event.ACTIVATE,activateHandler);
    }
    // Apple recommends that each time an app activates, it subscribe for
    // push notifications.
    public function activateHandler(e:Event):void{
    // Before subscribing to push notifications, ensure the device supports
it.

        // supportedNotificationStyles returns the types of notifications
        // that the OS platform supports
        if(RemoteNotifier.supportedNotificationStyles.toString() != "")
        {
        remoteNot.subscribe(subscribeOptions);
        }
        else{
        tt.appendText("\n Remote Notifications not supported on this Platform
!");

        }
        }
        public function subsButtonHandler(e:MouseEvent):void{
        remoteNot.subscribe(subscribeOptions);
        }
        // Optionally unsubscribe from push notifications at runtime.
        public function unSubsButtonHandler(e:MouseEvent):void{
        remoteNot.unsubscribe();
        tt.text += "\n UNSUBSCRIBED";
        }
        public function clearButtonHandler(e:MouseEvent):void{
        tt.text = " ";
    }
```

```
    }
    // Receive notification payload data and use it in your app
    public function notificationHandler(e:RemoteNotificationEvent):void{
    tt.appendText("\nRemoteNotificationEvent type: " + e.type +
    "\nbubbles: " + e.bubbles + "\ncancelable " +e.cancelable);
    for (var x:String in e.data) {
    tt.text += "\n"+ x + ": " + e.data[x];
    }
    }
    // If the subscribe() request succeeds, a RemoteNotificationEvent of
    // type TOKEN is received, from which you retrieve e.tokenId,
    // which you use to register with the server provider (urbanairship, in
    // this example.
    public function tokenHandler(e:RemoteNotificationEvent):void
    {
    tt.appendText("\nRemoteNotificationEvent type: "+e.type +"\nBubbles:
    "+ e.bubbles + "\ncancelable " +e.cancelable +"\ntokenID:\n"+ e.tokenId +"\n");
    urlString = new
    String("https://go.urbanairship.com/api/device_tokens/" +
    e.tokenId);
    urlreq = new URLRequest(urlString);
    urlreq.authenticate = true;
    urlreq.method = URLRequestMethod.PUT;
    URLRequestDefaults.setLoginCredentialsForHost
    ("go.urbanairship.com",
    "1ssB2iV_RL6_UBLiYMQVfg", "t-kZlzXGQ6-yU8T3iHiSyQ");
    urlLoad.load(urlreq);
    urlLoad.addEventListener(IOErrorEvent.IO_ERROR,iohandler);
    urlLoad.addEventListener(Event.COMPLETE,compHandler);
    urlLoad.addEventListener(HTTPStatusEvent.HTTP_STATUS,httpHandler);
    }
    private function iohandler(e:IOErrorEvent):void
    {
    tt.appendText("\n In IOError handler" + e.errorID + " " +e.type);
    }
    private function compHandler(e:Event):void{
    tt.appendText("\n In Complete handler, "+"status: " +e.type + "\n");
    }
    private function httpHandler(e:HTTPStatusEvent):void{
    tt.appendText("\n in httpstatus handler, "+"Status: " + e.status);
    }
    // If the subscription request fails, StatusEvent is dispatched with
    // error level and code.
    public function statusHandler(e:StatusEvent):void{
    tt.appendText("\n statusHandler");
    tt.appendText("event Level" + e.level +"\nevent code " +
    e.code + "\ne.currentTarget: " + e.currentTarget.toString());
    }
    }
    }
```

Abilitazione delle notifiche push nel file XML dell'applicazione

Per poter utilizzare le notifiche push nella vostra applicazione, dovete specificare quanto segue nel tag `Entitlements` (sotto il tag `iphone`):

```
<iphone>
    ...
    <Entitlements>
    <![CDATA[
        <key>aps-environment</key>
        <string>development</string>
    ]]>
    </Entitlements>
</iphone>
```

Quando siete pronti a eseguire il push dell'applicazione all'App Store, un elemento `<string>` per lo sviluppo alla produzione:

```
<string>production</string>
```

Se la vostra applicazione supporta stringhe localizzate, specificate le lingue nel tag `supportedLanguages`, sotto al tag `initialWindow`, come nell'esempio seguente:

```
<supportedLanguages>en de cs es fr it ja ko nl pl pt</supportedLanguages>
```

Creazione di un profilo di provisioning e di un certificato per l'abilitazione dei servizi push di iOS

Per abilitare le comunicazioni tra l'applicazione e il servizio APN, dovete inserire nell'applicazione un profilo di provisioning e un certificato che abiliti i servizi push di iOS. Per farlo, procedete nel modo seguente:

- 1 Accedete al vostro account di sviluppatore Apple.
- 2 Navigate fino al Provisioning Portal.
- 3 Fate clic sulla scheda App IDs.
- 4 Fate clic sul pulsante New App ID.
- 5 Specificate una descrizione e un identificatore pacchetto (non utilizzate l'asterisco (*) nell'identificatore pacchetto).
- 6 Fate clic su Invia. Il Provisioning Portal genera l'ID applicazione e visualizza di nuovo la pagina App IDs.
- 7 Fate clic su Configure, alla destra dell'ID applicazione. Viene visualizzata la pagina Configure App ID.
- 8 Selezionate la casella di controllo Enable per il servizio APN. Noterete che vi sono due tipi di certificati SSL push: uno per lo sviluppo/test e un altro per la produzione.
- 9 Fate clic sul pulsante Configure a destra di Development Push SSL Certificate. Viene visualizzata la pagina Generate Certificate Signing Request (CSR).
- 10 Generate una CSR utilizzando l'utility Keychain Access, come spiegato nella pagina.
- 11 Generate il certificato SSL.
- 12 Scaricate e installate il certificato SSL.
- 13 (Facoltativo) Ripetete i passaggi da 9 a 12 per il certificato SSL push per la produzione.
- 14 Fate clic su Done. Viene visualizzata la pagina Configure App ID.
- 15 Fate clic su Done. Viene visualizzata la pagina App IDs. Noterete il cerchio verde accanto a Push Notification per il vostro ID applicazione.
- 16 Assicuratevi di salvare i vostri certificati SSL, in quanto vi serviranno successivamente per abilitare le comunicazioni tra l'applicazione e il provider.
- 17 Fate clic sulla scheda Provisioning per visualizzare la pagina Provisioning Profiles.
- 18 Create un profilo di provisioning per il vostro nuovo ID applicazione e scaricatelo.

19 Fate clic sulla scheda Certificates e scaricate un nuovo certificato per il nuovo profilo di provisioning.

Uso dei suoni nelle notifiche push

Per abilitare le notifiche sonore nella vostra applicazione, dovete inserire i file audio come qualsiasi altra risorsa, ma nella stessa directory dei file SWF e app-xml. Ad esempio:

```
Build/adt -package -target ipa-app-store -provisioning-profile _-.mobileprovision -storetype pkcs12 -keystore _-.p12 test.ipa test-app.xml test.swf sound.caf sound1.caf
```

Apple supporta i seguenti formati di dati audio (su file aiff, wav o caf):

- PCM lineare
- MA4 (IMA/ADPCM)
- uLaw
- aLaw

Uso di notifiche di avviso localizzate

Per utilizzare le notifiche di avviso localizzate nella vostra applicazione, inserite stringhe localizzate sotto forma di cartelle lproj. Ad esempio, per supportare avvisi in spagnolo, procedete nel modo seguente:

- 1 Create una cartella es.lproj all'interno del progetto, allo stesso livello del file app-xml.
- 2 All'interno della cartella es.lproj, create un file di testo denominato Localizable.Strings.
- 3 Aprite Localizable.Strings in un editor di testo e aggiungete le chiavi dei messaggi e le stringhe localizzate corrispondenti. Ad esempio:

```
"PokeMessageFormat" = "La notificación de alertas en español."
```
- 4 Salvate il file.
- 5 Quando l'applicazione riceve una notifica di avviso con questo valore chiave e la lingua del dispositivo è lo spagnolo, viene visualizzato il testo dell'avviso tradotto.

Configurazione di un provider di notifiche remote

È necessario un provider di notifiche remote che invii le notifiche push alla vostra applicazione. Questa applicazione server funge da provider, accettando i vostri input di push e trasferendo la notifica con i relativi dati al servizio APN, che, a sua volta, invia la notifica push a un'applicazione client.

Per informazioni dettagliate sull'invio di notifiche push da un provider di notifiche remote, consultate l'articolo [Provider Communication with Apple Push Notification Service](#) della libreria per sviluppatori Apple.

Opzioni per provider di notifiche remote

Le opzioni disponibili per la configurazione di un provider di notifiche remote sono le seguenti:

- Create un vostro provider, basato sul server open-source APNS-php. Per impostare un server PHP potete utilizzare <http://code.google.com/p/apns-php/>. Questo progetto Google Code consente di progettare un'interfaccia in grado di soddisfare i vostri specifici requisiti.
- Usate un provider di servizi. Ad esempio, <http://urbanairship.com/> offre un provider di servizi APN pronto per l'uso. Dopo aver eseguito la registrazione al servizio, dovete iniziare a fornire il token del vostro dispositivo utilizzando codice simile al seguente:

```
private var urlreq:URLRequest;

private var urlLoad:URLLoader = new URLLoader();
private var urlString:String;
//When subscription is successful then only call the

following code

urlString = new
String("https://go.urbanairship.com/api/device_tokens/" + e.tokenId);
urlreq = new URLRequest(urlString);
urlreq.authenticate = true;
urlreq.method = URLRequestMethod.PUT;

URLRequestDefaults.setLoginCredentialsForHost("go.urbanairship.com",
    "Application Key", "Application Secret");
urlLoad.load(urlreq);

urlLoad.addEventListener(IOErrorEvent.IO_ERROR, iohandler);
urlLoad.addEventListener(Event.COMPLETE, compHandler);

urlLoad.addEventListener(HTTPStatusEvent.HTTP_STATUS, httpHandler);
private function iohandler(e:IOErrorEvent):void{
    trace("\n In IOError handler" + e.errorID + " "
+e.type);
}
private function compHandler(e:Event):void{
    trace("\n In Complete handler,"+"status: " +e.type +
"\n");
}
private function httpHandler(e:HTTPStatusEvent):void{
    tt.appendText("\n in httpstatus handler,"+ "Status:
" + e.status);
}
```

Per inviare notifiche di prova potete usare gli strumenti Urban Airship.

Certificati per il provider di notifiche remote

Dovete copiare il certificato SSL e il codice privato (precedentemente generato) nel percorso appropriato sul server del provider di notifiche remote. In genere, potete unire questi due file in un unico file .pem. Per farlo, procedete nel modo seguente:

- 1 Aprite una finestra di terminale.
- 2 Create un file .pem dal certificato SSL digitando il seguente comando:

```
openssl x509 -in aps_developer_identity.cer -inform der -out TestPushDev.pem
```
- 3 Create un file .pem a partire dalla chiave privata (.p12) digitando il seguente comando:

```
openssl pkcs12 -nocerts -out TestPushPrivateKey.pem -in certificates.p12
```
- 4 Unite i due file .pem in un unico file digitando il comando seguente:

```
cat TestPushDev.pem TestPushPrivateKey.pem > FinalTestPush.pem
```
- 5 Fornite il file .pem creato al provider durante la creazione dell'applicazione push lato server.

Per maggiori informazioni, consultate l'articolo [Installing the SSL Certificate and Key on the Server](#) della guida Local and Push Notification Programming Guide di Apple.

Gestione delle notifiche push in un'applicazione

La gestione delle notifiche push in un'applicazione comporta quanto segue:

- Configurazione e accettazione delle notifiche push da parte degli utenti a livello globale
- Accettazione da parte degli utenti delle singole notifiche push
- Gestione delle notifiche push e dei dati di payload delle notifiche

Configurazione e accettazione delle notifiche push

La prima volta che un utente avvia un'applicazione abilitata per le notifiche push, iOS visualizza la finestra di dialogo **nome app desidera inviarti notifiche push**, con i pulsanti Non consentire e OK. Se l'utente sceglie OK, l'applicazione potrà ricevere tutti i tipi di notifiche che ha sottoscritto. Se l'utente seleziona Non consentire, l'applicazione non riceverà alcuna notifica.

***Nota:** gli utenti possono inoltre selezionare Impostazioni > Notifiche per controllare i tipi specifici di notifiche che desiderano ricevere per ciascuna applicazione abilitata per le notifiche push.*

Apple raccomanda di sottoscrivere le notifiche push per ogni applicazione che viene attivata. Quando la vostra applicazione richiama `RemoteNotifier.subscribe()`, essa riceve un `RemoteNotificationEvent` di tipo `token` contenente un `tokenId` numerico a 32 byte che identifica in modo univoco tale applicazione su un determinato dispositivo.

Ogni volta che il dispositivo riceve una notifica push, viene visualizzata una finestra a comparsa con i pulsanti Chiudi e Avvia. Se l'utente tocca Chiudi, non accade nulla; se invece seleziona Avvia, iOS richiama l'applicazione, che riceverà un `flash.events.RemoteNotificationEvent` di tipo `notification`, come descritto di seguito.

Gestione delle notifiche push e dei dati di payload

Quando il provider di notifiche remote invia una notifica a un dispositivo (usando il `tokenID`), la vostra applicazione riceverà un `flash.events.RemoteNotificationEvent` di tipo `notification`, indipendentemente dal fatto che l'applicazione sia o meno in esecuzione. A questo punto, l'applicazione eseguirà un'elaborazione della notifica specifica dell'applicazione. Se l'applicazione gestisce i dati delle notifiche, potrete accedervi mediante la proprietà `RemoteNotificationEvent.data` formattata con JSON.

Capitolo 8: Sviluppo di applicazioni AIR per dispositivi TV

Funzionalità AIR per dispositivi TV

Potete creare applicazioni Adobe® AIR® per dispositivi TV, quali televisori, videoregistratori digitali e lettori Blu-ray, se il dispositivo contiene AIR per TV. AIR per TV è ottimizzato per i dispositivi TV, in modo da utilizzare, ad esempio, gli acceleratori hardware di un dispositivo per il video e la grafica ad alte prestazioni.

Le applicazioni AIR per dispositivi TV sono applicazioni basate su SWF, non su HTML. Un'applicazione AIR per TV può sfruttare l'accelerazione hardware così come altre funzionalità AIR indicate per un ambiente domestico quale ad esempio il soggiorno di un appartamento.

Profili dispositivo

AIR usa i profili per definire un set di dispositivi target con funzionalità analoghe. Usate i seguenti profili per le applicazioni AIR per TV:

- Il profilo `tv`. Usate questo profilo nelle applicazioni AIR destinate a un dispositivo AIR per TV.
- Il profilo `extendedTV`. Usate questo profilo se l'applicazione AIR per TV utilizza estensioni native.

Le funzioni ActionScript definite per questi profili sono descritte nella sezione “[Profili dispositivo](#)” a pagina 254. Le specificità di ActionScript relative alle applicazioni AIR per TV sono trattate nella [Guida di riferimento di ActionScript 3.0 per la piattaforma Adobe Flash](#).

Per informazioni dettagliate sui profili AIR per TV, vedete “[Profili supportati](#)” a pagina 146.

Accelerazione hardware

I dispositivi TV contengono acceleratori hardware che consentono di incrementare sensibilmente le prestazioni grafiche e video dell'applicazione AIR. Per sfruttare questi acceleratori hardware, vedete “[Considerazioni sulla progettazione di applicazioni AIR per TV](#)” a pagina 127.

Protezione del contenuto

AIR per TV permette di creare esperienze utente articolate basate su contenuti video di valore, quali film di successo di Hollywood oppure pellicole indipendenti ed episodi di serie TV. I fornitori di contenuti hanno la possibilità di creare applicazioni interattive sfruttando gli strumenti Adobe. Possono infatti integrare i prodotti Adobe Server nella propria infrastruttura di distribuzione dei contenuti oppure collaborare con i partner di ecosistema Adobe.

La protezione del contenuto è un requisito fondamentale per la distribuzione di video a valore aggiunto. AIR per TV supporta Adobe® Flash® Access™, una soluzione per la protezione e la monetizzazione del contenuto che rispetta tutte le più rigide esigenze di sicurezza dei titolari dei contenuti, comprese le major del cinema.

Flash Access supporta:

- Streaming e download video.
- Vari modelli di business, tra cui quelli basati su annunci pubblicitari, abbonamento, noleggio e vendita elettronica.

- Varie tecnologie di erogazione dei contenuti, tra cui lo streaming HTTP dinamico, lo streaming via RTMP (Real Time Media Protocol) con Flash® Media Server e il download progressivo con HTTP.

AIR per TV include anche il supporto incorporato per RTMPE, la versione cifrata di RTMP, per le soluzioni di streaming esistenti che hanno requisiti di sicurezza meno stringenti. RTMPE e le tecnologie di verifica SWF correlate sono supportate in Flash Media Server.

Per ulteriori informazioni, vedete [Adobe Flash Access](#).

Audio multicanale

A partire da AIR 3, AIR per TV supporta l'audio multicanale per i video scaricati con download progressivo da un server HTTP. Il supporto comprende i codec seguenti:

- AC-3 (Dolby Digital)
- E-AC-3 (Enhanced Dolby Digital)
- DTS Digital Surround
- DTS Express
- Audio DTS-HD High Resolution
- Audio DTS-HD Master

***Nota:** il supporto dell'audio multicanale nei video scaricati in streaming da un server Adobe Flash Media non è ancora disponibile.*

Input di videogame

A partire da AIR 3, AIR per TV supporta le API ActionScript che consentono alle applicazioni di comunicare con i dispositivi di input videogiochi collegati, ad esempio, joystick, gamepad e wand. Benché questi accessori siano chiamati dispositivi di input per videogiochi, possono essere utilizzati non solo dai videogiochi ma anche da applicazioni AIR per TV.

È disponibile un'ampia gamma di dispositivi di input per videogiochi con funzionalità differenti. Di conseguenza, i dispositivi sono definiti a livello generale nell'API in modo tale che un'applicazione possa funzionare bene con diversi tipi di dispositivi di input per videogiochi (anche sconosciuti).

La classe `GameInput` è il punto di ingresso delle API ActionScript per l'input da videogiochi. Per ulteriori informazioni, vedete [GameInput](#).

Rendering con grafica accelerata Stage3D

A partire da AIR 3, AIR per TV supporta il rendering con grafica accelerata Stage3D. Le API ActionScript [Stage3D](#) sono una serie di API di basso livello con accelerazione grafica che abilitano funzionalità 2D e 3D avanzate. Queste API di basso livello danno agli sviluppatori la flessibilità necessaria per sfruttare l'accelerazione hardware tramite GPU al fine di ottenere sensibili miglioramenti delle prestazioni. Potete anche usare motori di gaming che supportano le API ActionScript Stage3D.

Per ulteriori informazioni, vedete [Gaming engines, 3D, and Stage 3D](#) (Motori di gaming, 3D e Stage3D).

Estensioni native

Quando utilizzate come target il profilo `extendedTV`, potete usare i pacchetti ANE (estensioni native AIR).

Solitamente i produttori di dispositivi mettono a disposizione i pacchetti ANE per fornire l'accesso a funzioni del dispositivo altrimenti non supportate da AIR. Ad esempio, un'estensione nativa potrebbe consentirvi di cambiare canale su un televisore o mettere in pausa la riproduzione in un lettore video.

Quando create il pacchetto di un'applicazione AIR per TV che utilizza pacchetti ANE, il risultato dell'operazione è un file AIRN anziché un file AIR.

Le estensioni native per i dispositivi AIR per TV sono sempre estensioni native *device-bundled*. Device-bundled significa che le librerie delle estensioni sono installate sul dispositivo AIR per TV. Il pacchetto ANE che includete nel pacchetto della vostra applicazione non contiene *mai* le librerie native delle estensioni. A volte contiene una versione solo ActionScript dell'estensione nativa. Questa versione solo ActionScript è uno stub o simulatore dell'estensione. Il produttore del dispositivo installa l'estensione reale, comprese le librerie native, sul dispositivo.

Se sviluppate estensioni native, tenete presente quanto segue:

- Consultate sempre il produttore del dispositivo se state creando un'estensione nativa AIR per TV per i suoi dispositivi.
- Per alcuni dispositivi AIR per TV, solo il produttore si occupa della creazione delle estensioni native.
- Per tutti i dispositivi AIR per TV, è il produttore a decidere quali estensioni native installare.
- I tool di sviluppo per la creazione di estensioni native AIR per TV sono diversi a seconda del produttore.

Per ulteriori informazioni sulle estensioni native nella vostra applicazione AIR, vedete [“Uso di estensioni native per Adobe AIR”](#) a pagina 154 (Uso di estensioni native per Adobe AIR).

Per informazioni sulla creazione di estensioni native, vedete [Developing Native Extensions for Adobe AIR](#) (Sviluppo di estensioni native per Adobe AIR).

Considerazioni sulla progettazione di applicazioni AIR per TV

Considerazioni per il video

Linee guida per la codifica video

Per lo streaming video su dispositivi TV, Adobe consiglia le seguenti linee guida per la codifica:

Codec video:	H.264, profilo Main o High, codifica progressiva
Risoluzione:	720i, 720p, 1080i o 1080p
Frequenza fotogrammi:	24 fotogrammi al secondo oppure 30 fotogrammi al secondo
Codec audio:	AAC-LC o AC-3, 44.1 kHz, stereo, o questi codec per audio multicanale: E-AC-3, DTS, DTS Express, DTS-HD High Resolution Audio, DTS-HD Master Audio

Velocità di trasmissione combinata: fino a 8M bps a seconda dell'ampiezza di banda disponibile

Velocità di trasmissione audio: fino a 192 Kbps

Forma dei pixel: 1 × 1

Adobe consiglia di utilizzare il codec H.264 per la distribuzione di video a dispositivi AIR per TV.

***Nota:** AIR per TV supporta anche il video codificato con i codec Sorenson Spark e On2 VP6. Tuttavia, l'hardware non esegue la decodifica e la presentazione per questi codec, che invece vengono eseguite dal runtime utilizzando il software, con l'effetto di riprodurre il video con una frequenza di fotogrammi molto inferiore. Di conseguenza, usate il codec H.264 se possibile.*

La classe StageVideo

AIR per TV supporta la decodifica e la presentazione hardware dei video con codifica H.264. Utilizzate la classe StageVideo per abilitare questa funzionalità.

Vedete [Uso della classe StageVideo per la presentazione con accelerazione hardware](#) nella *Guida per gli sviluppatori di ActionScript 3.0* per informazioni dettagliate su:

- L'API della classe StageVideo e delle classi correlate
- Le limitazioni derivanti dall'uso della classe StageVideo

Per supportare in modo ottimale le applicazioni AIR esistenti che utilizzano l'oggetto Video per i video con codifica H.264, AIR per TV usa *internamente* un oggetto StageVideo. In questo modo la riproduzione video sfrutta i benefici della decodifica e della presentazione hardware. Tuttavia, l'oggetto Video è sottoposto alle stesse restrizioni che valgono per un oggetto StageVideo. Ad esempio, se l'applicazione tenta di ruotare il video, la rotazione non avviene, poiché è l'hardware, e non il runtime, che esegue la presentazione del video.

Quando invece scrivete nuove applicazioni, usate l'oggetto StageVideo per i video con codifica H.264.

Per un esempio di utilizzo della classe StageVideo, vedete l'articolo [Delivering video and content for the Flash Platform on TV](#).

Linee guida per la distribuzione video

Su un dispositivo AIR per TV, l'ampiezza di banda disponibile della rete può variare durante la riproduzione video. Le variazioni possono verificarsi, ad esempio, quando un altro utente utilizza la stessa connessione Internet.

Di conseguenza, Adobe consiglia di distribuire i video utilizzando velocità di trasferimento adattive. Ad esempio, sul lato server, Flash Media Server supporta tale funzionalità. Sul lato client, potete utilizzare il framework OSMF (Open Source Media Framework).

I protocolli seguenti sono disponibili per la distribuzione di contenuto video a un'applicazione AIR per TV tramite la rete:

- Streaming dinamico HTTP e HTTPS (formato F4F)
- Streaming RTMP, RTMPE, RTMFP, RTMPT e RTMPTE
- Download progressivo HTTP e HTTPS

Per ulteriori informazioni, consultare i seguenti riferimenti:

- [Adobe Flash Media Server Developer's Guide](#)
- [Open Source Media Framework](#)

Considerazioni per l'audio

Il codice ActionScript per la riproduzione sonora non è differente nelle applicazioni AIR per TV rispetto ad altri tipi di applicazioni AIR. Per informazioni, vedete [Operazioni con l'audio](#) nella *Guida per gli sviluppatori di ActionScript 3.0*.

Relativamente al supporto dell'audio multicanale in AIR per TV, considerate quanto segue:

- AIR per TV supporta l'audio multicanale per i video scaricati con download progressivo da un server HTTP. Il supporto dell'audio multicanale nei video scaricati in streaming da un server Adobe Flash Media non è ancora disponibile.
- Anche se AIR per TV supporta numerosi codec audio, non tutti i *dispositivi* AIR per TV supportano l'intera serie. Utilizzate il `flash.system.Capabilities` metodo `hasMultiChannelAudio()` per verificare se un dispositivo AIR per TV supporta un particolare codec audio multicanale, quale AC-3.

Ad esempio, considerate un'applicazione che scarica progressivamente un file video da un server. Il server presenta diversi file video H.264 che supportano diversi codec audio multicanale. L'applicazione può utilizzare `hasMultiChannelAudio()` per determinare quale file video richiedere dal server. In alternativa, l'applicazione può inviare al server la stringa contenuta in `Capabilities.serverString`. Tale stringa indica quali codec audio multicanale sono disponibili, consentendo al server di selezionare il file video appropriato.

- Quando utilizzate uno dei codec audio DTS, in alcuni scenari `hasMultiChannelAudio()` restituisce `true`, ma l'audio DTS non viene riprodotto.

Considerate, ad esempio, un lettore Blu-ray con un output S/PDIF, collegato a un vecchio amplificatore. Il vecchio amplificatore non supporta DTS, ma S/PDIF non dispone di un protocollo per notificarlo al lettore Blu-ray. Il lettore Blu-ray invia il flusso DTS al vecchio amplificatore, ma non viene emesso alcun suono. Di conseguenza, quando utilizzate DTS, è buona norma fornire un'interfaccia utente in modo che l'utente possa indicare se non viene riprodotto alcun suono. L'applicazione potrà quindi ripristinare un codec diverso.

La tabella seguente indica quando utilizzare codec audio differenti nelle applicazioni AIR per TV. La tabella indica inoltre quando i dispositivi AIR per TV impiegano acceleratori hardware per decodificare un codec audio. La decodifica hardware migliora le prestazioni e alleggerisce la CPU.

Codec audio	Disponibilità su dispositivi AIR per TV	Decodifica hardware	Quando usare questo codec audio	Ulteriori informazioni
AAC	Sempre	Sempre	Nei video con codifica H.264. Per lo streaming audio, ad esempio nell'ambito di un servizio di streaming musicale su Internet.	Quando utilizzate un flusso AAC solo audio, incapsulatelo in un contenitore MP4.
MP3	Sempre	No	Per i suoni nei file SWF dell'applicazione. Nei video con codifica Sorenson Spark o On2 VP6.	Un video H.264 che usa il formato mp3 per l'audio non viene riprodotto sui dispositivi AIR per TV.
AC-3 (Dolby Digital) E-AC-3 (Enhanced Dolby Digital) DTS Digital Surround DTS Express Audio DTS-HD High Resolution Audio DTS-HD Master	Controllare	Sì	Nei video con codifica H.264.	Generalmente, AIR per TV trasmette un flusso audio multicanale a un ricevitore audio/video esterno che decodifica e riproduce l'audio.
Speex	Sempre	No	Ricezione di un flusso vocale dal vivo.	Un video H.264 che usa Speex per l'audio non viene riprodotto sui dispositivi AIR per TV. Utilizzate Speex solo nei video con codifica Sorenson Spark o On2 VP6.
NellyMoser	Sempre	No	Ricezione di un flusso vocale dal vivo.	Un video H.264 che usa NellyMoser per l'audio non viene riprodotto sui dispositivi AIR per TV. Utilizzate NellyMoser solo nei video con codifica Sorenson Spark o On2 VP6.

Nota: alcuni file video contengono due flussi audio. Ad esempio, un file video può contenere sia flussi AAC che AC3. AIR for TV non supporta tali file video, quindi l'uso di tale file può portare all'assenza di audio per il video.

Accelerazione hardware grafica

Uso dell'accelerazione hardware grafica

I dispositivi AIR per TV offrono l'accelerazione hardware per le operazioni con grafica 2D. Gli acceleratori grafici hardware del dispositivo sgravano la CPU dal carico di lavoro necessario per effettuare le seguenti operazioni:

- Rendering bitmap
- Ridimensionamento bitmap
- Fusione bitmap
- Riempimento di rettangoli pieni

L'accelerazione grafica tramite hardware fa sì che molte operazioni grafiche in un'applicazione AIR per TV siano ad alte prestazioni. Alcune di queste operazioni sono:

- Le transizioni di scorrimento
- Le transizioni di ridimensionamento

- Dissolvenza in entrata e in uscita
- Composizione di più immagini con alfa

Per ottenere i benefici prestazionali dell'accelerazione grafica hardware per questi tipi di operazioni, utilizzate una delle tecniche seguenti:

- Impostate la proprietà `cacheAsBitmap` su `true` per gli oggetti `MovieClip` e altri oggetti di visualizzazione il contenuto per lo più non è soggetto a variazioni. Quindi, eseguite le transizioni di scorrimento, le transizioni di dissolvenza e le fusioni alfa su questi oggetti.
- Utilizzate la proprietà `cacheAsBitmapMatrix` sugli oggetti di visualizzazione che volete ridimensionare o traslare (applicare il riposizionamento `x` e `y`).

Utilizzando la classe `Matrix` per il ridimensionamento e la traslazione, gli acceleratori hardware del dispositivo eseguono le operazioni. In alternativa, valutate la possibilità di modificare le dimensioni di un oggetto di visualizzazione la cui proprietà `cacheAsBitmap` è impostata su `true`. Quando le dimensioni cambiano, il software del runtime ridisegna la bitmap. Il ridisegno tramite il software produce prestazioni inferiori rispetto al ridimensionamento con accelerazione hardware eseguito mediante un'operazione `Matrix`.

Ad esempio, considerate un'applicazione che visualizza un'immagine che si espande ogni volta che è selezionata da un utente finale. Utilizzate l'operazione di ridimensionamento `Matrix` più volte per dare l'illusione dell'espansione dell'immagine. Tuttavia, a seconda delle dimensioni dell'immagine originale e di quella finale, la qualità dell'immagine finale potrebbe non risultare accettabile. Pertanto, ripristinate le dimensioni dell'oggetto di visualizzazione quando le operazioni di espansione sono state completate. Poiché `cacheAsBitmap` è `true`, il software del runtime ridisegna l'oggetto di visualizzazione, ma solo una volta, e riproduce un'immagine di alta qualità.

***Nota:** solitamente, i dispositivi AIR per TV non supportano la rotazione e l'inclinazione con accelerazione hardware. Quindi, se specificate la rotazione e l'inclinazione nella classe `Matrix`, AIR per TV esegue tutte le operazioni `Matrix` nel software. Queste operazioni software possono avere un impatto negativo sulle prestazioni.*

- Utilizzate la classe `BitmapData` per creare comportamenti di caching bitmap personalizzati.

Per ulteriori informazioni sul caching bitmap, vedete le risorse seguenti.

- [Memorizzazione nella cache di oggetti di visualizzazione](#)
- [Caching bitmap](#)
- [Caching manuale delle bitmap](#)

Gestione della memoria grafica

Per eseguire le operazioni grafiche accelerate, gli acceleratori hardware usano una memoria grafica speciale. Se la vostra applicazione usa tutta la memoria grafica, funzionerà più lentamente perché AIR per TV torna a utilizzare il software per le operazioni grafiche.

Per gestire l'utilizzo della memoria grafica da parte dell'applicazione:

- Quando avete finito di utilizzare un'immagine o altri dati bitmap, rilasciate la relativa memoria grafica. Per farlo, chiamate il metodo `dispose()` della proprietà `bitmapData` dell'oggetto `Bitmap`. Ad esempio:

```
myBitmap.bitmapData.dispose();
```

***Nota:** il rilascio del riferimento all'oggetto `BitmapData` non libera immediatamente la memoria grafica. Il processo di garbage collection del runtime a un certo punto libera effettivamente la memoria grafica, ma una chiamata del metodo `dispose()` offre un maggiore controllo all'applicazione.*

- Utilizzate PerfMaster Deluxe, un'applicazione AIR fornita da Adobe, per comprendere meglio il funzionamento dell'accelerazione grafica hardware sul dispositivo di destinazione. Questa applicazione mostra i fotogrammi al secondo richiesti per eseguire varie operazioni. Usate PerfMaster Deluxe per confrontare implementazioni differenti della stessa operazione. Ad esempio, confrontate lo spostamento di un'immagine bitmap con lo spostamento di un'immagine vettoriale. PerfMaster Deluxe è disponibile su [Flash Platform for TV](#).

Gestione dell'elenco di visualizzazione

Per rendere invisibile un oggetto di visualizzazione, impostate la relativa proprietà `visible` su `false`. In tal modo l'oggetto rimane nell'elenco di visualizzazione, ma AIR per TV non ne esegue il rendering e non lo visualizza. Questa tecnica è utile per gli oggetti che entrano ed escono frequentemente dalla visualizzazione, poiché è a bassa intensità di elaborazione. Tuttavia, l'impostazione della proprietà `visible` su `false` non rilascia nessuna delle risorse dell'oggetto. Di conseguenza, quando avete finito di visualizzare un oggetto, o comunque non dovete visualizzarlo per molto tempo, rimuovetelo dall'elenco di visualizzazione. Inoltre, impostate tutti i riferimenti all'oggetto su `null`. Queste azioni consentono al garbage collector di rilasciare le risorse dell'oggetto.

Utilizzo di immagini PNG e JPEG

Due tra i più comuni formati di immagine usati nelle applicazioni sono PNG e JPEG. In merito all'uso di questi formati grafici nelle applicazioni AIR per TV, considerate quanto segue:

- AIR per TV solitamente usa l'accelerazione hardware per decodificare i file JPEG.
- AIR per TV solitamente usa il software per decodificare i file PNG. La decodifica dei file PNG nel software è veloce.
- PNG è l'unico formato bitmap utilizzabile da più piattaforme che supporta la trasparenza (un canale alfa).

Di conseguenza, osservate queste indicazioni quando usate questi formati immagine nelle applicazioni:

- Usate i file JPEG per le fotografie al fine di sfruttare i vantaggi della decodifica hardware accelerata.
- Usate i file di immagine PNG per gli elementi dell'interfaccia utente. Gli elementi dell'interfaccia utente possono infatti avere un'impostazione alfa, e la decodifica software garantisce prestazioni sufficientemente veloci per tali elementi.

Lo stage nelle applicazioni AIR per TV

Quando create un'applicazione AIR per TV ed effettuate operazioni con la classe Stage, valutate i seguenti fattori:

- Risoluzione dello schermo
- Area di visualizzazione sicura
- Modalità di ridimensionamento dello stage
- Allineamento dello stage
- Stato di visualizzazione dello stage
- Progettazione per formati di schermo differenti
- Impostazione della qualità per lo stage

Risoluzione dello schermo

Attualmente, i dispositivi TV presentano in genere una delle seguenti risoluzioni dello schermo: 540p, 720p, 1080p. Queste risoluzioni corrispondono ai valori seguenti nella classe `ActionScript Capabilities`:

Risoluzione dello schermo	Capabilities.screenResolutionX	Capabilities.screenResolutionY
540p	960	540
720p	1280	720
1080p	1920	1080

Per scrivere un'applicazione AIR per TV a schermo intero per un dispositivo specifico, codificate in modo permanente `Stage.stageWidth` e `Stage.stageHeight` per la risoluzione dello schermo del dispositivo. Tuttavia, per scrivere un'applicazione a schermo intero eseguibile su più dispositivi, utilizzate le proprietà `Capabilities.screenResolutionX` e `Capabilities.screenResolutionY` per impostare le dimensioni dello stage.

Ad esempio:

```
stage.stageWidth = Capabilities.screenResolutionX;  
stage.stageHeight = Capabilities.screenResolutionY;
```

Area di visualizzazione sicura

L'*area di visualizzazione sicura* di un televisore è un'area dello schermo rientrata rispetto ai margini dello schermo stesso. Il rientro di quest'area è sufficiente per far sì che l'utente finale possa vedere l'intera area, senza che la cornice dello schermo TV possa oscurarne qualsiasi parte. Poiché la cornice, che racchiude fisicamente lo schermo, varia a seconda della marca e del modello, anche la quantità di rientro necessaria è variabile. L'area di visualizzazione sicura ha lo scopo di garantire che una determinata area dello schermo sia sempre visibile. Viene anche definita *area di sicurezza titolo* (title safe area).

La *sovrascansione* è l'area dello schermo che non è visibile perché si trova dietro la cornice.

Adobe consiglia di applicare un rientro del 7,5% su ogni bordo dello schermo. Ad esempio:



Area di visualizzazione sicura per una risoluzione schermo di 1920x1080

Tenete sempre conto dell'area di visualizzazione sicura quando progettate un'applicazione AIR per TV a schermo intero:

- Utilizzate l'intero schermo per gli sfondi, ad esempio le immagini di sfondo o i colori di sfondo.
- Utilizzate l'area di visualizzazione sicura solo per gli elementi critici di un'applicazione, quali il testo, la grafica, il video e gli elementi dell'interfaccia utente come i pulsanti.

La tabella seguente mostra le dimensioni dell'area di visualizzazione sicura per ogni risoluzione tipica dello schermo, con l'applicazione di un rientro del 7,5%.

Risoluzione dello schermo	Larghezza e altezza dell'area di visualizzazione sicura	Larghezza del rientro sinistro e destro	Altezza del rientro superiore e inferiore
960 x 540	816 x 460	72	40
1280 x 720	1088 x 612	96	54
1920 x 1080	1632 x 918	144	81

Tuttavia, è sempre opportuno calcolare l'area di visualizzazione sicura in modo dinamico. Ad esempio:

```
var horizontalInset, verticalInset, safeAreaWidth, safeAreaHeight:int;

horizontalInset = .075 * Capabilities.screenResolutionX;
verticalInset = .075 * Capabilities.screenResolutionY;
safeAreaWidth = Capabilities.screenResolutionX - (2 * horizontalInset);
safeAreaHeight = Capabilities.screenResolutionY - (2 * verticalInset);
```

Modalità di ridimensionamento dello stage

Impostate `Stage.scaleMode` su `StageScaleMode.NO_SCALE` e intercettate gli eventi di ridimensionamento dello stage.

```
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.addEventListener(Event.RESIZE, layoutHandler);
```

Questa impostazione fa corrispondere le coordinate dello stage alle coordinate in pixel. Insieme allo stato di visualizzazione `FULL_SCREEN_INTERACTIVE` e all'allineamento dello stage `TOP_LEFT`, questa impostazione consente di utilizzare in modo efficace l'area di visualizzazione sicura.

Specificamente, nelle applicazioni a schermo intero, questa modalità di ridimensionamento comporta che le proprietà `stageWidth` e `stageHeight` della classe `Stage` corrispondano alle proprietà `screenResolutionX` e `screenResolutionY` della classe `Capabilities`.

Inoltre, quando le dimensioni della finestra dell'applicazione cambiano, il contenuto dello stage mantiene le dimensioni che sono state definite. Il runtime non esegue alcuna operazione automatica di layout o ridimensionamento. Il runtime invia inoltre l'evento `resize` della classe `Stage` quando le dimensioni della finestra cambiano. Pertanto potete controllare completamente il modo in cui viene modificato il contenuto dell'applicazione quando questa viene avviata e quando la finestra dell'applicazione viene ridimensionata.

Nota: il comportamento `NO_SCALE` è uguale a quello di qualsiasi altra applicazione AIR. Nelle applicazioni AIR per TV, tuttavia, questa impostazione è di importanza fondamentale per l'utilizzo dell'area di visualizzazione sicura.

Allineamento dello stage

Impostate `Stage.align` su `StageAlign.TOP_LEFT`:

```
stage.align = StageAlign.TOP_LEFT;
```

Con questo allineamento, la coordinata 0, 0 viene collocata nell'angolo superiore sinistro dello schermo, ovvero nella posizione più comoda per il posizionamento del contenuto con ActionScript.

Insieme alla modalità di ridimensionamento `NO_SCALE` e allo stato di visualizzazione `FULL_SCREEN_INTERACTIVE`, questa impostazione consente di utilizzare in modo efficace l'area di visualizzazione sicura.

Stato di visualizzazione dello stage

Impostate `Stage.displayState` in un'applicazione AIR per TV a schermo intero su `StageDisplayState.FULL_SCREEN_INTERACTIVE`:

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

Questo valore imposta l'applicazione AIR affinché espanda lo stage fino a occupare l'intero schermo, consentendo l'input da tastiera dell'utente.

Adobe consiglia di utilizzare l'impostazione `FULL_SCREEN_INTERACTIVE`. Insieme alla modalità di ridimensionamento `NO_SCALE` e all'allineamento dello stage `TOP_LEFT`, questa impostazione consente di utilizzare in modo efficace l'area di visualizzazione sicura.

Di conseguenza, per un'applicazione a schermo intero, scrivete il codice seguente in un gestore per l'evento `ADDED_TO_STAGE` nella classe principale del documento:

```
private function onStage(evt:Event):void
{
    stage.scaleMode = StageScaleMode.NO_SCALE;
    stage.align = StageAlign.TOP_LEFT;
    stage.addEventListener(Event.RESIZE, onResize);
    stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
}
```

Quindi, nel gestore dell'evento `RESIZE`:

- Confrontate le dimensioni della risoluzione dello schermo con la larghezza e l'altezza dello stage. Se corrispondono, l'evento `RESIZE` si è verificato perché lo stato di visualizzazione dello stage è cambiato in `FULL_SCREEN_INTERACTIVE`.
- Calcolate e salvate le dimensioni dell'area di visualizzazione sicura e i relativi rientri.

```
private function onResize(evt:Event):void
{
    if ((Capabilities.screenResolutionX == stage.stageWidth) &&
        (Capabilities.screenResolutionY == stage.stageHeight))
    {
        // Calculate and save safe viewing area dimensions.
    }
}
```

Quando le dimensioni dello stage sono uguali a `Capabilities.screenResolutionX` e `screenResolutionY`, AIR per TV fa in modo che l'hardware produca la migliore fedeltà possibile per il video e la grafica.

Nota: la fedeltà alla quale la grafica e il video vengono visualizzati su uno schermo TV può variare rispetto ai valori `Capabilities.screenResolutionX` e `screenResolutionY`, a seconda del dispositivo che esegue AIR per TV. Ad esempio, un set-top box che esegue AIR per TV può avere una risoluzione video di 1280 x 720 mentre il televisore collegato potrebbe avere una risoluzione di 1920x1080. In ogni caso, AIR per TV fa in modo che l'hardware produca sempre la migliore fedeltà possibile. Pertanto, in questo esempio, l'hardware riproduce un video 1080p a una risoluzione di 1920x1080.

Progettazione per formati di schermo differenti

Potete sviluppare la stessa applicazione AIR per TV a schermo intero in modo che possa funzionare ed essere visualizzata correttamente su più dispositivi AIR per TV. Procedete nel modo seguente:

- 1 Impostate le proprietà dello stage `scaleMode`, `align` e `displayState` sui valori consigliati: `StageScaleMode.NO_SCALE`, `StageAlign.TOP_LEFT` e `StageDisplayState.FULL_SCREEN_INTERACTIVE`, rispettivamente.
- 2 Impostate l'area di visualizzazione sicura in base a `Capabilities.screenResolutionX` e `Capabilities.screenResolutionY`.
- 3 Regolate le dimensioni e il layout del contenuto in base alla larghezza e all'altezza dell'area di visualizzazione sicura. Anche se gli oggetti del contenuto sono di grandi dimensioni, specialmente rispetto alle applicazioni per dispositivi mobili, concetti quali layout dinamico, posizionamento relativo e contenuto adattivo hanno le stesse implicazioni. Per ulteriori informazioni su come ActionScript supporta questi concetti, vedete [Authoring mobile Flash content for multiple screen sizes](#).

Qualità dello stage

La proprietà `Stage.quality` di un'applicazione AIR per TV è sempre `StageQuality.High`. Non potete modificarla. Questa proprietà specifica la qualità di rendering per tutti gli oggetti dello stage.

Gestione dell'input tramite telecomando

Gli utenti solitamente interagiscono con un'applicazione AIR per TV utilizzando un telecomando. Tuttavia, l'input da telecomando deve essere gestito esattamente come l'input da tastiera di un'applicazione desktop. Specificamente, gestite l'evento `KeyboardEvent.KEY_DOWN`. Per maggiori informazioni, vedete [Rilevamento dell'input da tastiera](#) nella *Guida per gli sviluppatori di ActionScript 3.0*.

I tasti del telecomando vengono mappati a costanti ActionScript. Ad esempio, i tasti sul tastierino direzionale di un telecomando vengono mappati nel modo seguente:

Tasto del tastierino direzionale del telecomando	Costante ActionScript 3.0
Su	<code>Keyboard.UP</code>
Giù	<code>Keyboard.DOWN</code>
Sinistra	<code>Keyboard.LEFT</code>
Destra	<code>Keyboard.RIGHT</code>
OK o Seleziona	<code>Keyboard.ENTER</code>

In AIR 2.5 sono state aggiunte molte altre costanti `Keyboard` per il supporto dell'input da telecomando. Per un elenco completo, vedete la [classe Keyboard](#) nella *Guida di riferimento di ActionScript 3.0 per la piattaforma Adobe Flash*.

Per garantire che la vostra applicazione funzioni sul numero più ampio di dispositivi possibile, Adobe consiglia quanto segue:

- Utilizzate solo i tasti del tastierino direzionale, se possibile.

Le serie di tasti del telecomando variano da un dispositivo all'altro, ma in genere tutti i dispositivi hanno un tastierino direzionale con gli stessi tasti.

Ad esempio, il telecomando di un lettore Blu-ray solitamente non presenta i tasti "Canale seguente" e "Canale precedente". Anche i tasti Riproduci, Pausa e Stop non sono presenti su tutti i telecomandi.

- Utilizzate i tasti Menu e Informazioni se l'applicazione necessita di altri tasti oltre a quelli del tastierino direzionale. Questi due tasti, dopo quelli direzionali, sono i più comuni sui telecomandi.
- Considerate l'ampia diffusione dei telecomandi universali.
Anche se state creando un'applicazione per un dispositivo particolare, tenete presente che molti utenti non usano il telecomando fornito con il dispositivo, bensì un telecomando universale. Inoltre, gli utenti non programmano sempre il proprio telecomando universale per simulare tutti i tasti del telecomando del dispositivo. Pertanto, è consigliabile utilizzare solo i tasti più comuni.
- Fate in modo che l'utente abbia sempre a disposizione una "via d'uscita" da qualsiasi situazione mediante uno dei tasti del tastierino direzionale.
Talvolta l'applicazione ha un buon motivo per utilizzare un tasto che non è tra quelli più comuni presenti sui telecomandi. Fornire una via d'uscita tramite uno dei tasti direzionali garantisce un comportamento regolare dell'applicazione su tutti i dispositivi.
- Non richiedete un input del puntatore a meno che non siate sicuri che il dispositivo AIR per TV di destinazione è in grado di generare un input tramite puntatore.
Benché molte applicazioni desktop prevedano l'input tramite mouse, la maggior parte dei televisori non supporta l'input da puntatore. Di conseguenza, se convertite delle applicazioni desktop per renderle utilizzabili sui televisori, ricordatevi di modificare l'applicazione in modo da escludere l'input da mouse. Le modifiche necessarie riguardano la gestione degli eventi e le istruzioni da fornire all'utente. Ad esempio, quando appare la schermata di avvio dell'applicazione, non visualizzate un testo che dice "Fate clic per iniziare".

Gestione dello stato di attivazione

Quando un elemento dell'interfaccia utente è attivo in un'applicazione desktop, esso è il target degli eventi di input utente quali quelli attivati tramite tastiera e mouse. Inoltre, un'applicazione evidenzia l'elemento dell'interfaccia utente che ha lo stato di attivazione. La gestione dello stato di attivazione in un'applicazione AIR per TV è differente rispetto a quella in un'applicazione desktop perché:

- Le applicazioni desktop spesso usano il tasto Tab per assegnare lo stato di attivazione al successivo elemento dell'interfaccia utente. L'uso del tasto Tab non è applicabile alle applicazioni AIR per TV. I dispositivi con telecomando solitamente non dispongono di un tasto Tab. Pertanto, la gestione dello stato di attivazione con la proprietà `tabEnabled` di un oggetto di visualizzazione come sul desktop non è possibile.
- Le applicazioni desktop spesso attendono che l'utente utilizzi il mouse per assegnare lo stato di attivazione a un elemento dell'interfaccia utente.

Pertanto, all'interno dell'applicazione, effettuate le seguenti operazioni:

- Aggiungete allo stage un listener di eventi per intercettare gli eventi Keyboard quali `KeyboardEvent.KEY_DOWN`.
- Fornite una logica applicativa per determinare quale elemento dell'interfaccia utente deve essere evidenziato all'utente finale. Assicuratevi che un elemento dell'interfaccia utente venga evidenziato all'avvio dell'applicazione.
- In base alla logica dell'applicazione, inviate l'evento Keyboard ricevuto dallo stage all'oggetto corrispondente all'elemento appropriato dell'interfaccia utente.

Potete utilizzare `Stage.focus` o `Stage.assignFocus()` per assegnare lo stato di attivazione a un elemento dell'interfaccia utente. Potete quindi aggiungere un listener di eventi a tale oggetto di visualizzazione in modo che riceva gli eventi da tastiera.

Progettazione dell'interfaccia utente

Per garantire il buon funzionamento dell'interfaccia utente di un'applicazione AIR per TV sui televisori, incorporate le raccomandazioni riportate di seguito e relative a:

- Capacità di risposta dell'applicazione
- Usabilità dell'applicazione
- Personalità e aspettative dell'utente

Capacità di risposta

Adottate i seguenti suggerimenti per aumentare al massimo la capacità di risposta di un'applicazione AIR per TV.

- Riducete il più possibile le dimensioni del file SWF iniziale dell'applicazione.

Nel file SWF iniziale, caricate solo le risorse necessarie per avviare l'applicazione. Ad esempio, caricate solo l'immagine della schermata di avvio dell'applicazione.

Benché questa raccomandazione sia valida per le applicazioni AIR desktop, è ancora più importante per i dispositivi AIR per TV. Ad esempio, i dispositivi AIR per TV non dispongono di una capacità di elaborazione equivalente a quella dei computer desktop. Inoltre, essi memorizzano l'applicazione in una memoria flash, che non consente un accesso rapido come i dischi rigidi dei computer desktop.

- Fate sì che l'applicazione venga eseguita ad almeno 20 fotogrammi al secondo.

Progettate la grafica in modo da raggiungere questo obiettivo. La complessità delle operazioni grafiche può influire sulla frequenza dei fotogrammi. Per consigli e suggerimenti su come migliorare le prestazioni di rendering, vedete [Ottimizzazione delle prestazioni per la piattaforma Adobe Flash](#).

***Nota:** l'hardware grafico dei dispositivi AIR per TV solitamente aggiorna lo schermo a una frequenza di 60 Hz o 120 Hz (60 o 120 volte al secondo). L'hardware esegue una scansione dello stage per rilevare gli aggiornamenti a una frequenza, ad esempio, di 30 o 60 fotogrammi al secondo per la visualizzazione su uno schermo da 60 Hz o 120 Hz. Tuttavia, la possibilità che l'utente possa fruire di queste elevate frequenze di fotogrammi dipende dalla complessità delle operazioni grafiche dell'applicazione.*

- Aggiornate lo schermo entro 100 - 200 millisecondi dall'input dell'utente.

Gli utenti diventano impazienti se gli aggiornamenti richiedono più tempo, con la conseguenza che spesso premono i tasti più volte.

Usabilità

Gli utenti delle applicazioni AIR per TV si trovano solitamente in un soggiorno, magari seduti su un divano a tre metri dal televisore. A volte la stanza è poco illuminata. Di solito l'utente usa un telecomando per inviare i comandi al dispositivo. L'applicazione può essere utilizzata da più persone, a volte insieme, altre volte in successione.

Pertanto, per progettare l'interfaccia utente dell'applicazione con un occhio alla sua usabilità, tenete conto delle considerazioni seguenti:

- Create elementi dell'interfaccia utente di grandi dimensioni.

Quando realizzate il testo, i pulsanti o qualsiasi altro elemento dell'interfaccia utente, tenete presente che l'utente è seduto dall'altra parte della stanza. Fate in modo che tutto sia facilmente visibile e leggibile da una distanza, ad esempio, di tre metri. Non cedete alla tentazione di riempire lo schermo di elementi solo perché è grande.

- Usate un buon contrasto per fare in modo che il contenuto sia facile da visualizzare e da leggere dalla posizione dell'utente.
- Fate sì che sia evidente quale elemento dell'interfaccia utente è attivo rendendo quell'elemento più luminoso.

- Utilizzate il movimento solo se necessario. Ad esempio, lo scorrimento da una schermata a quella successiva per conferire continuità può dare buoni risultati. Tuttavia, il movimento può anche distrarre l'attenzione se non aiuta l'utente a orientarsi oppure non è intrinseco all'applicazione.
- Fornite sempre all'utente un modo semplice per tornare indietro nell'interfaccia utente.

Per ulteriori informazioni sull'uso del telecomando, vedete “[Gestione dell'input tramite telecomando](#)” a pagina 136.

Personalità e aspettative dell'utente

Tenete presente che gli utenti delle applicazioni AIR per TV solitamente cercano un intrattenimento televisivo di qualità in un contesto di svago e relax. Non sono necessariamente esperti conoscitori di computer e cose tecnologiche.

Pertanto, progettate applicazioni AIR per TV con le seguenti caratteristiche:

- Non usate termini tecnici.
- Evitate le finestre di dialogo modali (a scelta obbligatoria).
- Usate istruzioni semplici e informali, appropriate per un ambiente domestico, non per un contesto di lavoro o tecnico.
- Utilizzate immagini che abbiano l'alta qualità grafica che gli utenti televisivi si aspettano.
- Create un'interfaccia utente che funzioni facilmente con un dispositivo di telecomando. Non usate interfacce utente o elementi visivi che sono indicati soprattutto per applicazioni desktop o mobili. Ad esempio, le interfacce utente dei dispositivi desktop e mobili spesso prevedono pulsanti sui quali l'utente deve puntare e fare clic con un mouse o un dito.

Caratteri e testo

Potete usare sia i caratteri dispositivo che quelli incorporati nelle applicazioni AIR per TV.

I caratteri dispositivo sono quelli installati all'interno di un dispositivo. Tutti i dispositivi AIR per TV hanno i seguenti caratteri dispositivo:

Nome carattere	Descrizione
<code>_sans</code>	Il carattere dispositivo <code>_sans</code> è di tipo sans-serif (senza grazie). Il carattere dispositivo <code>_sans</code> installato in tutti i dispositivi AIR per TV è Myriad Pro. Tipicamente, un carattere sans-serif (senza grazie) viene visualizzato meglio su un televisore rispetto ai caratteri serif (con grazie), in virtù della distanza di visione.
<code>_serif</code>	Il carattere dispositivo <code>_serif</code> è di tipo serif (con grazie). Il carattere dispositivo <code>_serif</code> installato in tutti i dispositivi AIR per TV è Minion Pro.
<code>_typewriter</code>	Il carattere dispositivo <code>_typewriter</code> è di tipo monospace (a spaziatura fissa). Il carattere dispositivo <code>_typewriter</code> installato in tutti i dispositivi AIR per TV è Courier Std.

Tutti i dispositivi AIR per TV hanno anche i seguenti caratteri dispositivo asiatici:

Nome carattere	Lingua	Categoria carattere	Codice internazionale
RyoGothicPlusN-Regular	Giapponese	sans	ja
RyoTextPlusN-Regular	Giapponese	serif	ja
AdobeGothicStd-Light	Coreano	sans	ko

Nome carattere	Lingua	Categoria carattere	Codice internazionale
AdobeHeitiStd-Regular	Cinese semplificato	sans	zh_CN
AdobeSongStd-Light	Cinese semplificato	serif	zh_CN
AdobeMingStd-Light	Cinese tradizionale	serif	zh_TW e zh_HK

Questi caratteri dispositivo AIR per TV presentano le seguenti caratteristiche:

- Provengono dalla libreria Adobe® Type Library
- Vengono visualizzati bene sui televisori
- Sono progettati per la titolazione video
- Sono caratteri outline, non bitmap

Nota: i produttori dei dispositivi spesso incorporano altri caratteri dispositivo nei loro prodotti in aggiunta ai caratteri dispositivo AIR per TV.

Adobe mette a disposizione un'applicazione chiamata FontMaster Deluxe che visualizza tutti i caratteri dispositivo presenti in un dispositivo. L'applicazione è disponibile su [Flash Platform for TV](#).

Potete anche incorporare i caratteri nelle applicazioni AIR per TV. Per informazioni sui caratteri incorporati, vedete [Rendering avanzato del testo](#) nella *Guida per gli sviluppatori di ActionScript 3.0*.

Adobe fornisce le seguenti raccomandazioni sull'uso dei campi di testo TLF:

- Usate i campi di testo TLF per le lingue asiatiche al fine di sfruttare le impostazioni specifiche del contesto linguistico in cui viene eseguita l'applicazione. Impostate la proprietà `locale` dell'oggetto `TextLayoutFormat` associato all'oggetto `TLFTextField`. Per determinare le impostazioni locali, vedete [Selezione delle impostazioni locali](#) nella *Guida per gli sviluppatori di ActionScript 3.0*.
- Specificate il nome del carattere nella proprietà `fontFamily` dell'oggetto `TextLayoutFormat` se il carattere non è uno dei caratteri dispositivo AIR per TV. AIR per TV utilizza il carattere se è disponibile nel dispositivo, altrimenti, in base all'impostazione `locale`, lo sostituisce con il carattere dispositivo AIR per TV appropriato.
- Specificate `_sans`, `_serif`, o `_typewriter` per la proprietà `fontFamily` e impostate la proprietà `locale` per far sì che AIR per TV scelga il carattere dispositivo AIR per TV corretto. A seconda delle impostazioni locali, AIR per TV sceglie dal set di caratteri dispositivo asiatici o non asiatici. Queste impostazioni forniscono un metodo semplice per utilizzare automaticamente il carattere corretto per le quattro principali lingue asiatiche e l'inglese.

Nota: se utilizzate i campi di testo classici per il testo di una lingua asiatica, specificate il nome di un carattere dispositivo AIR per TV per garantire un rendering corretto. Se siete sicuri che un altro carattere è installato nel dispositivo di destinazione, potete anche specificare quel carattere.

Per quanto riguarda le prestazioni dell'applicazione, considerate quanto segue:

- I campi di testo classici consentono un rendering più veloce dei campi di testo TLF.
- Un campo di testo classico che usa caratteri bitmap garantisce le prestazioni più veloci.

I caratteri bitmap forniscono una bitmap per ogni singolo simbolo, a differenza dei caratteri outline che forniscono solo i dati del contorno di ciascun simbolo. Sia i caratteri dispositivo che quelli incorporati possono essere caratteri bitmap.

- Se specificate un carattere dispositivo, assicuratevi che sia installato nel dispositivo di destinazione. Se non lo è, AIR per TV trova e utilizza un altro carattere che è installato nel dispositivo. Tuttavia, questo comportamento rallenta le prestazioni dell'applicazione.

- Come con qualsiasi altro oggetto di visualizzazione, se un oggetto TextField rimane per lo più invariato, impostatene la proprietà `cacheAsBitmap` su `true`. Questa impostazione permette di migliorare le prestazioni nelle transizioni quali la dissolvenza, lo scorrimento e la fusione alfa. Usate `cacheAsBitmapMatrix` per il ridimensionamento e la traslazione. Per ulteriori informazioni, vedete “[Accelerazione hardware grafica](#)” a pagina 130.

Sicurezza del file system

Le applicazioni AIR per TV sono applicazioni AIR e, di conseguenza, possono accedere al file system del dispositivo. Tuttavia, per un dispositivo "da soggiorno" è fondamentale che un'applicazione non possa accedere ai file di sistema del dispositivo o ai file di altre applicazioni. Gli utenti dei televisori e di dispositivi correlati non si aspettano e non tollerano errori del dispositivo; dopo tutto, stanno guardando la TV.

Di conseguenza, un'applicazione AIR per TV ha una visibilità limitata del file system del dispositivo. Utilizzando ActionScript 3.0, l'applicazione può accedere solo a directory specifiche (e alle relative sottodirectory). Inoltre, i nomi delle directory utilizzati in ActionScript non sono i nomi effettivi delle directory del dispositivo. Questo livello supplementare protegge le applicazioni AIR per TV dall'accesso fraudolento o non desiderato a file locali che non appartengono all'applicazione.

Per informazioni dettagliate, vedete [Visualizzazione directory per applicazioni AIR per TV](#).

Sandbox dell'applicazione AIR

Le applicazioni AIR per TV vengono eseguite nella sandbox dell'applicazione AIR, descritta in [Sandbox dell'applicazione AIR](#).

L'unica differenza per le applicazioni AIR per TV consiste nel fatto che esse hanno un accesso limitato al file system, come descritto in “[Sicurezza del file system](#)” a pagina 141.

Ciclo di vita dell'applicazione

A differenza di quanto avviene in un ambiente desktop, l'utente finale non può chiudere la finestra in cui viene eseguita l'applicazione AIR per TV. Pertanto, mettete a disposizione nell'interfaccia utente un meccanismo per uscire dall'applicazione.

In genere, un dispositivo consente all'utente finale di uscire direttamente da un'applicazione mediante il tasto Esci del telecomando. Tuttavia, AIR per TV non invia l'evento `flash.events.Event.EXITING` all'applicazione, quindi dovete salvare frequentemente lo stato dell'applicazione in modo tale che essa possa ripristinarsi a uno stato ragionevole all'avvio successivo.

Cookie HTTP

AIR per TV supporta cookie HTTP persistenti e di sessione. AIR per TV memorizza i cookie delle applicazioni AIR in una directory specifica dell'applicazione:

```
/app-storage/<app id>/Local Store
```

Il file dei cookie file è denominato `cookies`.

Nota: su altri tipi di dispositivi, quali dispositivi desktop, AIR non memorizza cookie per ogni singola applicazione. La memorizzazione di cookie specifici dell'applicazione supporta il modelli di sicurezza dell'applicazione e del sistema di AIR per TV.

Utilizzate la proprietà ActionScript `URLRequest.manageCookies` nel modo seguente:

- Impostate `manageCookies` su `true`. Questo è il valore predefinito e indica che AIR per TV aggiunge automaticamente i cookie alle richieste HTTP e memorizza i cookie nella risposta HTTP.

***Nota:** anche quando `manageCookies` è `true`, l'applicazione può aggiungere manualmente un cookie a una richiesta HTTP mediante `URLRequest.requestHeaders`. Se questo cookie ha lo stesso nome di un cookie gestito da AIR per TV, la richiesta contiene due cookie con lo stesso nome. I valori dei due cookie possono essere differenti.*

- Impostate `manageCookies` su `false`. Questo valore indica che l'applicazione è responsabile dell'invio dei cookie nelle richieste HTTP e della memorizzazione dei cookie nella risposta HTTP.

Per ulteriori informazioni, vedete [URLRequest](#).

Flusso di lavoro per lo sviluppo di un'applicazione AIR per TV

Per sviluppare applicazioni AIR per TV potete utilizzare gli strumenti della piattaforma Adobe Flash indicati di seguito:

- Adobe Flash Professional

Adobe Flash Professional CS5.5 supporta AIR 2.5 per TV, la prima versione di AIR che supporta le applicazioni AIR per TV.

- Adobe Flash® Builder®

Flash Builder 4.5 supporta AIR 2.5 per TV.

- AIR SDK

A partire da AIR 2.5, potete sviluppare le vostre applicazioni utilizzando i tool della riga di comando disponibili nel kit SDK AIR. Per scaricare AIR SDK, vedete <http://www.adobe.com/it/products/air/sdk/>.

Utilizzo di Flash Professional

L'uso di Flash Professional per sviluppare, provare e pubblicare applicazioni AIR per TV è analogo all'utilizzo dello stesso programma per le applicazioni AIR desktop.

Tuttavia, quando scrivete il codice ActionScript 3.0, usate solo le classi e i metodi supportati dai profili AIR `tv` e `extendedTV`. Per informazioni dettagliate, vedete “[Profili dispositivo](#)” a pagina 254.

Impostazioni del progetto

Effettuate le operazioni seguenti per configurare il progetto per un'applicazione AIR per TV:

- Nella scheda Flash della finestra di dialogo Impostazioni pubblicazione, impostate la voce Lettore su almeno AIR 2.5.
- Nella scheda Generali della finestra delle impostazioni di Adobe AIR (Impostazioni applicazione e programma di installazione), impostate il profilo su `TV` o `extendedTV`.

Esecuzione del debug

Potete eseguire l'applicazione utilizzando AIR Debug Launcher in Flash Professional. Procedete nel modo seguente:

- Per eseguire l'applicazione in modalità debug, selezionate:

Debug > Debug filmato > In AIR Debug Launcher (Desktop)

Una volta fatta questa selezione, per le operazioni di debug successive potrete selezionare:

Debug > Debug filmato > Debug

- Per eseguire l'applicazione senza le funzionalità di debug, selezionate:

Controllo > Prova filmato > In AIR Debug Launcher (Desktop)

Una volta fatta questa selezione, per le esecuzioni successive potrete selezionare Controllo > Prova filmato > Prova.

Poiché avete impostato il profilo AIR TV o extendedTV, AIR Debug Launcher fornisce un menu con i pulsanti di un telecomando. Potete usare questo menu per simulare i tasti premuti sul telecomando di un dispositivo.

Per ulteriori informazioni, vedete “[Debug remoto con Flash Professional](#)” a pagina 151.

Uso delle estensioni native

Se la vostra applicazione usa un'estensione nativa, seguite le istruzioni riportate in “[Elenco delle operazioni da eseguire per utilizzare un'estensione nativa](#)” a pagina 156.

Tuttavia, se un'applicazione usa delle estensioni native:

- Non potete pubblicare l'applicazione utilizzando Flash Professional. Per la pubblicazione, usate ADT. Vedete “[Creazione del pacchetto con ADT](#)” a pagina 148.
- Non potete eseguire l'applicazione o il debug della stessa con Flash Professional. Per il eseguire il debug dell'applicazione sul computer di sviluppo, usate ADL. Vedete “[Simulazione del dispositivo con ADL](#)” a pagina 149.

Utilizzo di Flash Builder

A partire da Flash Builder 4.5, Flash Builder supporta lo sviluppo di AIR per TV. L'uso di Flash Builder per sviluppare, provare e pubblicare applicazioni AIR per TV è analogo all'utilizzo dello stesso programma per le applicazioni AIR desktop.

Configurazione dell'applicazione

L'applicazione deve:

- Utilizzare l'elemento `Application` come classe contenitore nel file MXML, se utilizzate un file MXML:

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">

  <!-- Place elements here. -->

</s:Application>.
```

Importante: le applicazioni AIR per TV non supportano l'elemento `WindowedApplication`.

Nota: non dovete necessariamente usare un file MXML. Potete invece creare un progetto `ActionScript 3.0`.

- Utilizzare solo le classi e i metodi `ActionScript 3.0` supportati dai profili AIR tv e extendedTV. Per informazioni dettagliate, vedete “[Profili dispositivo](#)” a pagina 254.

Inoltre, nel file XML dell'applicazione, verificate che:

- L'attributo `xmlns` dell'elemento `application` sia impostato su AIR 2.5:

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```
- L'elemento `supportedProfiles` includa `tv` o `extendedTV`:

```
<supportedProfiles>tv</supportedProfiles>
```

Debug dell'applicazione

Potete eseguire l'applicazione utilizzando AIR Debug Launcher in Flash Builder. Procedete nel modo seguente:

- 1 Selezionate **Run > Debug Configurations**.
- 2 Verificate che il campo **Profile** sia impostato su **Desktop**.
- 3 Selezionate **Run > Debug** per eseguire l'applicazione in modalità debug oppure selezionate **Run > Run** per eseguirla senza le funzionalità di debug.

Poiché avete impostato l'elemento `supportedProfiles` su `TV` o `extendedTV`, AIR Debug Launcher fornisce un menu con i pulsanti di un telecomando. Potete usare questo menu per simulare i tasti premuti sul telecomando di un dispositivo.

Per ulteriori informazioni, vedete [“Debug remoto con Flash Builder”](#) a pagina 152.

Uso delle estensioni native

Se la vostra applicazione usa un'estensione nativa, seguite le istruzioni riportate in [“Elenco delle operazioni da eseguire per utilizzare un'estensione nativa”](#) a pagina 156.

Tuttavia, se un'applicazione usa delle estensioni native:

- Non potete pubblicare l'applicazione utilizzando Flash Builder. Per la pubblicazione, usate ADT. Vedete [“Creazione del pacchetto con ADT”](#) a pagina 148.
- Non potete eseguire l'applicazione o il debug della stessa con Flash Builder. Per il eseguire il debug dell'applicazione sul computer di sviluppo, usate ADL. Vedete [“Simulazione del dispositivo con ADL”](#) a pagina 149.

Proprietà del descrittore applicazione AIR per TV

Come per le altre applicazioni AIR, dovete impostare le proprietà di base dell'applicazione nel file descrittore dell'applicazione. Le applicazioni con profilo TV ignorano alcune delle proprietà specifiche delle applicazioni desktop, quali le dimensioni della finestra e la trasparenza. Le applicazioni destinate a dispositivi del profilo `extendedTV` possono utilizzare le estensioni native. Queste applicazioni identificano le estensioni native utilizzate in un elemento `extensions`.

Impostazioni comuni

Varie impostazioni del descrittore dell'applicazione sono importanti per tutte le applicazioni del profilo TV.

Versione del runtime AIR necessaria

Specificate la versione del runtime AIR necessaria per l'applicazione utilizzando lo spazio dei nomi del file descrittore dell'applicazione.

Lo spazio dei nomi, assegnato nell'elemento `application`, determina in larga parte quali funzioni può utilizzare la vostra applicazione. Ad esempio, considerate il caso di un'applicazione che utilizza lo spazio dei nomi AIR 2.5 mentre l'utente ha installato una versione successiva. In questo caso, l'applicazione "vede" comunque il comportamento AIR 2.5, anche se il comportamento è differente nella versione di AIR successiva. Solo quando cambiate lo spazio dei nomi e pubblicate un aggiornamento l'applicazione potrà accedere al nuovo comportamento e alle nuove funzioni. Gli aggiornamenti correttivi relativi alla sicurezza rappresentano un'importante eccezione a questa regola.

Specificate lo spazio dei nomi utilizzando l'attributo `xmlns` dell'elemento root `application`:

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

AIR 2.5 è la prima versione di AIR che supporta le applicazioni TV.

Identità dell'applicazione

Diverse impostazioni devono essere univoche per ciascuna applicazione che pubblicate. Queste impostazioni includono gli elementi `id`, `name` e `filename`.

```
<id>com.example.MyApp</id>  
<name>My Application</name>  
<filename>MyApplication</filename>
```

Versione dell'applicazione

Specificate la versione dell'applicazione nell'elemento `versionNumber`. Quando specificate un valore per `versionNumber`, potete usare una sequenza composta da un massimo di tre numeri separati da punti, come in "0.1.2". Ogni segmento del numero di versione può contenere fino a tre cifre. (In altre parole, "999.999.999" è il numero di versione più alto possibile.) Non dovete includere necessariamente tutti e tre i segmenti nel numero: "1" e "1.0" sono numeri di versione perfettamente validi.

Potete anche specificare un'etichetta per la versione utilizzando l'elemento `versionLabel`. Quando aggiungete un'etichetta di versione, questa viene visualizzata al posto del numero di versione.

```
<versionNumber>1.23.7</versionNumber>  
<versionLabel>1.23 Beta 7</versionLabel>
```

SWF principale dell'applicazione

Specificate il file SWF principale dell'applicazione nell'elemento secondario `versionLabel` dell'elemento `initialWindow`. Quando un'applicazione ha come target dispositivi nel profilo TV, dovete utilizzare un file SWF (le applicazioni basate su HTML non sono supportate).

```
<initialWindow>  
  <content>MyApplication.swf</content>  
</initialWindow>
```

Dovete includere il file nel pacchetto AIR (usando ADT o il vostro ambiente IDE). Un semplice riferimento al nome nel descrittore dell'applicazione non determina l'inclusione automatica del file nel pacchetto.

Proprietà della schermata principale

Vari elementi secondari dell'elemento `initialWindow` controllano l'aspetto iniziale e il comportamento della finestra principale dell'applicazione. Anche se la maggior parte di queste proprietà viene ignorata nei dispositivi dei profili TV, potete usare l'elemento `fullScreen`:

- `fullScreen` - Specifica se l'applicazione deve occupare interamente il display del dispositivo oppure condividere il display con il chrome standard del sistema operativo.

```
<fullScreen>true</fullScreen>
```

L'elemento visible

L'elemento `visible` è un elemento secondario dell'elemento `initialWindow`. AIR per TV ignora l'elemento `visible` perché il contenuto dell'applicazione è sempre visibile sui dispositivi AIR per TV.

Tuttavia, impostate comunque l'elemento `visible` su `true` se l'applicazione verrà utilizzata anche su dispositivi desktop.

Sui dispositivi desktop, il valore predefinito di questo elemento è `false`. Pertanto, se non includete l'elemento `visible`, il contenuto dell'applicazione non sarà visibile sui dispositivi desktop. Benché sia possibile utilizzare la classe `ActionScript NativeWindow` per rendere visibile il contenuto sui dispositivi desktop, i profili dispositivo TV non supportano la classe `NativeWindow`. Se provate a utilizzare la classe `NativeWindow` su un'applicazione eseguita su un dispositivo AIR per TV, l'applicazione non viene caricata. Non ha importanza se chiamate un metodo della classe `NativeWindow`; un'applicazione che usa la classe non viene caricata su un dispositivo AIR per TV.

Profili supportati

Se la vostra applicazione è pensata unicamente per un dispositivo TV, potete impedirne l'installazione su altri tipi di dispositivi informatici. Escludete gli altri profili dall'elenco dei profili supportati nell'elemento `supportedProfiles`:

```
<supportedProfiles>tv extendedTV</supportedProfiles>
```

Se un'applicazione usa un'estensione nativa, includete solo il profilo `extendedTV` nell'elenco dei profili supportati:

```
<supportedProfiles>extendedTV</supportedProfiles>
```

Se omettete l'elemento `supportedProfiles`, si presuppone che l'applicazione supporti tutti i profili.

Non includete *solo* il profilo `tv` nell'elenco `supportedProfiles`. Alcuni dispositivi TV eseguono sempre AIR per TV in una modalità che corrisponde al profilo `extendedTV`. Questo comportamento consente a AIR per TV di eseguire applicazioni che utilizzano estensioni native. Se il vostro elemento `supportedProfiles` specifica solo `tv`, significa che il contenuto è dichiarato incompatibile con la modalità AIR per TV per `extendedTV`. Di conseguenza, alcuni dispositivi TV non caricano un'applicazione che specifica soltanto il profilo `tv`.

Per un elenco delle classi `ActionScript` supportate nei profili `tv` ed `extendedTV`, vedete “[Funzionalità supportate dai profili](#)” a pagina 255.

Estensioni native obbligatorie

Le applicazioni che supportano il profilo `extendedTV` possono utilizzare le estensioni native.

Dichiarate, mediante gli elementi `extensions` e `extensionID`, tutte le estensioni native utilizzate dall'applicazione AIR nel descrittore dell'applicazione. L'esempio seguente illustra la sintassi da adottare per specificare due estensioni native richieste:

```
<extensions>
  <extensionID>com.example.extendedFeature</extensionID>
  <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

Se un'estensione non è elencata, l'applicazione non può utilizzarla.

L'elemento `extensionID` ha lo stesso valore dell'elemento `id` nel file descrittore dell'estensione. Quest'ultimo è un file XML denominato `extension.xml`, impacchettato nel file ANE fornito dal produttore del dispositivo.

Se elencate un'estensione nell'elemento `extensions` ma nel dispositivo AIR per TV non è installata tale estensione, l'applicazione non può essere eseguita. L'eccezione a questa regola è rappresentata dal caso in cui il file ANE che impacchettate con la vostra applicazione AIR per TV contenga una versione stub dell'estensione. In tal caso, l'applicazione può essere eseguita e utilizza la versione stub dell'estensione. La versione stub contiene codice ActionScript ma non codice nativo.

Icone dell'applicazione

I requisiti per le icone dell'applicazione nei dispositivi TV dipendono dal singolo prodotto. Ad esempio, il produttore del dispositivo specifica:

- Icone necessarie e relative dimensioni.
- Tipi di file necessari e relative convenzioni per i nomi.
- Come fornire le icone per l'applicazione, ad esempio se devono essere impacchettate insieme all'applicazione.
- Se specificare le icone in un elemento `icon` nel file descrittore dell'applicazione.
- Comportamento nel caso in cui l'applicazione non fornisca le icone.

Vedete il produttore del dispositivo per ulteriori dettagli.

Impostazioni ignorate

Le applicazioni per dispositivi TV ignorano le impostazioni applicazione valide solo per i dispositivi mobili, le finestre native o le funzioni del sistema operativo. Le impostazioni ignorate sono:

- `allowBrowserInvocation`
- `aspectRatio`
- `autoOrients`
- `customUpdateUI`
- `fileTypes`
- `height`
- `installFolder`
- `maximizable`
- `maxSize`
- `minimizable`
- `minSize`
- `programMenuFolder`
- `renderMode`
- `resizable`
- `systemChrome`
- `title`
- `transparent`
- `visible`
- `width`

- x
- y

Creazione del pacchetto di un'applicazione AIR per TV

Creazione del pacchetto con ADT

Potete usare il tool della riga di comando ADT di AIR per creare il pacchetto di un'applicazione AIR per TV. A partire dalla versione 2.5 del kit SDK AIR, ADT supporta la creazione di pacchetti per dispositivi TV. Prima di creare il pacchetto, compilate tutto il codice ActionScript e MXML. Dovete inoltre avere un certificato di firma del codice. Potete creare un certificato con il comando `ADT -certificate`.

Per informazioni dettagliate sui comandi e le opzioni ADT, vedete “[ADT \(AIR Developer Tool\)](#)” a pagina 171.

Creazione di un pacchetto AIR

Per creare un pacchetto AIR, utilizzate il comando `ADT -package`:

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 myApp.air myApp-app.xml myApp.swf icons
```

L'esempio presuppone che:

- Il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316.)
- Il certificato `codesign.p12` si trovi nella directory principale rispetto alla posizione in cui eseguite il comando ADT.

Eseguite il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf` e una directory di icone.

Quando eseguite il comando come indicato, ADT richiede la password dell'archivio di chiavi. Non tutti i programmi shell visualizzano i caratteri della password che digitate; quando avete immesso tutti i caratteri della password, premete Invio. In alternativa, potete usare il parametro `storepass` per includere la password nel comando ADT.

Creazione di un pacchetto AIRN

Se la vostra applicazione AIR per TV usa un'estensione nativa, create un pacchetto AIRN anziché un pacchetto AIR. Per creare un pacchetto AIRN, usate il comando `ADT package`, impostando il tipo di target su `airn`.

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp-app.xml myApp.swf icons -extdir C:\extensions
```

L'esempio presuppone che:

- Il percorso del tool ADT sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316.)
- Il certificato `codesign.p12` si trovi nella directory principale rispetto alla posizione in cui eseguite il comando ADT.
- Il parametro `-extdir` indica una directory che contiene i file ANE utilizzati dall'applicazione.

Questi file contengono una versione stub o simulatore esclusivamente dell'estensione ActionScript. La versione dell'estensione che contiene il codice nativo è installata nel dispositivo AIR per TV.

Eseguite il comando dalla directory che contiene i file dell'applicazione. Nell'esempio, i file dell'applicazione sono `myApp-app.xml` (il file descrittore dell'applicazione), `myApp.swf` e una directory di icone.

Quando eseguite il comando come indicato, ADT richiede la password dell'archivio di chiavi. Non tutti i programmi shell visualizzano i caratteri della password che digitate; quando avete immesso tutti i caratteri della password, premete Invio. In alternativa, potete usare il parametro `storepass` per includere la password nel comando.

Potete anche creare un file AIRI per un'applicazione AIR per TV che utilizza estensioni native. Il file AIRI è uguale al file AIRN, con la differenza che non è firmato. Ad esempio:

```
adt -prepare myApp.airi myApp.xml myApp.swf icons -extdir C:\extensions
```

Potete quindi creare un file AIRN dal file AIRI quando siete pronti per firmare l'applicazione:

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp.airi
```

Per ulteriori informazioni, vedete [Developing Native Extensions for Adobe AIR](#) (Sviluppo di estensioni native per Adobe AIR).

Creazione di un pacchetto con Flash Builder o Flash Professional

Flash Professional e Flash Builder consentono di pubblicare o esportare i pacchetti AIR senza dover eseguire manualmente ADT. La procedura per la creazione di un pacchetto AIR per un'applicazione AIR è descritta nella documentazione di questi programmi.

Attualmente, tuttavia, solo ADT è in grado di creare pacchetti AIRN, i pacchetti dell'applicazione per applicazioni AIR per TV che usano estensioni native.

Per ulteriori informazioni, consultare i seguenti riferimenti:

- [Package AIR applications with Flash Builder](#)
- [Pubblicazione per Adobe AIR con Flash Professional](#)

Debug di applicazioni AIR per TV

Simulazione del dispositivo con ADL

Il modo più veloce e facile di eseguire il test e il debug della maggior parte delle funzioni di un'applicazione consiste nell'eseguire l'applicazione sul computer di sviluppo utilizzando l'utilità ADL (Adobe Debug Launcher).

ADL usa l'elemento `supportedProfiles` nel descrittore dell'applicazione per determinare quale profilo utilizzare. Specificamente:

- Se sono elencati più profili, ADL sceglie il primo dell'elenco.
- Potete usare il parametro `-profile` di ADL per selezionare uno degli altri profili dell'elenco `supportedProfiles`.
- Se non includete l'elemento `supportedProfiles` nel descrittore dell'applicazione, potete specificare qualsiasi profilo per l'argomento `-profile`.

Ad esempio, utilizzate il comando seguente per avviare un'applicazione simulando il profilo `tv`:

```
adl -profile tv myApp-app.xml
```

Quando si simula il profilo `tv` o `extendedTV` in ambiente desktop con ADL, l'applicazione viene eseguita in un contesto il più simile possibile a quello di un dispositivo di destinazione. Ad esempio:

- Le API ActionScript che non fanno parte del profilo indicato nell'argomento `-profile` non sono disponibili.

- ADL consente l'immissione di comandi di input del dispositivo, ad esempio comandi remoti, tramite comandi di menu.
- Se è specificato `tv` o `extendedTV` nell'argomento `-profile`, ADL può simulare la classe `StageVideo` sul desktop.
- Specificando `extendedTV` nell'argomento `-profile`, si consente all'applicazione di utilizzare gli stub o i simulatori di estensioni native impacchettati con il file AIRN dell'applicazione.

Tuttavia, poiché ADL esegue l'applicazione sul desktop, la prova di applicazioni AIR per TV con ADL è soggetta ad alcune limitazioni:

- Non riflette le prestazioni dell'applicazione sul dispositivo di destinazione. Dovete eseguire i test delle prestazioni direttamente sul dispositivo di destinazione.
- Non simula le limitazioni dell'oggetto `StageVideo`. In genere si usa la classe `StageVideo`, e non la classe `Video`, per riprodurre un video quando si crea un'applicazione AIR destinata a dispositivi TV. La classe `StageVideo` sfrutta i vantaggi in termini di prestazioni dell'hardware del dispositivo, ma presenta anche alcuni limiti relativamente alla visualizzazione. ADL riproduce il video sul desktop senza tali limitazioni. Di conseguenza, è opportuno testare la riproduzione video direttamente sul dispositivo di destinazione.
- Non è in grado di simulare il codice nativo di un'estensione nativa. Potete comunque specificare il profilo `extendedTV`, che supporta le estensioni native, nell'argomento ADL `-profile`. ADL permette di testare un'applicazione con la versione stub o simulatore solo dell'estensione ActionScript inclusa nel pacchetto ANE. Tuttavia, in genere l'estensione corrispondente installata nel dispositivo include anche codice nativo. Per testare l'applicazione utilizzando l'estensione con il codice nativo, eseguirla sul dispositivo di destinazione.

Per ulteriori informazioni, vedete “[ADL \(AIR Debug Launcher\)](#)” a pagina 165.

Uso delle estensioni native

Se l'applicazione utilizza estensioni native, il comando ADL presenta la sintassi dell'esempio seguente:

```
adl -profile extendedTV -extdir C:\extensionDirs myApp-app.xml
```

L'esempio presuppone che:

- Il percorso del tool ADL sia nella definizione di percorso della vostra shell della riga di comando. (Vedete “[Variabili d'ambiente dei percorsi](#)” a pagina 316.)
- La directory corrente contenga i file dell'applicazione. Questi file includono i file SWF e il file descrittore dell'applicazione, in questo caso `myApp-app.xml`.
- Il parametro `-extdir` indica una directory che contiene una directory per ciascuna estensione nativa utilizzata dall'applicazione. Ognuna di queste directory contiene il file ANE *spacchettato* di un'estensione nativa. Ad esempio:

```
C:\extensionDirs
  extension1.ane
    META-INF
      ANE
        default
          library.swf
        extension.xml
        signatures.xml
    catalog.xml
    library.swf
    mimetype
  extension2.ane
    META-INF
      ANE
        default
          library.swf
        extension.xml
        signatures.xml
    catalog.xml
    library.swf
    mimetype
```

Questi file contengono una versione stub o simulatore esclusivamente dell'estensione solo per ActionScript. La versione dell'estensione che contiene il codice nativo è installata nel dispositivo AIR per TV.

Per ulteriori informazioni, vedete [Developing Native Extensions for Adobe AIR](#) (Sviluppo di estensioni native per Adobe AIR).

Immissione di comandi

ADL simula i pulsanti del telecomando di un dispositivo TV. Potete inviare l'input di questi pulsanti al dispositivo simulato utilizzando il menu visualizzato nel momento in cui ADL viene avviato utilizzando uno dei profili TV.

Dimensione dello schermo

È possibile provare l'applicazione su schermi di dimensioni differenti impostando il parametro ADL `-screenize`. Potete specificare una stringa contenente i quattro valori che rappresentano la larghezza e l'altezza degli schermi in formato normale e ingrandito.

Specificate sempre le dimensioni in pixel per il layout verticale, ovvero specificate una larghezza di valore inferiore a quello dell'altezza. Ad esempio:

```
adl -screenize 728x1024:768x1024 myApp-app.xml
```

Istruzioni trace

Quando eseguite un'applicazione TV sul desktop, l'output trace viene stampato sul debugger oppure nella finestra di terminale usata per avviare ADL.

Debug remoto con Flash Professional

Potete usare Flash Professional per eseguire il debug remoto dell'applicazione AIR per TV mentre è in esecuzione sul dispositivo di destinazione. Tuttavia, la procedura da seguire per configurare il debug remoto dipende dal dispositivo. Ad esempio, il kit "Adobe® AIR® per TV MAX 2010 Hardware Development Kit" contiene la documentazione relativa alla procedura dettagliata da seguire per tale dispositivo.

Indipendentemente dal dispositivo di destinazione, tuttavia, effettuate i seguenti passaggi per preparare il debug remoto:

- 1 Nella finestra di dialogo Impostazioni pubblicazione, nella scheda Flash, selezionate Consenti debug.
Questa opzione fa sì che Flash Professional includa le informazioni di debug in tutti i file SWF creati dal file FLA.
- 2 Nella scheda Firma della finestra delle impostazioni di Adobe AIR (Impostazioni applicazione e programma di installazione), selezionate l'opzione per preparare un file AIR Intermediate (AIRI).
Quando state ancora sviluppando l'applicazione, è sufficiente utilizzare un file AIRI, che non richiede una firma digitale.
- 3 Pubblicare l'applicazione, creando il file AIRI.

Gli ultimi passaggi riguardano l'installazione ed esecuzione dell'applicazione nel dispositivo di destinazione. Questi passaggi, tuttavia, dipendono dal dispositivo specifico.

Debug remoto con Flash Builder

Potete usare anche Flash Builder per eseguire il debug remoto dell'applicazione AIR per TV mentre è in esecuzione sul dispositivo di destinazione. Tuttavia, la procedura da seguire per effettuare il debug remoto dipende dal dispositivo.

Indipendentemente dal dispositivo di destinazione, tuttavia, effettuate i seguenti passaggi per preparare il debug remoto:

- 1 Selezionate Project > Export Release Build. Selezionate l'opzione per preparare un file AIR Intermediate (AIRI).
Quando state ancora sviluppando l'applicazione, è sufficiente utilizzare un file AIRI, che non richiede una firma digitale.
- 2 Pubblicare l'applicazione, creando il file AIRI.
- 3 Modificate il pacchetto AIRI dell'applicazione includendo i file SWF che contengono le informazioni di debug.
I file SWF che contengono le informazioni di debug si trovano nella directory del progetto Flash Builder dell'applicazione, in una directory chiamata bin-debug. Sostituite i file SWF nel pacchetto AIRI con quelli presenti nella directory bin-debug.

Su un computer di sviluppo Windows, potete eseguire questa sostituzione procedendo nel modo seguente:

- 1 Rinominate il file del pacchetto AIRI sostituendo l'estensione .airi con l'estensione .zip.
- 2 Estraiete il contenuto del file ZIP.
- 3 Sostituite i file SWF nella struttura di directory estratta con quelli in bin-debug.
- 4 Comprimate in un nuovo file ZIP i file presenti nella directory estratta.
- 5 Cambiate di nuovo l'estensione del file ZIP in .airi.

Se usate un computer di sviluppo Mac, la procedura da seguire per questa sostituzione dipende dal dispositivo. In genere, tuttavia, dovrete:

- 1 Installare il pacchetto AIRI sul dispositivo di destinazione.
- 2 Sostituire i file SWF nella directory di installazione dell'applicazione sul dispositivo di destinazione con quelli presenti nella directory bin-debug.

Ad esempio, considerate il dispositivo incluso in Adobe AIR for TV MAX 2010 Hardware Development Kit. Installate il pacchetto AIRI come descritto nella documentazione del kit. Quindi, usate telnet sulla riga di comando del computer di sviluppo Mac per accedere al dispositivo di destinazione. Sostituite i file SWF nella directory di installazione dell'applicazione in `/opt/adobe/stagecraft/apps/<nome applicazione>/` con quelli presenti nella directory `bin-debug`.

La procedura di debug remoto descritta di seguito fa riferimento a Flash Builder e al dispositivo incluso in Adobe AIR for TV MAX 2010 Hardware Development Kit.

- 1 Sul computer che esegue Flash Builder (il computer di sviluppo), eseguite AIR for TV Device Connector, fornito con MAX 2010 Hardware Development Kit. Viene indicato l'indirizzo IP del computer di sviluppo.
- 2 Sul dispositivo del kit hardware, avviate l'applicazione DevMaster, a sua volta fornita con il kit di sviluppo.
- 3 Nell'applicazione DevMaster, immettete l'indirizzo IP del computer di sviluppo così come indicato in AIR for TV Device Connector.
- 4 Nell'applicazione DevMaster, verificate che sia selezionato Enable Remote Debugging.
- 5 Chiudete l'applicazione DevMaster.
- 6 Sul computer di sviluppo, selezionate Start in AIR for TV Connector.
- 7 Sul dispositivo del kit hardware, avviate un'altra applicazione. Verificate che in AIR for TV Device Connector siano visualizzate le informazioni di traccia.

In caso contrario, il computer di sviluppo e il dispositivo del kit hardware non sono collegati. Verificate che la porta del computer di sviluppo utilizzata per le informazioni di traccia sia disponibile. Potete scegliere una porta diversa in AIR for TV Device Connector. Inoltre, verificate che il firewall consenta l'accesso alla porta selezionata.

Quindi, avviate il debugger in Flash Builder. Procedete nel modo seguente:

- 1 In Flash Builder, selezionate Run > Debug Configurations.
- 2 Dalla configurazione di debug esistente, che è per il debug locale, copiate il nome del progetto.
- 3 Nella finestra di dialogo Debug Configurations, selezionate Web Application. Quindi selezionate l'icona New Launch Configuration.
- 4 Selezionate il nome del progetto nel pannello Project.
- 5 Nella sezione URL Or Path To Launch, rimuovete il segno di spunta da Use Default. Inoltre, immettete `about:blank` nel campo di testo.
- 6 Selezionate Apply per salvare le modifiche.
- 7 Selezionate Debug per avviare il debugger Flash Builder.
- 8 Avviate l'applicazione sul dispositivo del kit hardware.

A questo punto potete usare il debugger di Flash Builder, ad esempio per impostare dei punti di interruzione ed esaminare le variabili.

Capitolo 9: Uso di estensioni native per Adobe AIR

Le estensioni native per Adobe AIR forniscono le API di ActionScript che consentono di accedere a funzionalità specifiche del dispositivo programmate nel codice nativo. Gli sviluppatori di estensioni native talvolta collaborano con i produttori di dispositivi e talvolta con sviluppatori di terze parti.

Se state sviluppando un'estensione nativa, consultate [Developing Native Extensions for Adobe AIR](#) (Sviluppo di estensioni native per Adobe AIR).

Un'estensione nativa è una combinazione di:

- Classi ActionScript
- Codice nativo

Tuttavia, in qualità di sviluppatori di applicazioni AIR che impiegano estensioni native, lavorerete solo con le classi di ActionScript.

Le estensioni native sono utili nelle seguenti situazioni:

- L'implementazione del codice nativo fornisce accesso alle funzionalità specifiche della piattaforma. Queste funzionalità specifiche della piattaforma non sono disponibili nelle classi di ActionScript integrate e non è possibile implementarle nelle classi di ActionScript specifiche dell'applicazione. L'implementazione del codice nativo è in grado di fornire tali funzionalità in quanto ha accesso a software e hardware specifici del dispositivo.
- Un'implementazione del codice nativo può talvolta risultare più veloce di un'implementazione che usa solo ActionScript.
- L'implementazione del codice nativo può fornire ad ActionScript l'accesso a codice nativo precedente.

Alcuni esempi di estensioni native si trovano nel Centro per sviluppatori Adobe. Ad esempio, un'estensione nativa fornisce alle applicazioni AIR l'accesso alla funzionalità di vibrazione di Android. Consultate [Native extensions for Adobe AIR](#) (Estensioni native per Adobe AIR).

File ANE (AIR Native Extension)

Quando impacchettano le estensioni native, gli sviluppatori creano dei file ANE. I file ANE sono file di archivio che contengono le librerie e le risorse necessarie per l'estensione nativa.

Tenete presente che per alcuni dispositivi il file ANE contiene la libreria dei codici nativi impiegati dall'estensione nativa. Tuttavia, per i restanti dispositivi, la libreria dei codici nativi è installata all'interno del dispositivo. In alcuni casi, l'estensione nativa non ha alcun codice nativo per un particolare dispositivo e viene implementata solo con ActionScript.

In qualità di sviluppatori di applicazioni AIR, potete utilizzare il file ANE come segue:

- Inserite il file ANE nel percorso di libreria dell'applicazione, esattamente come inserite un file SWC nel percorso di libreria. Questa operazione consente all'applicazione di fare riferimento alle classi ActionScript dell'estensione.

Nota: durante la compilazione dell'applicazione, accertatevi di utilizzare i collegamenti dinamici al file ANE. Se utilizzate Flash Builder, specificate External nel pannello ActionScript Builder Path Properties; se invece utilizzate la riga di comando, digitate `-external-library-path`.

- Inserite il file ANE nell'applicazione AIR.

Estensioni native e la classe ActionScript NativeProcess

ActionScript 3.0 fornisce la classe NativeProcess, che consente a un'applicazione AIR di eseguire processi nativi nel sistema operativo host. Questa funzionalità è simile alle estensioni native, che consentono l'accesso a funzioni e librerie specifiche della piattaforma. Quando dovete decidere se utilizzare la classe NativeProcess o un'estensione nativa, considerate quanto segue:

- Solo il profilo AIR `extendedDesktop` supporta la classe NativeProcess. Di conseguenza, per le applicazioni con i profili AIR `mobileDevice` e `extendedMobileDevice`, le estensioni native sono la sola scelta possibile.
- Gli sviluppatori di estensioni native spesso forniscono implementazioni native per varie piattaforme, tuttavia l'API di ActionScript che forniscono è generalmente la stessa per tutte le piattaforme. Se si usa la classe NativeProcess, il codice ActionScript per avviare il processo nativo può variare da piattaforma a piattaforma.
- La classe NativeProcess avvia un processo separato, mentre un'estensione nativa esegue lo stesso processo dell'applicazione AIR. Di conseguenza, se avete problemi di blocco del codice, l'impiego della classe NativeProcess è meno rischioso. Tuttavia, un processo separato potrebbe implicare l'implementazione di una gestione delle comunicazioni tra processi.

Estensioni native e librerie delle classi ActionScript (file SWC)

Un file SWC è una libreria delle classi ActionScript in formato di archivio. Il file SWC contiene un file SWF e altri file di risorse. Il file SWC è un sistema pratico per condividere classi di ActionScript, anziché codice ActionScript individuale e file di risorse.

Un pacchetto di estensioni native è un file ANE. Come il file SWC, anche il file ANE è una libreria di classi di ActionScript contenente un file SWF e altri file di risorse in formato di archivio. Tuttavia, la differenza sostanziale tra un file ANE e un file SWC è che solo il file ANE può contenere una libreria di codici nativi.

***Nota:** durante la compilazione dell'applicazione, accertatevi di utilizzare i collegamenti dinamici al file ANE. Se utilizzate Flash Builder, specificate External nel pannello ActionScript Builder Path Properties; se invece utilizzate la riga di comando, digitate `-external-library-path`.*

Altri argomenti presenti nell' Aiuto

[Informazioni sui file SWC](#)

Dispositivi supportati

A partire da AIR 3, potete usare le estensioni native nelle applicazioni per i seguenti dispositivi:

- Dispositivi Android, a partire da Android 2.2
- Dispositivi iOS, a partire da iOS 4.0
- iOS Simulator, a partire da AIR 3.3

- Blackberry PlayBook
- Dispositivi desktop Windows che supportano AIR 3.0
- Dispositivi desktop Mac OS X che supportano AIR 3.0

Spesso, una stessa estensione nativa può essere destinata a più piattaforme. Il file ANE dell'estensione contiene librerie native e di ActionScript per ogni piattaforma supportata. In genere, le librerie di ActionScript presentano le stesse interfacce pubbliche per tutte le piattaforme. Le librerie native sono necessariamente differenti.

Talvolta un'estensione nativa supporta una piattaforma predefinita. L'implementazione della piattaforma predefinita presenta solo codice ActionScript e nessun codice nativo. Se avete confezionato un'applicazione per una piattaforma non specificamente supportata dall'estensione, l'applicazione userà l'implementazione predefinita quando viene eseguita. Ad esempio, prendete in considerazione un'estensione che fornisce una funzione applicabile unicamente a dispositivi mobili. L'estensione può anche fornire un'implementazione predefinita che un'applicazione desktop può usare per simulare la funzione.

Profili dispositivo supportati

I seguenti profili AIR supportano le estensioni native:

- `extendedDesktop`, a partire da AIR 3.0
- `mobileDevice`, a partire da AIR 3.0
- `extendedMobileDevice`, a partire da AIR 3.0

Altri argomenti presenti nell' Aiuto

[Supporto del profilo AIR](#)

Elenco delle operazioni da eseguire per utilizzare un'estensione nativa

Per utilizzare un'estensione nativa nella vostra applicazione, dovete eseguire le seguenti operazioni:

- 1 Dichiarare l'estensione nel file descrittore dell'applicazione.
- 2 Includere il file ANE nel percorso di libreria dell'applicazione.
- 3 Creare il pacchetto dell'applicazione.

Dichiarazione dell'estensione nel file descrittore dell'applicazione

Tutte le applicazioni AIR presentano un file descrittore dell'applicazione. Quando un'applicazione usa un'estensione nativa, il file descrittore dell'applicazione include l'elemento `<extensions>`. Ad esempio:

```
<extensions>
  <extensionID>com.example.Extension1</extensionID>
  <extensionID>com.example.Extension2</extensionID>
</extensions>
```

L'elemento `extensionID` ha lo stesso valore dell'elemento `id` nel file descrittore dell'estensione. Quest'ultimo è un file XML denominato `extension.xml`, impacchettato nel file ANE. Potete usare uno strumento di estrazione archivi per analizzare il file `extension.xml`.

Inserimento del file ANE nel percorso di libreria dell'applicazione

Per compilare un'applicazione che impiega un'estensione, dovete includere il file ANE nel percorso di libreria.

Uso del file ANE con Flash Builder

Se l'applicazione fa uso di un'estensione nativa, includete il file ANE dell'estensione nativa nel percorso di libreria. Quindi, utilizzate Flash Builder per compilare il codice ActionScript.

Effettuate i passaggi seguenti (viene usato Flash Builder 4.5.1):

- 1 Cambiate l'estensione del nome del file ANE da `.ane` a `.swc`. Questo passaggio è necessario per consentire a Flash Builder di trovare il file.
- 2 Selezionate **Project > Properties** nel progetto Flash Builder.
- 3 Selezionate **Flex Build Path** nella finestra di dialogo **Properties**.
- 4 Nella scheda **Library Path**, selezionate **Add SWC...**
- 5 Individuate il file SWC desiderato e selezionate **Open**.
- 6 Selezionate **OK** nella finestra di dialogo **Add SWC...**
Il file ANE ora appare nella scheda **Library Path** della finestra di dialogo **Properties**.
- 7 Espandete la voce corrispondente al file SWC. Fate doppio clic su **Link Type** per aprire la finestra di dialogo **Library Path Item Options**.
- 8 Nella finestra di dialogo **Library Path Item Options**, cambiate l'impostazione di **Link Type** in **External**.

Ora potete compilare l'applicazione selezionando, ad esempio, **Project > Build Project**.

Uso del file ANE con Flash Professional

Se l'applicazione fa uso di un'estensione nativa, includete il file ANE dell'estensione nativa nel percorso di libreria. Quindi, utilizzate Flash Professional CS5.5 per compilare il codice ActionScript. Procedete nel modo seguente:

- 1 Cambiate l'estensione del nome del file ANE da `.ane` a `.swc`. Questo passaggio è necessario per consentire a Flash Professional di trovare il file.
- 2 Selezionate **File > Impostazioni ActionScript** nel file FLA.
- 3 Selezionate la scheda **Percorso libreria** nella finestra di dialogo **Impostazioni avanzate ActionScript 3.0**.
- 4 Fate clic sul pulsante **Specifica file SWC**.
- 5 Individuate il file SWC desiderato e selezionate **Apri**.

Il file SWC ora appare nella scheda Percorso libreria della finestra di dialogo Impostazioni avanzate ActionScript 3.0.

- 6 Con il file SWC selezionato, selezionate il pulsante Imposta opzioni di concatenamento per una libreria.
- 7 Nella finestra di dialogo Opzioni elemento percorso di libreria, cambiate l'impostazione di Tipo di collegamento in Esterno.

Creazione del pacchetto di un'applicazione che usa estensioni native

Utilizzate ADT per creare il pacchetto di un'applicazione che usa estensioni native. Non potete creare il pacchetto dell'applicazione utilizzando Flash Professional CS5.5 o Flash Builder 4.5.1.

Per maggiori informazioni sull'uso di ADT, consultate il documento [AIR Developer Tool \(ADT\)](#).

Ad esempio, il seguente comando ADT consente di creare un file DMG (un file di installazione nativo per Mac OS X) per un'applicazione che usa estensioni native:

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    index.html resources
    -extdir extensionsDir
```

Il seguente comando consente di creare un pacchetto APK per un dispositivo Android:

```
adt -package
    -target apk
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
    -extdir extensionsDir
```

Il seguente comando consente di creare un pacchetto iOS per un'applicazione di iPhone:

```
adt -package
    -target ipa-ad-hoc
    -storetype pkcs12 -keystore ../AppleDistribution.p12
    -provisioning-profile AppleDistribution.mobileprofile
    myApp.ipa
    myApp-app.xml
    myApp.swf icons Default.png
    -extdir extensionsDir
```

Tenete presente quanto segue:

- Utilizzate un tipo di pacchetto di installazione nativo.
- Specificate la directory dell'estensione.
- Verificate che il file ANE supporti il dispositivo di destinazione dell'applicazione.

Uso di un tipo di pacchetto di installazione nativo

Il pacchetto dell'applicazione deve essere un programma di installazione nativo. Non potete creare un pacchetto AIR (un pacchetto .air) multipiattaforma per un'applicazione che usa estensioni native, in quanto le estensioni native generalmente contengono codice nativo. Tuttavia, in genere le estensioni native supportano più piattaforme native con le stesse API di ActionScript. In questi casi, potete usare lo stesso file ANE in pacchetti di installazione nativi differenti.

La tabella che segue riepiloga i valori da usare per l'opzione `-target` del comando ADT:

Piattaforma di destinazione dell'applicazione	-target
Dispositivi desktop Mac OS X o Windows	-target native -target bundle
Android	-target apk o altre destinazioni per pacchetti Android
iOS	-target ipa-ad-hoc o altre destinazioni per pacchetti iOS
iOS Simulator	-target ipa-test-interpretor-simulator -target ipa-debug-interpretor-simulator

Indicazione della directory dell'estensione

Utilizzate l'opzione ADT `-extdir` per indicare ad ADT la directory che contiene le estensioni native (file ANE).

Per maggiori dettagli su questa opzione, consultate “[Opzioni per file e percorsi](#)” a pagina 188.

Verifica che il file ANE supporti il dispositivo di destinazione dell'applicazione

Quando fornisce un file ANE, lo sviluppatore dell'estensione nativa vi informa sulle piattaforme supportate dall'estensione. Potete anche usare uno strumento di estrazione archivio per analizzare il contenuto del file ANE. I file estratti includono una directory per ciascuna piattaforma supportata.

Conoscere quali piattaforme sono supportate dall'estensione è importante per la creazione del pacchetto dell'applicazione che impiega il file ANE. Considerate le seguenti regole:

- Per creare un pacchetto di applicazione Android, è necessario che il file ANE includa la piattaforma `Android-ARM`. In alternativa, è necessario che il file ANE includa la piattaforma predefinita e almeno un'altra piattaforma.
- Per creare un pacchetto di applicazione iOS, è necessario che il file ANE includa la piattaforma `iPhone-ARM`. In alternativa, è necessario che il file ANE includa la piattaforma predefinita e almeno un'altra piattaforma.
- Per creare un pacchetto di applicazione iOS Simulator, è necessario che il file ANE includa la piattaforma `iPhone-x86`.
- Per creare un pacchetto di applicazione Mac OS X, è necessario che il file ANE includa la piattaforma `MacOS-x86`. In alternativa, è necessario che il file ANE includa la piattaforma predefinita e almeno un'altra piattaforma.
- Per creare un pacchetto di applicazione Windows, è necessario che il file ANE includa la piattaforma `Windows-x86`. In alternativa, è necessario che il file ANE includa la piattaforma predefinita e almeno un'altra piattaforma.

Capitolo 10: Compilatori ActionScript

Per poter includere codice ActionScript e MXML in un'applicazione AIR, tale codice deve prima essere compilato. Se usate un ambiente IDE (Integrated Development Environment), quale Adobe Flash Builder o Adobe Flash Professional, l'IDE gestisce la compilazione "dietro le quinte". Tuttavia, potete anche chiamare i compilatori ActionScript dalla riga di comando per creare i file SWF se non utilizzate un ambiente IDE oppure usate uno script di compilazione.

Informazioni sugli strumenti della riga di comando AIR in Flex SDK

Ognuno degli strumenti della riga di comando che utilizzate per creare un'applicazione Adobe AIR chiama il corrispondente strumento utilizzato per creare le applicazioni:

- amxmlc chiama mxmmlc per compilare le classi dell'applicazione
- acompc chiama compc per compilare le classi di librerie e componenti
- aasdoc chiama asdoc per generare i file della documentazione a partire dai commenti del codice sorgente

L'unica differenza tra le versioni Flex e AIR di queste utilità è che le versioni AIR caricano le opzioni di configurazione dal file air-config.xml invece che dal file flex-config.xml.

Gli strumenti Flex SDK e le relative opzioni della riga di comando sono descritti in modo dettagliato nella [documentazione Flex](#). In questa documentazione gli strumenti di Flex SDK vengono descritti a un livello di base per consentirvi di acquisire familiarità con il loro utilizzo ed evidenziare le differenze tra la creazione di applicazioni Flex e la creazione di applicazioni AIR.

Altri argomenti presenti nell'Aiuto

“[Creazione della prima applicazione AIR desktop con Flex SDK](#)” a pagina 39

Impostazione del compilatore

Per specificare le opzioni di compilazione, in genere si utilizzano sia la riga di comando che uno o più file di configurazione. Il file di configurazione globale di Flex SDK contiene i valori predefiniti che vengono utilizzati a ogni esecuzione dei compilatori. Potete modificare questo file in base alle esigenze del vostro ambiente di sviluppo. La directory frameworks dell'installazione di Flex SDK contiene due file di configurazione globali per Flex. Il file air-config.xml viene utilizzato quando si esegue il compilatore amxmlc. Questo file configura il compilatore per AIR includendo le librerie AIR. Il file flex-config.xml viene utilizzato quando si esegue mxmmlc.

I valori di configurazione predefiniti sono utili per acquisire dimestichezza con il funzionamento di Flex e AIR. Prima di iniziare un progetto di grandi dimensioni, tuttavia, è opportuno esaminare più da vicino le opzioni disponibili. A livello di progetto potete utilizzare un file di configurazione locale, che ha la precedenza sui valori globali, per definire valori specifici del progetto per le opzioni di compilazione.

Nota: per le applicazioni AIR non vengono utilizzate opzioni di compilazione specifiche, ma durante la compilazione di un'applicazione AIR è necessario fare riferimento alle librerie AIR. In genere i riferimenti a queste librerie vengono specificati in un file di configurazione a livello di progetto, in un file di uno strumento di creazione come Ant o direttamente nella riga di comando.

Compilazione di file di origine MXML e ActionScript per AIR

Potete compilare le risorse Adobe® ActionScript® 3.0 e MXML dell'applicazione AIR con il compilatore MXML della riga di comando (amxmlc). Non dovete compilare applicazioni basate su HTML. Per compilare un file SWF in Flash Professional, pubblicate il filmato in un file SWF.

La sintassi di base da riga di comando per l'uso di amxmlc è la seguente:

```
amxmlc [compiler options] -- MyAIRApp.mxml
```

dove *[compiler options]* specifica le opzioni della riga di comando utilizzate per la compilazione dell'applicazione AIR.

Il comando amxmlc chiama il compilatore standard Flex mxmlc con un parametro aggiuntivo, vale a dire `+configname=air`. Questo parametro istruisce il compilatore all'utilizzo del file `air-config.xml` anziché del file `flex-config.xml`. Per altri aspetti, l'impiego di amxmlc è identico a quello di mxmlc.

Il compilatore carica il file di configurazione `air-config.xml` specificando le librerie AIR e Flex solitamente richieste per la compilazione di un'applicazione AIR. È inoltre possibile utilizzare un file di configurazione locale a livello di progetto per ignorare o aggiungere ulteriori opzioni alla configurazione globale. Solitamente, il sistema più semplice per creare un file di configurazione locale consiste nel modificare una copia della versione globale. È possibile caricare il file locale con l'opzione `-load-config`:

-load-config=project-config.xml Ignora le opzioni globali.

-load-config+=project-config.xml Aggiunge ulteriori valori alle opzioni globali che accettano più del valore, come ad esempio l'opzione `-library-path`. Le opzioni globali che accettano solo un valore vengono ignorate.

Se si utilizza una speciale convenzione di denominazione per il file di configurazione locale, il compilatore amxmlc carica automaticamente il file locale. Ad esempio, se il file MXML principale è `RunningMan.mxml`, denominate il file di configurazione locale `RunningMan-config.xml`. A questo punto, per compilare l'applicazione, dovete solo digitare:

```
amxmlc RunningMan.mxml
```

`RunningMan-config.xml` viene caricato automaticamente dal momento che il relativo nome file corrisponde a quello del file MXML compilato.

Esempi di amxmlc

Negli esempi seguenti viene illustrato l'utilizzo del compilatore amxmlc. (Devono essere compilate solo le risorse ActionScript e MXML dell'applicazione.)

Compilare un file MXML di AIR:

```
amxmlc myApp.mxml
```

Compilare e impostare il nome di output:

```
amxmlc -output anApp.swf -- myApp.mxml
```

Compilare un file ActionScript di AIR:

```
amxmlc MyApp.as
```

Specificare un file di configurazione del compilatore:

```
amxmlc -load-config config.xml -- MyApp.mxml
```

Aggiungere ulteriori opzioni da un altro file di configurazione:

```
amxmlc -load-config+=moreConfig.xml -- MyApp.mxml
```

Aggiungere librerie sulla riga di comando, in aggiunta alle librerie già presenti nel file di configurazione:

```
amxmlc -library-path+=/libs/libOne.swc,/libs/libTwo.swc -- MyApp.mxml
```

Compilare un file MXML di AIR senza utilizzare un file di configurazione (Win):

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, ^  
[AIR SDK]/frameworks/libs/air/airframework.swc, ^  
-library-path [Flex SDK]/frameworks/libs/framework.swc ^  
-- MyApp.mxml
```

Compilare un file MXML di AIR senza utilizzare un file di configurazione (Mac OS X o Linux):

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, \  
[AIR SDK]/frameworks/libs/air/airframework.swc, \  
-library-path [Flex 3 SDK]/frameworks/libs/framework.swc \  
-- MyApp.mxml
```

Compilare un file MXML di AIR per utilizzare una libreria condivisa a livello di runtime:

```
amxmlc -external-library-path+=../lib/myLib.swc -runtime-shared-libraries=myrsl.swf --  
MyApp.mxml
```

Compilare un file MXML di AIR per utilizzare un file ANE (utilizzare `-external-library-path` per il file ANE):

```
amxmlc -external-library-path+=../lib/myANE.ane -output=myAneApp.swf -- myAneApp.mxml
```

Compilazione da Java (con il percorso della classe impostato in modo da includere `mxmlc.jar`):

```
java flex2.tools.Compiler +flexlib [Flex SDK 3]/frameworks +configname=air [additional  
compiler options] -- MyApp.mxml
```

L'opzione `flexlib` identifica il percorso della directory delle strutture di Flex SDK, consentendo al compilatore di individuare il file `flex_config.xml`.

Compilazione da Java (senza impostazione del percorso della classe):

```
java -jar [Flex SDK 2]/lib/mxmlc.jar +flexlib [Flex SDK 3]/frameworks +configname=air  
[additional compiler options] -- MyApp.mxml
```

Per richiamare il compilatore con Apache Ant (nell'esempio viene utilizzata un'attività Java per eseguire `mxmlc.jar`):

```
<property name="SDK_HOME" value="C:/Flex46SDK"/>
<property name="MAIN_SOURCE_FILE" value="src/myApp.mxml"/>
<property name="DEBUG" value="true"/>
<target name="compile">
  <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
    <arg value="-debug=${DEBUG}"/>
    <arg value="+flexlib=${SDK_HOME}/frameworks"/>
    <arg value="+configname=air"/>
    <arg value="-file-specs=${MAIN_SOURCE_FILE}"/>
  </java>
</target>
```

Compilazione di un componente o libreria di codice AIR (Flex)

Il compilatore di componenti `acompc` consente di compilare librerie AIR e componenti indipendenti. Il compilatore di componenti `acompc` si comporta come il compilatore `amxmlc`, con le seguenti eccezioni:

- È necessario specificare quali classi della base di codice desiderate includere nella libreria o nel componente.
- `acompc` non cerca automaticamente un file di configurazione locale. Per utilizzare un file di configurazione del progetto, è necessario utilizzare l'opzione `-load-config`.

Il comando `acompc` chiama il compilatore di componenti Flex standard `compc`, ma ne carica le opzioni di configurazione dal file `air-config.xml` invece di `flex-config.xml`.

File di configurazione del compilatore di componenti

Utilizzando un file di configurazione locale è possibile evitare di digitare (e quindi di digitare in modo errato) il percorso di origine e i nomi delle classi sulla riga di comando. Aggiungete l'opzione `-load-config` alla riga di comando di `acompc` per caricare il file di configurazione locale.

Nell'esempio seguente viene illustrata una configurazione per la creazione di una libreria con due classi, `ParticleManager` e `Particle`, entrambe presenti nel pacchetto `com.adobe.samples.particles`. I file si trovano nella cartella `source/com/adobe/samples/particles`.

```
<flex-config>
  <compiler>
    <source-path>
      <path-element>source</path-element>
    </source-path>
  </compiler>
  <include-classes>
    <class>com.adobe.samples.particles.ParticleManager</class>
    <class>com.adobe.samples.particles.Particle</class>
  </include-classes>
</flex-config>
```

Per compilare la libreria utilizzando il file di configurazione denominato `ParticleLib-config.xml`, digitate:

```
acompc -load-config ParticleLib-config.xml -output ParticleLib.swc
```

Per eseguire lo stesso comando interamente sulla riga di comando, digitate:


```
acompc -source-path source -include-classes com.adobe.samples.particles.Particle  
com.adobe.samples.particles.ParticleManager -output ParticleLib.swc
```

(Digitate il comando per intero sulla riga di comando, oppure utilizzate il carattere di continuazione della riga per la shell di comando.)

Esempi di `acompc`

In questi esempi si presuppone l'impiego di un file di configurazione denominato `myLib-config.xml`.

Compilare una libreria o un componente AIR:

```
acompc -load-config myLib-config.xml -output lib/myLib.swc
```

Compilare una libreria condivisa di runtime:

```
acompc -load-config myLib-config.xml -directory -output lib
```

Prima di eseguire il comando, la cartella `lib` deve essere esistente e vuota.

Capitolo 11: ADL (AIR Debug Launcher)

Con ADL (AIR Debug Launcher) potete eseguire nel corso dello sviluppo applicazioni basate sia su SWF che su HTML. Mediante ADL potrete eseguire un'applicazione senza prima effettuare la creazione del pacchetto e installarla. Per impostazione predefinita in ADL viene utilizzato un runtime incluso con SDK, pertanto per utilizzare ADL non occorrerà installare separatamente il runtime.

ADL stampa istruzioni trace ed errori di runtime nell'output standard, ma non supporta punti di interruzione o altre funzionalità di debug. Potete utilizzare il debugger di Flash (o un ambiente di sviluppo integrato come Flash Builder) per problemi di debug complessi.

***Nota:** Se le istruzioni `trace()` non vengono visualizzate sulla console, verificate di non aver specificato `ErrorReportingEnable` o `TraceOutputFileEnable` nel file `mm.cfg`. Per maggiori informazioni sulle posizioni di questo file a seconda della piattaforma, vedete [Editing the mm.cfg file](#).*

Poiché AIR supporta il debug direttamente, non è necessario utilizzare una versione di debug del runtime (come avviene invece in Adobe® Flash® Player). Per eseguire il debug dalla riga di comando si utilizzano Flash Debugger e ADL (AIR Debug Launcher).

Flash Debugger si trova nella directory Flex SDK. Le versioni native, ad esempio `fdb.exe` in Windows, si trovano nella sottodirectory `bin`. La versione Java si trova nella sottodirectory `lib`. AIR Debug Launcher, `adl.exe`, si trova nella directory `bin` dell'installazione di Flex SDK (non esiste una versione Java separata).

***Nota:** non è possibile avviare un'applicazione AIR direttamente con `fdb` perché `fdb` tenta di avviarla con Flash Player. Lasciate che l'applicazione AIR si connetta a una sessione `fdb` in esecuzione.*

Utilizzo di ADL

Per eseguire un'applicazione mediante ADL, utilizzate lo schema seguente:

```
adl application.xml
```

application.xml è il file descrittore dell'applicazione per questa applicazione.

La sintassi completa di ADL è la seguente:

```
adl [-runtime runtime-directory]
    [-pubid publisher-id]
    [-nodebug]
    [-atlogin]
    [-profile profileName]
    [-screensize value]
    [-extdir extension-directory]
    application.xml
    [root-directory]
    [-- arguments]
```

(Le voci in parentesi quadre, [], sono opzionali.)

-runtime runtime-directory Specifica la directory contenente il runtime da utilizzare. Se non è specificata, viene utilizzata la directory di runtime nello stesso SDK in cui viene utilizzato il programma ADL. Se spostate ADL fuori dalla relativa cartella SDK, dovete specificare la directory di runtime. In Windows e Linux, specificate la directory contenente la directory `Adobe AIR`. In Mac OS X, specificate la directory contenente `Adobe AIR.framework`.

-pubid publisher-id Assegna il valore specificato come ID editore dell'applicazione AIR per questa esecuzione. Specificando un ID editore temporaneo è possibile testare le funzionalità di un'applicazione AIR, ad esempio la comunicazione su una connessione locale, che utilizzano l'ID editore per l'identificazione univoca dell'applicazione. A partire da AIR 1.5.3, potete anche specificare l'ID editore nel file descrittore dell'applicazione (senza usare questo parametro).

***Nota:** a partire da AIR 1.5.3, l'ID editore non viene più automaticamente calcolato e assegnato a un'applicazione AIR. Potete specificare un ID editore quando create un aggiornamento di un'applicazione AIR esistente, ma per le nuove applicazioni l'ID editore non è necessario e non deve essere specificato.*

-nodebug Disattiva il supporto per il debug. Se utilizzata, il processo dell'applicazione non consente la connessione al debugger Flash e le finestre di dialogo per le eccezioni non gestite vengono eliminate. (Le istruzioni trace vengono comunque stampate nella finestra della console.) La disattivazione del debug consente di eseguire l'applicazione più rapidamente e di emulare più fedelmente la modalità di esecuzione di un'applicazione installata.

-atlogin Simula l'avvio dell'applicazione al login. Questo flag consente di provare la logica dell'applicazione che viene eseguita solo quando un'applicazione è impostata per avviarsi quando l'utente effettua il login. Quando si utilizza `-atlogin`, la proprietà `reason` dell'oggetto `InvokeEvent` inviato all'applicazione sarà `login` anziché `standard` (a meno che l'applicazione non sia già in esecuzione).

-profile profileName ADL esegue il debug dell'applicazione utilizzando il profilo specificato. Il dato `profileName` può essere uno dei seguenti valori:

- desktop
- extendedDesktop
- mobileDevice

Se il descrittore dell'applicazione include un elemento `supportedProfiles`, il profilo che specificate con `-profile` deve essere membro dell'elenco di profili supportati. Se il flag `-profile` non viene utilizzato, il primo profilo nel descrittore dell'applicazione viene utilizzato come profilo attivo. Se il descrittore dell'applicazione non include l'elemento `supportedProfiles` e non utilizzate il flag `-profile`, viene usato il profilo `desktop`.

Per ulteriori informazioni, vedete “[supportedProfiles](#)” a pagina 248 e “[Profili dispositivo](#)” a pagina 254.

-screensize valore Il formato schermo simulato da utilizzare per l'esecuzione di applicazioni nel profilo `mobileDevice` sul desktop. Specificate il formato dello schermo come tipo di schermo predefinito oppure immettendo le dimensioni in pixel della larghezza e altezza normali per il layout verticale, più la larghezza e altezza a schermo intero. Per specificare il valore in base al tipo, utilizzate uno dei seguenti tipi di schermo predefiniti:

Tipo di schermo	Larghezza x altezza normali	Larghezza x altezza a schermo intero
480	720 x 480	720 x 480
720	1280 x 720	1280 x 720
1080	1920 x 1080	1920 x 1080
Droid	480 x 816	480 x 854
FWQVGA	240 x 432	240 x 432
FWVGA	480 x 854	480 x 854
HVGA	320 x 480	320 x 480
iPad	768 x 1004	768 x 1024
iPadRetina	1536 x 2008	1536 x 2048

Tipo di schermo	Larghezza x altezza normali	Larghezza x altezza a schermo intero
iPhone	320 x 460	320 x 480
iPhoneRetina	640 x 920	640 x 960
iPhone5Retina	640 x 1096	640 x 1136
iPhone6	750 x 1294	750 x 1334
iPhone6Plus	1242 x 2148	1242 x 2208
iPod	320 x 460	320 x 480
iPodRetina	640 x 920	640 x 960
iPod5Retina	640 x 1096	640 x 1136
NexusOne	480 x 762	480 x 800
QVGA	240 x 320	240 x 320
SamsungGalaxyS	480 x 762	480 x 800
SamsungGalaxyTab	600 x 986	600 x 1024
WQVGA	240 x 400	240 x 400
WVGA	480 x 800	480 x 800

Per specificare le dimensioni in pixel dello schermo, utilizzate il formato seguente:

```
widthXheight:fullscreenWidthXfullscreenHeight
```

Specificate sempre le dimensioni in pixel per il layout verticale, ovvero specificate una larghezza di valore inferiore a quello dell'altezza. Ad esempio, lo schermo di NexusOne può essere specificato con:

```
-screensize 480x762:480x800
```

-extdir extension-directory La directory in cui il runtime deve cercare le estensioni native. La directory contiene una sottodirectory per ciascuna estensione nativa utilizzata dall'applicazione. Ognuna di queste sottodirectory contiene il file ANE *spacchettato* di un'estensione. Ad esempio:

```
C:\extensionDirs\  
  extension1.ane\  
    META-INF\  
      ANE\  
        Android-ARM\  
          library.swf  
          extension1.jar  
          extension.xml  
          signatures.xml  
        catalog.xml  
        library.swf  
        mimetype  
  extension2.ane\  
    META-INF\  
      ANE\  
        Android-ARM\  
          library.swf  
          extension2.jar  
          extension.xml  
          signatures.xml  
        catalog.xml  
        library.swf  
        mimetype
```

Quando usate il parametro `-extdir`, prendete in considerazione quanto segue:

- Il comando ADT richiede che ciascuna delle directory specificate abbia l'estensione `.ane`. Tuttavia, la parte del nome file prima del suffisso `“.ane”` può essere qualsiasi nome file valido. *Non* è necessario che corrisponda al valore dell'elemento `extensionID` del file descrittore dell'applicazione.
- Potete specificare il parametro `-extdir` più di una volta.
- L'uso del parametro `-extdir` varia a seconda che si utilizzi il tool ADT o ADL. In ADT, il parametro specifica una directory contenente file ANE.
- Potete anche usare la variabile ambientale `AIR_EXTENSION_PATH` per specificare le directory dell'estensione. Vedete [“Variabili d'ambiente ADT”](#) a pagina 195.

application.xml Il file descrittore dell'applicazione. Vedete [“File descrittori delle applicazioni AIR”](#) a pagina 212. Il descrittore dell'applicazione è l'unico parametro richiesto da ADL e, nella maggior parte dei casi, l'unico necessario.

root-directory Specifica la directory principale dell'applicazione da eseguire. Se il valore non viene specificato, si utilizza la directory contenente il file descrittore dell'applicazione.

--arguments Qualsiasi stringa di caratteri che appare dopo `--` viene trasmessa all'applicazione come argomento della riga di comando.

Nota: quando si avvia un'applicazione AIR già in esecuzione, non viene avviata una nuova istanza dell'applicazione, ma viene inviato un evento `invoke` all'istanza in esecuzione.

Esempi ADL

Eseguire un'applicazione nella directory corrente:

```
adl myApp-app.xml
```

Eseguire un'applicazione nella sottodirectory della directory corrente:

```
adl source/myApp-app.xml release
```

Eseguire un'applicazione e passare due argomenti della riga di comando, "tick" e "tock":

```
adl myApp-app.xml -- tick tock
```

Eseguire un'applicazione utilizzando un runtime specifico:

```
adl -runtime /AIRSDK/runtime myApp-app.xml
```

Eseguire un'applicazione senza il supporto del debug:

```
adl -nodebug myApp-app.xml
```

Eseguire un'applicazione nel profilo dispositivo mobile e simulare il formato schermo NexusOne:

```
adl -profile mobileDevice -screenize NexusOne myMobileApp-app.xml
```

Eseguire un'applicazione utilizzando Apache Ant per l'esecuzione (i percorsi mostrati nell'esempio sono per Windows):

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
<property name="APP_ROOT" value="c:/dev/MyApp/bin-debug"/>
<property name="APP_DESCRIPTOR" value="${APP_ROOT}/myApp-app.xml"/>

<target name="test">
  <exec executable="${ADL}">
    <arg value="${APP_DESCRIPTOR}"/>
    <arg value="${APP_ROOT}"/>
  </exec>
</target>
```

Codici di errore e di uscita di ADL

Nella tabella seguente vengono descritti i codici di uscita stampati da ADL:

Codice di uscita	Descrizione
0	Avvio riuscito. Viene effettuata l'uscita da ADL dopo l'uscita dall'applicazione AIR.
1	Chiamata riuscita di un'applicazione AIR già in esecuzione. Uscita immediata da ADL.
2	Errore di utilizzo. Gli argomenti forniti ad ADL non sono corretti.
3	Impossibile trovare il runtime.
4	Impossibile avviare il runtime. Spesso ciò accade perché la versione specificata nell'applicazione non corrisponde a quella del runtime.
5	Si è verificato un errore. Motivo sconosciuto.
6	Impossibile trovare il file descrittore dell'applicazione.
7	Il contenuto del descrittore dell'applicazione non è valido. Di solito quest'errore indica che il file XML non è stato formato correttamente.
8	Impossibile trovare il file del contenuto dell'applicazione principale (specificato nell'elemento <content> del file descrittore dell'applicazione).

Codice di uscita	Descrizione
9	Il file del contenuto dell'applicazione principale non è un file SWF o HTML valido.
10	L'applicazione non supporta il profilo specificato nell'opzione -profile.
11	L'argomento -screenize non è supportato nel profilo corrente.

Capitolo 12: ADT (AIR Developer Tool)

Lo strumento ADT (AIR Developer Tool) è un tool multi-funzione della riga di comando utilizzato per lo sviluppo di applicazioni AIR. Potete usare ADT per le seguenti operazioni:

- Compilare un'applicazione AIR in un file di installazione .air
- Compilare un'applicazione AIR come programma di installazione nativo, ad esempio .exe per Windows, .ipa per iOS oppure .apk per Android
- Compilare un'estensione nativa come file di estensione nativa AIR (ANE, AIR Native Extension)
- Firmare un'applicazione AIR con un certificato digitale
- Sostituire (migrare) la firma digitale utilizzata per gli aggiornamenti delle applicazioni
- Determinare i dispositivi connessi a un computer
- Creare un certificato digitale autofirmato di firma del codice
- Installare, avviare e disinstallare da remoto un'applicazione su un dispositivo mobile
- Installare e disinstallare da remoto il runtime AIR su un dispositivo mobile

ADT è un programma Java incluso in [AIR SDK](#). Per utilizzarlo è necessario avere Java 1.5 o successivo. L'SDK include un file di script che consente di chiamare il tool ADT. Per utilizzare questo script, la posizione del programma Java deve essere inclusa nella variabile d'ambiente dei percorsi. Se la directory `bin` di AIR SDK è a sua volta indicata nella variabile d'ambiente dei percorsi, potete semplicemente digitare `adt` sulla riga di comando, con gli argomenti appropriati, per chiamare ADT. (Se non sapete come impostare la variabile d'ambiente dei percorsi, vedete la documentazione del sistema operativo. Inoltre, le procedure per l'impostazione del percorso sulla maggior parte dei computer sono descritte in “[Variabili d'ambiente dei percorsi](#)” a pagina 316.

L'uso di ADT richiede almeno 2 GB di memoria del sistema. Se la quantità di memoria è inferiore, ADT potrebbe esaurire la memoria disponibile, specialmente durante la compilazione di applicazioni per iOS.

Presupponendo che sia Java che la directory `bin` di AIR SDK siano inclusi nella variabile dei percorsi, potete eseguire ADT utilizzando la seguente sintassi di base:

```
adt -command options
```

Nota: nella maggior parte degli ambienti di sviluppo integrati, inclusi Adobe Flash Builder e Adobe Flash Professional, è possibile eseguire la compilazione e la firma di applicazioni AIR. In genere non è necessario utilizzare ADT per queste attività comuni se già utilizzate un ambiente di sviluppo di questo tipo. Potrebbe tuttavia essere necessario utilizzarlo come tool della riga di comando per funzioni non supportate dall'ambiente di sviluppo integrato. Inoltre, potete usare ADT come tool della riga di comando nell'ambito di un processo di compilazione automatizzato.

Comandi ADT

Il primo argomento passato a ADT specifica uno dei comandi seguenti.

- `-package` - compila un'applicazione AIR o un'estensione nativa AIR (ANE, AIR Native Extension).
- `-prepare` - compila un'applicazione AIR come file intermedio (AIR) ma non la firma. I file AIRI non possono essere installati.

- `-sign` - firma un pacchetto AIRI prodotto con il comando `-prepare`. I comandi `-prepare` e `-sign` consentono di eseguire la compilazione e la firma in momenti diversi. Potete usare il comando `-sign` anche per firmare o ri-firmare un pacchetto ANE.
- `-migrate` - applica una firma di migrazione a un pacchetto AIR firmato, permettendovi di usare un certificato di firma codice nuovo o rinnovato.
- `-certificate` - crea un certificato digitale autofirmato di firma del codice.
- `-checkstore` - verifica che un certificato digitale in un archivio di chiavi sia accessibile.
- `-installApp` - installa un'applicazione AIR su un dispositivo o un emulatore di dispositivo.
- `-launchApp` - avvia un'applicazione AIR su un dispositivo o un emulatore di dispositivo.
- `-appVersion` - segnala la versione di un'applicazione AIR attualmente installata su un dispositivo o un emulatore di dispositivo.
- `-uninstallApp` - disinstalla un'applicazione AIR da un dispositivo o un emulatore di dispositivo.
- `-installRuntime` - installa il runtime AIR su un dispositivo o un emulatore di dispositivo.
- `-runtimeVersion` - segnala la versione del runtime AIR attualmente installato su un dispositivo o un emulatore di dispositivo.
- `-uninstallRuntime` - disinstalla il runtime AIR attualmente installato da un dispositivo o un emulatore di dispositivo.
- `-version` - segnala il numero di versione di ADT.
- `-devices` - segnala le informazioni dispositivo per i dispositivi mobili o emulatori connessi.
- `-help` - visualizza l'elenco dei comandi e delle opzioni.

Molti comandi ADT hanno in comune varie serie di flag di opzioni e di parametri, che sono descritte in dettaglio separatamente:

- [“Opzioni di firma codice ADT”](#) a pagina 185
- [“Opzioni per file e percorsi”](#) a pagina 188
- [“Opzioni per la connessione del debugger”](#) a pagina 189
- [“Opzioni per le estensioni native”](#) a pagina 189

Comando ADT package

Il comando `-package` deve essere eseguito dalla directory dell'applicazione principale. Questo comando prevede le seguenti sintassi:

Creare un pacchetto AIR dai file componenti dell'applicazione:

```
adt -package
    AIR_SIGNING_OPTIONS
    -target packageType
    -sampler
    -hideAneLibSymbols
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    FILE_OPTIONS
```

Creare un pacchetto nativo dai file componenti dell'applicazione:

```
adt -package
    AIR_SIGNING_OPTIONS
    -target packageType
    DEBUGGER_CONNECTION_OPTIONS
    -airDownloadURL URL
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    -platformsdk path
    FILE_OPTIONS
```

Creare un pacchetto nativo che includa un'estensione nativa dai file applicazione del componente:

```
adt -package
    AIR_SIGNING_OPTIONS
    -migrate MIGRATION_SIGNING_OPTIONS
    -target packageType
    DEBUGGER_CONNECTION_OPTIONS
    -airDownloadURL URL
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    -platformsdk path
    FILE_OPTIONS
```

Creare un pacchetto nativo da un file AIR o AIRI:

```
adt -package
    -target packageType
    NATIVE_SIGNING_OPTIONS
    output
    input_package
```

Creare un pacchetto di estensioni native dai file componenti l'estensione nativa:

```
adt -package
    AIR_SIGNING_OPTIONS
    -target ane
    output
    ANE_OPTIONS
```

Nota: non è necessario firmare un file ANE; quindi, i parametri `AIR_SIGNING_OPTIONS` in questo esempio sono opzionali.

AIR_SIGNING_OPTIONS Le opzioni di firma AIR identificano il certificato da utilizzare per firmare un file di installazione AIR. Le opzioni di firma sono descritte in maniera completa in [“Opzioni di firma codice ADT”](#) a pagina 185.

-migrate: questo flag specifica che l'applicazione è firmata con un certificato di migrazione oltre al certificato specificato nei parametri `AIR_SIGNING_OPTIONS`. Questo flag è valido solo se state creando un'applicazione desktop come programma di installazione nativo e l'applicazione utilizza un'estensione nativa. In altri casi viene rilevato un errore. Le opzioni di firma per il certificato di migrazione sono specificate come parametri **MIGRATION_SIGNING_OPTIONS**. Tali opzioni di firma sono descritte in maniera completa in [“Opzioni di firma codice ADT”](#) a pagina 185. Tramite il flag `-migrate` è possibile creare un aggiornamento per un'applicazione di un programma di installazione nativo desktop che utilizza un'estensione nativa e modificare il certificato di firma del codice per l'applicazione, come nel caso della scadenza del certificato originale. Per maggiori informazioni, consultate [“Firma di una versione aggiornata di un'applicazione AIR”](#) a pagina 206.

Il flag `-migrate` del comando `-package` è disponibile in AIR 3.6 e versioni successive.

-target Il tipo di pacchetto da creare. I tipi di pacchetto supportati sono:

- **air** - un pacchetto AIR. "air" è il valore predefinito e non occorre specificare il flag **-target** quando si creano file AIR o AIRL.
- **airn** - un pacchetto di applicazione nativo per dispositivi che usano il profilo tv esteso.
- **ane** - un pacchetto di estensioni native AIR
- Target del pacchetto Android:
 - **apk** - un pacchetto Android. Un pacchetto prodotto con questo target può essere installato solo su un dispositivo Android, non su un emulatore.
 - **apk-captive-runtime** - un pacchetto Android che contiene sia l'applicazione che una versione autonoma del runtime AIR. Un pacchetto prodotto con questo target può essere installato solo su un dispositivo Android, non su un emulatore.
 - **apk-debug** - un pacchetto Android con informazioni di debug supplementari. (I file SWF nell'applicazione devono a loro volta essere compilati con il supporto per il debug.)
 - **apk-emulator** - un pacchetto Android da utilizzare su un emulatore senza supporto per il debug. (Utilizzate il valore target **apk-debug** per consentire il debug sia sugli emulatori che sui dispositivi.)
 - **apk-profile** - un pacchetto Android che supporta le prestazioni dell'applicazione e i profili di memoria.
- Target del pacchetto iOS:
 - **ipa-ad-hoc** - un pacchetto iOS per la distribuzione ad hoc.
 - **ipa-app-store** - un pacchetto iOS per la distribuzione su Apple App Store.
 - **ipa-debug** - un pacchetto iOS con informazioni di debug supplementari. (I file SWF nell'applicazione devono a loro volta essere compilati con il supporto per il debug.)
 - **ipa-test** - un pacchetto iOS compilato con informazioni di debug o di ottimizzazione.
 - **ipa-debug-interpreter** - funzionalità equivalente a un pacchetto di debug, ma con compilazione più rapida. Tuttavia, il codice byte ActionScript viene interpretato e non tradotto in codice macchina. Di conseguenza, l'esecuzione del codice risulta più lenta in un pacchetto interprete.
 - **ipa-debug-interpreter-simulator** - Funzionalmente equivalente a **ipa-debug-interpreter**, ma compilato per iOS Simulator. Solo per Macintosh. Se usate questa opzione, dovete anche includere l'opzione **-platformsdk**, specificando il percorso di iOS Simulator SDK.
 - **ipa-test-interpreter** - funzionalità equivalente a un pacchetto di test, ma con compilazione più rapida. Tuttavia, il codice byte ActionScript viene interpretato e non tradotto in codice macchina. Di conseguenza, l'esecuzione del codice risulta più lenta in un pacchetto interprete.
 - **ipa-test-interpreter-simulator** - Funzionalmente equivalente a **ipa-test-interpreter**, ma compilato per iOS Simulator. Solo per Macintosh. Se usate questa opzione, dovete anche includere l'opzione **-platformsdk**, specificando il percorso di iOS Simulator SDK.
- **native** - un programma di installazione desktop nativo. Il tipo di file prodotto è il formato di installazione nativo del sistema operativo nel quale il comando viene eseguito:
 - EXE - Windows
 - DMG - Mac
 - DEB - Ubuntu Linux (AIR 2.6 o versioni precedenti)
 - RPM - Fedora o OpenSuse Linux (AIR 2.6 o versioni precedenti)

Per ulteriori informazioni, consultate “[Creazione del pacchetto di un file di installazione nativo desktop](#)” a pagina 59.

-sampler (solo per iOS, AIR 3.4 e versioni successive) Abilita il campionatore ActionScript basato sulla telemetria in applicazioni iOS. Tramite questo flag potete creare un profilo dell'applicazione con Adobe Scout. Anche se con [Scout](#) è possibile creare il profilo di qualsiasi contenuto su piattaforma Flash, abilitando una telemetria dettagliata potrete analizzare più in profondità la temporizzazione delle funzioni ActionScript, DisplayList, il rendering Stage3D e molto altro ancora. Tenete presente che l'uso di questo flag rallenta leggermente le prestazioni, pertanto non va usato per applicazioni di produzione.

-hideAneLibSymbols (solo per iOS, AIR 3.4 e versioni successive) Gli sviluppatori di applicazioni possono utilizzare più estensioni native da più origini e, se i file ANE condividono un nome simbolo comune, ADT genera un errore di "simbolo duplicato nel file dell'oggetto". In alcuni casi, questo errore si può addirittura manifestare con un blocco del sistema in fase di runtime. Potete utilizzare l'opzione `hideAneLibSymbols` per specificare se rendere visibili o meno i simboli della libreria ANE solo alle origini della libreria stessa (si) o a livello globale (no):

- **si** - Consente di nascondere i simboli ANE, risolvendo eventuali problemi di conflittualità dei simboli.
- **no** - (Predefinito) Non nasconde i simboli ANE. Questo è il comportamento precedente a AIR 3.4.

-embedBitcode (solo iOS, AIR 25 e successive) Gli sviluppatori di applicazioni possono utilizzare l'opzione `embedBitcode` per specificare se includere bitcode nelle loro applicazioni iOS specificando sì o no. Il valore predefinito di questa opzione se non specificato è no.

DEBUGGER_CONNECTION_OPTIONS Le opzioni di connessione del debugger specificano quando un pacchetto di debug deve tentare di connettersi a un debugger remoto in esecuzione su un altro computer oppure intercettare una connessione da un debugger remoto. Questo set di opzioni è supportato solo per i pacchetti di debug per dispositivi mobili (con target `apk-debug` e `ipa-debug`). Queste opzioni sono descritte in “[Opzioni per la connessione del debugger](#)” a pagina 189.

-airDownloadURL Specifica un URL alternativo per scaricare e installare il runtime AIR su dispositivi Android. Se non è specificato, un'applicazione AIR reindirizza l'utente al runtime AIR su Android Market se il runtime non è già installato.

Se l'applicazione è distribuita tramite un canale commerciale diverso da Android Market (che è amministrato da Google), potreste dover specificare l'URL per scaricare il runtime AIR da tale canale. Alcuni spazi commerciali alternativi non ammettono le applicazioni che devono essere scaricate da URL esterni. Questa opzione è supportata solo per i pacchetti Android.

NATIVE_SIGNING_OPTIONS Le opzioni di firma native identificano il certificato da utilizzare per firmare un file di pacchetto nativo. Queste opzioni di firma consentono di applicare una firma utilizzata dal sistema operativo nativo, non dal runtime AIR. Per il resto le opzioni sono identiche a `AIR_SIGNING_OPTIONS` e descritte in maniera completa in “[Opzioni di firma codice ADT](#)” a pagina 185.

Le firme native sono supportate in Windows e Android. In Windows è necessario specificare sia le opzioni di firma AIR che le opzioni di firma native. In Android è possibile specificare solo le opzioni di firma native.

In molti casi potete utilizzare lo stesso certificato di firma del codice per applicare sia una firma AIR che una firma nativa. Tuttavia, questa possibilità non è disponibile in tutti i casi. Ad esempio, la politica di Google per le applicazioni inviate a Android Market prevede che tutte le app debbano essere firmate con un certificato valido almeno fino all'anno 2033. Ciò significa che un certificato emesso da un'autorità di certificazione ben nota, assolutamente raccomandato quando si applica una firma AIR, non deve invece essere utilizzato per firmare un'app Android. (Nessuna autorità di certificazione emette certificati di firma codice con un periodo di validità così lungo.)

output Il nome del file di pacchetto da creare. L'estensione del file è opzionale. Se non la specificate, viene aggiunta un'estensione appropriata per il valore `-target` e il sistema operativo.

app_descriptor Il percorso del file descrittore dell'applicazione. Può essere specificato come percorso assoluto o come percorso relativo alla directory corrente. Il file descrittore dell'applicazione viene rinominato come *application.xml* nel file AIR.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione:

- Android - Il kit AIR 2.6+ SDK include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente AIR_ANDROID_SDK_HOME è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)
- iOS - AIR SDK viene fornito con una versione autonoma di iOS SDK. L'opzione -platformsdk consente di compilare applicazioni con un SDK esterno in modo da non essere vincolati all'uso del kit iOS SDK autonomo. Ad esempio, se avete creato un'estensione con l'ultimo SDK per iOS, potete specificare quello stesso SDK quando compilate l'applicazione. Inoltre, quando usate ADT con iOS Simulator, dovete anche includere l'opzione -platformsdk, specificando il percorso di iOS Simulator SDK.

Gli sviluppatori dell'applicazione **-arch** possono utilizzare questo argomento per creare APK per piattaforme x86, tramite i seguenti valori:

- APK pacchetti armv7 - ADT per piattaforma Android armv7.
- APK pacchetti x86 - ADT per piattaforma Android x86.

armv7 è il valore predefinito quando non è specificato alcun valore

FILE_OPTIONS Identifica i file dell'applicazione da includere nel pacchetto. Le opzioni per i file sono descritte in maniera completa in “Opzioni per file e percorsi” a pagina 188. Non specificate le opzioni file quando create un pacchetto nativo da un file AIR o AIRI.

input_airi Da specificare quando create un pacchetto nativo da un file AIRI. Le opzioni AIR_SIGNING_OPTIONS sono necessarie se il target è *air* (oppure non specificato).

input_air Da specificare quando create un pacchetto nativo da un file AIR. Non specificate le opzioni AIR_SIGNING_OPTIONS.

ANE_OPTIONS Identifica le opzioni e i file per la creazione di un pacchetto di estensioni native. Le opzioni del pacchetto di estensioni sono descritte dettagliatamente in “Opzioni per le estensioni native” a pagina 189.

Esempi di comando -package ADT

Creare pacchetti di specifici file dell'applicazione nella directory corrente per un'applicazione AIR basata su SWF:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf components.swc
```

Creare pacchetti di specifici file dell'applicazione nella directory corrente per un'applicazione AIR basata su HTML:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html AIRAliases.js image.gif
```

Creare pacchetti di tutti i file e sottodirectory nella directory di lavoro corrente:

```
adt -package -storetype pkcs12 -keystore ../cert.p12 myApp.air myApp.xml .
```

Nota: il file archivio di chiavi contiene la chiave privata impiegata per firmare l'applicazione. Non includete mai il certificato per la firma nel pacchetto AIR! Se sono stati utilizzati caratteri jolly nel comando ADT, collocate il file archivio di chiavi in una posizione differente, in modo che non venga incluso nel pacchetto. In questo esempio il file archivio di chiavi, *cert.p12*, risiede nella directory principale.

Creare pacchetti solo dei file principali e di una sottodirectory images:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf images
```

Creare pacchetti di un'applicazione basata su HTML e di tutti i file nelle sottodirectory HTML, scripts e images:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml index.html AIRAliases.js  
html scripts images
```

Creare un pacchetto del file application.xml e del file SWF principale situato in una directory di lavoro (release/bin):

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml -C  
release/bin myApp.swf
```

Creare un pacchetto di risorse da più di una posizione del file system di creazione. In questo esempio, prima della creazione del pacchetto le risorse dell'applicazione sono situate nelle seguenti cartelle:

```
/devRoot  
  /myApp  
    /release  
      /bin  
        myApp-app.xml  
        myApp.swf or myApp.html  
  /artwork  
    /myApp  
      /images  
        image-1.png  
        ...  
        image-n.png  
  /libraries  
    /release  
      /libs  
        lib-1.swf  
        lib-2.swf  
        lib-a.js  
        AIRAliases.js
```

L'esecuzione del seguente comando ADT dalla directory /devRoot/myApp:

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp-app.xml  
-C release/bin myApp.swf (or myApp.html)  
-C ../artwork/myApp images  
-C ../libraries/release libs
```

determina la seguente struttura di pacchetti:

```
/myAppRoot  
  /META-INF  
    /AIR  
      application.xml  
      hash  
  myApp.swf or myApp.html  
  mimetype  
  /images  
    image-1.png  
    ...  
    image-n.png  
  /libs  
    lib-1.swf  
    lib-2.swf  
    lib-a.js  
    AIRAliases.js
```

Eseguire ADT come programma Java per un'applicazione semplice basata su SWF (senza impostare il percorso di classe):

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air
myApp.xml myApp.swf
```

Eseguire ADT come programma Java per un'applicazione semplice basata su HTML (senza impostare il percorso di classe):

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air
myApp.xml myApp.html AIRAliases.js
```

Eseguire ADT come programma Java (con il percorso di classe Java impostato in modo da includere il pacchetto ADT.jar):

```
java -com.adobe.air.ADT -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml
myApp.swf
```

Eseguire ADT come attività Java in Apache Ant (anche se in genere è consigliabile usare il comando ADT direttamente negli script Ant). I percorsi visualizzati negli esempi si riferiscono a Windows:

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADT.JAR" value="\${SDK_HOME}/lib/adt.jar"/>

target name="package">
  <java jar="\${ADT.JAR}" fork="true" failonerror="true">
    <arg value="-package"/>
    <arg value="-storetype"/>
    <arg value="pkcs12"/>
    <arg value="-keystore"/>
    <arg value=".../ExampleCert.p12"/>
    <arg value="myApp.air"/>
    <arg value="myApp-app.xml"/>
    <arg value="myApp.swf"/>
    <arg value="icons/*.png"/>
  </java>
</target>
```

Nota: su alcuni computer, i caratteri a doppio byte nei percorsi del file system possono essere interpretati erroneamente. Se ciò si verifica, provate a impostare il JRE usato per eseguire ADT in modo da utilizzare il set di caratteri UTF-8. Questa impostazione viene effettuata in modo predefinito nello script utilizzato per avviare ADT su Mac e Linux. Nel file `adt.bat` di Windows, oppure quando eseguite ADT direttamente da Java, specificate l'opzione `-Dfile.encoding=UTF-8` nella riga di comando Java.

Comando ADT prepare

Il comando `-prepare` crea un pacchetto AIRI non firmato. Un pacchetto AIRI non può essere utilizzato in modo autonomo. Utilizzate il comando `-sign` per convertire un file AIRI in un pacchetto AIR firmato, oppure il comando `package` per convertire il file AIRI in un pacchetto nativo.

Il comando `-prepare` prevede la seguente sintassi:

```
adt -prepare output app_descriptor FILE_OPTIONS
```

output Il nome del file AIRI da creare.

app_descriptor Il percorso del file descrittore dell'applicazione. Può essere specificato come percorso assoluto o come percorso relativo alla directory corrente. Il file descrittore dell'applicazione viene rinominato come *application.xml* nel file AIR.

FILE_OPTIONS Identifica i file dell'applicazione da includere nel pacchetto. Le opzioni per i file sono descritte in maniera completa in [“Opzioni per file e percorsi”](#) a pagina 188.

Comando ADT sign

Il comando `-sign` firma i file AIRI e ANE.

Il comando `-sign` prevede la seguente sintassi:

```
adt -sign AIR_SIGNING_OPTIONS input output
```

AIR_SIGNING_OPTIONS Le opzioni di firma AIR identificano il certificato da utilizzare per firmare un file di pacchetto. Le opzioni di firma sono descritte in maniera completa in [“Opzioni di firma codice ADT”](#) a pagina 185.

input Il nome del file AIRI o ANE da firmare.

output Il nome del pacchetto firmato da creare.

Se un file ANE è già firmato, la vecchia firma viene rimossa. (Non è possibile firmare nuovamente i file AIR; per utilizzare una nuova firma per un aggiornamento dell'applicazione, utilizzare il comando `-migrate`.)

Comando ADT migrate

Il comando `-migrate` applica una firma di migrazione a un file AIR. Una firma di migrazione deve essere usata quando si rinnova o si sostituisce il certificato digitale e si devono aggiornare applicazioni firmate con il certificato vecchio.

Per maggiori informazioni sulla creazione di un pacchetto di applicazioni AIR con una firma di migrazione, consultate [“Firma di una versione aggiornata di un'applicazione AIR”](#) a pagina 206.

***Nota:** il certificato di migrazione deve essere applicato entro 365 giorni dalla scadenza del certificato. Una volta trascorso tale periodo di tolleranza, non è più possibile firmare gli aggiornamenti dell'applicazione con una firma di migrazione. Gli utenti possono prima effettuare l'aggiornamento a una versione dell'applicazione firmata con una firma di migrazione e quindi installare l'ultimo aggiornamento, oppure possono disinstallare l'applicazione originale e installare il nuovo pacchetto AIR.*

Per utilizzare una firma di migrazione, firmate prima la vostra applicazione AIR usando il certificato nuovo o rinnovato (con il comando `-package` o `-sign`), quindi applicate la firma di migrazione utilizzando il vecchio certificato e il comando `-migrate`.

Il comando `-migrate` utilizza la seguente sintassi:

```
adt -migrate AIR_SIGNING_OPTIONS input output
```

AIR_SIGNING_OPTIONS Le opzioni di firma AIR identificano il certificato originale utilizzato per firmare le versioni esistenti dell'applicazione AIR. Le opzioni di firma sono descritte in maniera completa in [“Opzioni di firma codice ADT”](#) a pagina 185.

input Il file AIR già firmato con il NUOVO certificato dell'applicazione.

output Il nome del pacchetto finale che porta le firme sia del vecchio che del nuovo certificato.

i nomi di file utilizzati per i file AIR di input e di output devono essere diversi.

***Nota:** il comando ADT migrate non può essere utilizzato con applicazioni desktop AIR che includono estensioni native, in quanto tali applicazioni sono inserite in un pacchetto come programmi di installazione nativi, non come file .air. Per modificare i certificati per un'applicazione desktop AIR che include un'estensione nativa, inserite in un pacchetto l'applicazione mediante il [“Comando ADT package”](#) a pagina 172 con il flag `-migrate`.*

Comando ADT checkstore

Il comando `-checkstore` permette di verificare la validità di un archivio di chiavi. Il comando prevede la sintassi seguente:

```
adt -checkstore SIGNING_OPTIONS
```

SIGNING_OPTIONS Le opzioni di firma che identificano l'archivio di chiavi da convalidare. Le opzioni di firma sono descritte in maniera completa in [“Opzioni di firma codice ADT”](#) a pagina 185.

Comando ADT certificate

Il comando `-certificate` permette di creare un certificato digitale autofirmato di firma del codice. Il comando prevede la sintassi seguente:

```
adt -certificate -cn name -ou orgUnit -o orgName -c country -validityPeriod years key-type  
output password
```

-cn La stringa assegnata al nome comune del nuovo certificato.

-ou Una stringa assegnata come unità organizzativa che rilascia il certificato. (Opzionale)

-o Una stringa assegnata come organizzazione che rilascia il certificato. (Opzionale)

-c Un codice Paese di due lettere, in base allo standard ISO-3166. Se viene fornito un codice non valido, il certificato non viene generato. (Opzionale)

-validityPeriod Il numero di anni per cui il certificato sarà valido. Se non è specificato, al certificato viene assegnata una validità di cinque anni. (Opzionale)

key_type Il tipo di chiave da utilizzare per il certificato è *2048-RSA*.

output Il percorso e il nome file del certificato da generare.

password La password per accedere al nuovo certificato. La password è obbligatoria per firmare i file AIR mediante questo certificato.

Comando ADT installApp

Il comando `-installApp` installa un'applicazione su un dispositivo o un emulatore.

Dovete disinstallare un'applicazione esistente prima di reinstallarla con questo comando.

Il comando prevede la sintassi seguente:

```
adt -installApp -platform platformName -platformsdk path-to-sdk -device deviceID -package  
fileName
```

-platform Il nome della piattaforma del dispositivo. Specificate *ios* o *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione (opzionale):

- **Android** - Il kit AIR 2.6+ SDK include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente `AIR_ANDROID_SDK_HOME` è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)

- iOS - AIR SDK viene fornito con una versione autonoma di iOS SDK. L'opzione `-platformsdk` consente di compilare applicazioni con un SDK esterno in modo da non essere vincolati all'uso del kit iOS SDK autonomo. Ad esempio, se avete creato un'estensione con l'ultimo SDK per iOS, potete specificare quello stesso SDK quando compilate l'applicazione. Inoltre, quando usate ADT con iOS Simulator, dovete anche includere l'opzione `-platformsdk`, specificando il percorso di iOS Simulator SDK.

-device Specificate `ios_simulator` e il numero di serie (Android) o l'handle (iOS) del dispositivo connesso. In iOS, questo parametro è obbligatorio; in Android, deve essere specificato solo se più dispositivi o emulatori Android sono collegati al computer e in esecuzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo oppure Dispositivo non valido specificato (iOS). Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

Nota: l'installazione di un file IPA direttamente su un dispositivo iOS è supportata in AIR 3.4 e versioni successive e richiede iTunes 10.5.0 o versioni successive.

Utilizzate il comando `adt -devices` (disponibile in AIR 3.4 e versioni successive) per determinare l'handle o il numero di serie dei dispositivi connessi. Tenete presente che in iOS occorre usare l'handle, non il codice UUID del dispositivo. Per ulteriori informazioni, vedete “Comando ADT devices” a pagina 185.

Inoltre, in Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

-package Il nome file del pacchetto da installare. In iOS deve essere un file IPA. In Android deve trattarsi di un pacchetto APK. Se il pacchetto specificato è già installato, ADT restituisce il codice di errore 14: Errore dispositivo.

Comando ADT appVersion

Il comando `-appVersion` segnala la versione di un'applicazione installata su un dispositivo o un emulatore. Il comando prevede la sintassi seguente:

```
adt -appVersion -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform Il nome della piattaforma del dispositivo. Specificate `ios` o `android`.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione:

- Android - Il kit AIR 2.6+ SDK include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente `AIR_ANDROID_SDK_HOME` è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)
- iOS - AIR SDK viene fornito con una versione autonoma di iOS SDK. L'opzione `-platformsdk` consente di compilare applicazioni con un SDK esterno in modo da non essere vincolati all'uso del kit iOS SDK autonomo. Ad esempio, se avete creato un'estensione con l'ultimo SDK per iOS, potete specificare quello stesso SDK quando compilate l'applicazione. Inoltre, quando usate ADT con iOS Simulator, dovete anche includere l'opzione `-platformsdk`, specificando il percorso di iOS Simulator SDK.

-device Specificate `ios_simulator` o il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori Android sono collegati al computer e in funzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

-appid L'ID applicazione AIR dell'applicazione installata. Se sul dispositivo non è installata un'applicazione con l'ID specificato, ADT restituisce il codice di uscita 14: Errore dispositivo.

Comando ADT launchApp

Il comando `-launchApp` esegue un'applicazione installata su un dispositivo o un emulatore. Il comando prevede la sintassi seguente:

```
adt -launchApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform Il nome della piattaforma del dispositivo. Specificate *ios* o *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione:

- Android - Il kit AIR 2.6+ SDK include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente `AIR_ANDROID_SDK_HOME` è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)
- iOS - AIR SDK viene fornito con una versione autonoma di iOS SDK. L'opzione `-platformsdk` consente di compilare applicazioni con un SDK esterno in modo da non essere vincolati all'uso del kit iOS SDK autonomo. Ad esempio, se avete creato un'estensione con l'ultimo SDK per iOS, potete specificare quello stesso SDK quando compilate l'applicazione. Inoltre, quando usate ADT con iOS Simulator, dovete anche includere l'opzione `-platformsdk`, specificando il percorso di iOS Simulator SDK.

-device Specificate *ios_simulator* o il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori Android sono collegati al computer e in funzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

-appid L'ID applicazione AIR dell'applicazione installata. Se sul dispositivo non è installata un'applicazione con l'ID specificato, ADT restituisce il codice di uscita 14: Errore dispositivo.

Comando ADT uninstallApp

Il comando `-uninstallApp` rimuove completamente un'applicazione installata su un dispositivo remoto o un emulatore. Il comando prevede la sintassi seguente:

```
adt -uninstallApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform Il nome della piattaforma del dispositivo. Specificate *ios* o *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione:

- Android - Il kit AIR 2.6+ SDK include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente AIR_ANDROID_SDK_HOME è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)
- iOS - AIR SDK viene fornito con una versione autonoma di iOS SDK. L'opzione -platformsdk consente di compilare applicazioni con un SDK esterno in modo da non essere vincolati all'uso del kit iOS SDK autonomo. Ad esempio, se avete creato un'estensione con l'ultimo SDK per iOS, potete specificare quello stesso SDK quando compilate l'applicazione. Inoltre, quando usate ADT con iOS Simulator, dovete anche includere l'opzione -platformsdk, specificando il percorso di iOS Simulator SDK.

-device Specificate *ios_simulator* o il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori Android sono collegati al computer e in funzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

-appid L'ID applicazione AIR dell'applicazione installata. Se sul dispositivo non è installata un'applicazione con l'ID specificato, ADT restituisce il codice di uscita 14: Errore dispositivo.

Comando ADT installRuntime

Il comando -installRuntime installa il runtime AIR su un dispositivo.

Dovete disinstallare la versione esistente del runtime AIR prima di reinstallarla con questo comando.

Il comando prevede la sintassi seguente:

```
adt -installRuntime -platform platformName -platformsdk path_to_sdk -device deviceID -package fileName
```

-platform Il nome della piattaforma del dispositivo. Attualmente questo comando è supportato solo sulla piattaforma Android. Usate il nome *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione. Attualmente, l'unico kit SDK di piattaforma supportato è Android. Il kit SDK AIR 2.6+ include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente AIR_ANDROID_SDK_HOME è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)

-device Il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori sono collegati al computer e in funzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

-package Il nome file del runtime da installare. In Android, deve trattarsi di un pacchetto APK. Se non viene specificato un pacchetto, viene scelto il runtime appropriato per il dispositivo o l'emulatore tra quelli disponibili in AIR SDK. Se il runtime è già installato, ADT restituisce il codice di errore 14: Errore dispositivo.

Comando ADT runtimeVersion

Il comando `-runtimeVersion` segnala la versione del runtime AIR installata su un dispositivo o un emulatore. Il comando prevede la sintassi seguente:

```
adt -runtimeVersion -platform platformName -platformsdk path_to_sdk -device deviceID
```

-platform Il nome della piattaforma del dispositivo. Attualmente questo comando è supportato solo sulla piattaforma Android. Usate il nome *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione. Attualmente, l'unico kit SDK di piattaforma supportato è Android. Il kit SDK AIR 2.6+ include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente `AIR_ANDROID_SDK_HOME` è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)

-device Il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori sono collegati al computer e in funzione. Se il runtime non è installato o il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

Comando ADT uninstallRuntime

Il comando `-uninstallRuntime` rimuove completamente il runtime AIR da un dispositivo o un emulatore. Il comando prevede la sintassi seguente:

```
adt -uninstallRuntime -platform platformName -platformsdk path_to_sdk -device deviceID
```

-platform Il nome della piattaforma del dispositivo. Attualmente questo comando è supportato solo sulla piattaforma Android. Usate il nome *android*.

-platformsdk Il percorso del kit SDK della piattaforma per il dispositivo di destinazione. Attualmente, l'unico kit SDK di piattaforma supportato è Android. Il kit SDK AIR 2.6+ include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Inoltre, il percorso del kit SDK della piattaforma non deve essere specificato nella riga di comando se la variabile d'ambiente `AIR_ANDROID_SDK_HOME` è già impostata. (Se entrambi i dati sono impostati, viene utilizzato il percorso indicato sulla riga di comando.)

-device Il numero di serie del dispositivo. Il dispositivo deve essere specificato solo se più dispositivi o emulatori sono collegati al computer e in funzione. Se il dispositivo specificato non è collegato, ADT restituisce il codice di uscita 14: Errore dispositivo. Se è collegato più di un dispositivo o di un emulatore e non viene specificato un dispositivo, ADT restituisce il codice di uscita 2: Errore di utilizzo.

In Android, usate il tool Android ADB per elencare i numeri di serie dei dispositivi collegati e degli emulatori in esecuzione:

```
adb devices
```

Comando ADT devices

Il comando ADT `-devices` visualizza gli ID dispositivo dei dispositivi mobili e degli emulatori attualmente connessi:

```
adt -devices -platform ios|android
```

-platform Il nome della piattaforma da controllare. Specificare `android` o `ios`.

Nota: in iOS, questo comando richiede iTunes 10.5.0 o una versione successiva.

Comando ADT help

Il comando ADT `-help` visualizza un riepilogo delle opzioni della riga di comando:

```
adt -help
```

L'output del comando `help` usa le seguenti convenzioni per i caratteri:

- `<>` - le voci tra parentesi angolari corrispondono a informazioni che devono essere specificate dall'utente.
- `()` - le voci tra parentesi tonde indicano opzioni considerate come gruppo nell'output del comando `help`.
- MAIUSCOLO - le voci in maiuscolo indicano una serie di opzioni che sono descritte separatamente.
- `|` - oppure. Ad esempio, `(A | B)` significa A oppure B.
- `? - 0 o 1`. Un punto di domanda dopo un elemento significa che l'elemento è opzionale e che ne può essere comunque utilizzata una sola istanza.
- `* - 0 o più`. Un asterisco dopo un elemento significa che l'elemento è opzionale e che ne può essere utilizzato un numero qualsiasi di istanze.
- `+ - 1 o più`. Un segno più dopo un elemento significa che l'elemento è obbligatorio e che ne possono essere utilizzate più istanze.
- nessun simbolo - Se un elemento è privo di suffisso, si tratta di un elemento obbligatorio per il quale deve essere specificata una sola istanza.

Serie di opzioni ADT

Vari comandi ADT hanno in comune le stesse serie di opzioni.

Opzioni di firma codice ADT

ADT utilizza JCA (Java Cryptography Architecture) per accedere alle chiavi private e ai certificati per la firma di applicazioni AIR. Le opzioni di firma identificano l'archivio di chiavi nonché il certificato e la chiave privata all'interno di quell'archivio.

L'archivio di chiavi deve includere sia la chiave privata che la catena di certificati associata. Se il certificato per la firma è concatenato a un certificato attendibile presente su un computer, il contenuto del campo del nome comune del certificato viene visualizzato come nome editore nella finestra di dialogo di installazione di AIR.

ADT richiede che il certificato sia conforme allo standard x509v3 ([RFC3280](#)) e includa l'estensione Extended Key Usage con valori appropriati per la firma del codice. I limiti definiti all'interno del certificato vengono rispettati e possono precludere l'utilizzo di alcuni certificati per la firma di applicazioni AIR.

***Nota:** se necessario, ADT utilizza le impostazioni proxy dell'ambiente di runtime Java per connettersi alle risorse Internet e verificare gli elenchi di revoca dei certificati e ottenere indicatori di data e ora. Se si incontrano problemi nella connessione a queste risorse Internet quando si utilizza ADT e la rete richiede impostazioni proxy specifiche, può essere necessario configurare le impostazioni proxy JRE.*

Sintassi delle opzioni di firma AIR

Le opzioni di firma prevedono la seguente sintassi (su un'unica riga di comando):

```
-alias aliasName
-storetype type
-keystore path
-storepass password1
-keypass password2
-providerName className
-tsa url
```

-alias L'alias di una chiave nell'archivio di chiavi. Se un archivio di chiavi contiene un solo certificato, non è necessario specificare un alias. Quando non è specificato alcun alias, ADT utilizza la prima chiave dell'archivio.

Non tutte le applicazioni di gestione dell'archivio di chiavi consentono l'assegnazione di un alias ai certificati. Quando usate l'archivio di chiavi di sistema di Windows, ad esempio, utilizzate come alias il nome distinto del certificato. L'utilità Java Keytool consente di elencare i certificati disponibili in modo che sia possibile determinare l'alias. Ad esempio, l'esecuzione del comando:

```
keytool -list -storetype Windows-MY
```

genera un output analogo al seguente per un certificato:

```
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
```

Per fare riferimento a questo certificato sulla riga di comando di ADT, impostate l'alias su:

```
CN=TestingCert,OU=QE,O=Adobe,C=US
```

In Mac OS X, l'alias di un certificato nel portachiavi corrisponde al nome visualizzato nell'applicazione Accesso portachiavi.

-storetype Il tipo di archivio di chiavi, determinato dall'implementazione dell'archivio di chiavi. L'implementazione dell'archivio di chiavi predefinita inclusa con la maggior parte delle installazioni di Java supporta i tipi `JKS` e `PKCS12`. Java 5.0 include il supporto per il tipo `PKCS11` per l'accesso agli archivi di chiavi sui token hardware, e il tipo `Keychain`, per l'accesso al portachiavi di Mac OS X. Java 6.0 include il supporto per il tipo `MSCAPI` (in Windows). Se sono stati installati e configurati altri provider JCA, potrebbero essere disponibili ulteriori tipi di archivi di chiavi. Se non è stato specificato alcun tipo di archivio di chiavi, viene utilizzato il tipo predefinito per il provider JCA predefinito.

Tipo di archivio	Formato dell'archivio di chiavi	Versione minima di Java
JKS	File dell'archivio di chiavi Java (keystore)	1.2
PKCS12	File PKCS12 (.p12 o .pfx)	1.4
PKCS11	Token hardware	1.5
KeychainStore	Portachiavi Mac OS X	1.5
Windows-MY o Windows-ROOT	MSCAPI	1.6

-keystore Il percorso al file dell'archivio di chiavi per tipi di archivi basati su file.

-storepass La password necessaria per accedere all'archivio di chiavi. Se non viene specificata, ADT richiede la password.

-keypass La password necessaria per accedere alla chiave privata utilizzata per firmare l'applicazione AIR. Se non viene specificata, ADT richiede la password.

***Nota:** se includete una password nel comando ADT, i caratteri della password vengono salvati nella cronologia della riga di comando. Di conseguenza, l'uso delle opzioni `-keypass` e `-storepass` è sconsigliato quando la sicurezza del certificato è un aspetto importante. Inoltre, se omettete le opzioni relative alla password, i caratteri che digitate nei prompt delle password non vengono visualizzati (per gli stessi motivi legati alla sicurezza). Digitate semplicemente la password e premete il tasto Invio.*

-providerName Il provider JCA per il tipo di archivio di chiavi specificato. Se il valore non viene specificato, ADT utilizza il provider predefinito per quel tipo di archivio di chiavi.

-tsa Specifica l'URL di un server di indicatori di data e ora conforme a [RFC3161](#) per applicare l'indicazione di data e ora alla firma digitale. Se non viene specificato alcun URL, viene utilizzato un server di indicatori data e ora fornito da Geotrust. Quando alla firma di un'applicazione AIR vengono aggiunte indicazioni di data e ora, è possibile installare l'applicazione anche dopo che il certificato è scaduto, in quanto l'indicatore di data e ora verifica che il certificato era valido al momento della firma.

Se ADT non è in grado di connettersi al server di indicazione di data e ora, la firma viene annullata e non è possibile produrre alcun pacchetto. Per disattivare l'aggiunta di indicatori di data e ora, specificate `-tsa none`. Un'applicazione AIR inserita in un pacchetto senza un indicatore di data e ora, tuttavia, non è più installabile dopo la scadenza del certificato per la firma.

***Nota:** molte delle opzioni di firma sono equivalenti alle opzioni corrispondenti dell'utilità Java Keytool. È possibile utilizzare l'utilità Keytool per esaminare e gestire archivi di chiavi su Windows. Allo stesso scopo, in Mac OS X si può utilizzare l'utilità di sicurezza Apple®.*

-provisioning-profile Il file di provisioning Apple iOS. (Richiesto solo per la compilazione di applicazioni iOS.)

Esempi di opzioni di firma

Firma mediante un file p12:

```
-storetype pkcs12 -keystore cert.p12
```

Firma mediante l'archivio di chiavi Java predefinito:

```
-alias AIRcert -storetype jks
```

Firma mediante uno specifico archivio di chiavi Java:

```
-alias AIRcert -storetype jks -keystore certStore.keystore
```

Firma mediante il portachiavi Mac OS X:

```
-alias AIRcert -storetype KeychainStore -providerName Apple
```

Firma mediante l'archivio di chiavi di sistema di Windows:

```
-alias cn=AIRCert -storetype Windows-MY
```

Firma mediante un token hardware (vedete le istruzioni del produttore del token sulla configurazione di Java per l'utilizzo del token e per il valore corretto di `providerName`):

```
-alias AIRCert -storetype pkcs11 -providerName tokenProviderName
```

Firma senza incorporare un indicatore di data e ora:


```
-storetype pkcs12 -keystore cert.p12 -tsa none
```

Opzioni per file e percorsi

Le opzioni relative ai file e ai percorsi specificano tutti i file che sono inclusi nel pacchetto. Le opzioni di file e percorsi prevedono la seguente sintassi:

```
files_and_dirs -C dir files_and_dirs -e file_or_dir dir -extdir dir
```

files_and_dirs I file e le directory da includere nel pacchetto del file AIR. È possibile specificare un numero qualsiasi di file e directory, delimitati da spazi vuoti. Se si elenca una directory, tutti i file e le sottodirectory inclusi, ad eccezione dei file nascosti, vengono aggiunti al pacchetto. Inoltre, se il file descrittore dell'applicazione viene specificato direttamente o attraverso l'espansione di directory o caratteri jolly, esso viene ignorato e non viene aggiunto al pacchetto una seconda volta. File e directory specificati devono trovarsi nella directory corrente o in una delle relative sottodirectory. Utilizzate l'opzione `-C` per cambiare la directory corrente.

Importante: non è possibile utilizzare caratteri jolly negli argomenti `file_or_dir` che seguono l'opzione `-C`. Le shell comandi, infatti, espandono i caratteri jolly prima della trasmissione degli argomenti ad ADT, che pertanto cerca i file nella posizione sbagliata. È tuttavia possibile utilizzare il carattere punto, vale a dire ".", per indicare la directory corrente. Ad esempio, `-C assets .` copia tutto nella directory `assets`, incluse tutte le sottodirectory, nel livello principale del pacchetto dell'applicazione.

`-C dir files_and_dirs` Cambia la directory di lavoro nel valore di `dir` prima dell'elaborazione di directory e file successivi aggiunti al pacchetto dell'applicazione (specificati in `files_and_dirs`). File o directory vengono aggiunti alla radice del pacchetto dell'applicazione. L'opzione `-C` può essere utilizzata ogniqualvolta si desidera includere file da più punti nel file system. Se per `dir` viene specificato un percorso relativo, esso viene risolto dalla directory di lavoro originale.

Quando ADT elabora i file e le directory incluse nel pacchetto, i percorsi relativi tra la directory corrente e i file di destinazione vengono archiviati. Questi percorsi vengono espansi nella directory dell'applicazione quando il pacchetto viene installato. Pertanto, se si specifica `-C release/bin lib/feature.swf`, il file `release/bin/lib/feature.swf` viene collocato nella sottodirectory `lib` della cartella principale dell'applicazione.

`-e file_or_dir dir` Inserisce il file o la directory nella directory di pacchetto specificata. Questa opzione non può essere usata quando si compila un file ANE.

Nota: l'elemento `<content>` del file descrittore dell'applicazione deve specificare la posizione finale del file principale dell'applicazione nella struttura di directory del pacchetto dell'applicazione.

`-extdir dir` Il valore di `dir` è il nome di una directory in cui ricercare le estensioni native (file ANE). Specificare un percorso assoluto o un percorso relativo alla directory corrente. Potete specificare l'opzione `-extdir` più di una volta.

La directory specificata contiene file ANE per ciascuna estensione nativa utilizzata dall'applicazione. Ogni file ANE di questa directory ha l'estensione file `.ane`. Tuttavia, il nome del file prima dell'estensione `.ane` non deve corrispondere al valore dell'elemento `extensionID` del file descrittore dell'applicazione.

Ad esempio, se utilizzate `-extdir/extensions`, la directory `extensions` potrà presentarsi come segue:

```
extensions/  
  extension1.ane  
  extension2.ane
```

Nota: l'uso dell'opzione `-extdir` varia a seconda che si utilizzi il tool ADT o ADL. In ADL, l'opzione specifica una directory contenente sottodirectory, ciascuna contenente un file ANE spaccettato. In ADT, le opzioni specificano una directory contenente file ANE.

Opzioni per la connessione del debugger

Quando il target del pacchetto è `apk-debug`, `ipa-debug` o `ipa-debug-interpreter`, le opzioni di connessione possono essere utilizzate per specificare se l'app tenta di connettersi a un debugger remoto (solitamente utilizzato per il debug via wi-fi) o di restare in ascolto per intercettare una connessione in entrata da un debugger remoto (solitamente utilizzato per il debug via USB). Utilizzate l'opzione `-connect` per connettere un debugger o l'opzione `-listen` per accettare una connessione da un debugger su un collegamento USB. Queste opzioni si escludono reciprocamente, quindi non possono essere usate insieme.

Il comando `-connect` prevede la sintassi seguente:

```
-connect hostString
```

-connect Se è presente, l'applicazione tenterà di connettersi a un debugger remoto.

hostString Una stringa che identifica il computer sul quale è in esecuzione il tool FDB (Flash debugging tool). Se non è specificata, l'app tenterà di connettersi a un debugger in esecuzione sul computer sul quale il pacchetto è stato creato. La stringa `hostString` può essere un nome di dominio computer completo (ad es. *nomecomputer.sottogruppo.esempio.com*) oppure un indirizzo IP (ad es. *192.168.4.122*). Se il computer specificato (o predefinito) non viene trovato, il runtime visualizza una finestra di dialogo che richiede di specificare un nome `host` valido.

Il comando `-listen` prevede la sintassi seguente:

```
-listen port
```

-listen Se è presente, il runtime attende di ricevere una connessione da un debugger remoto.

port (Opzionale) la porta su cui rilevare la connessione. Per impostazione predefinita, il runtime usa la porta 7936. Per ulteriori informazioni sull'uso dell'opzione `-listen`, vedete [“Debug remoto con FDB via USB”](#) a pagina 110.

Opzioni per i profili applicazione Android

Quando il target del pacchetto è `apk-profile`, le opzioni per i profili possono essere utilizzate per specificare quale file SWF precaricato deve essere utilizzato per la creazione dei profili delle prestazioni e della memoria. Le opzioni dei profili prevedono la seguente sintassi:

```
-preloadSWFPath directory
```

-preloadSWFPath Se presente, l'app tenterà di trovare il file SWF di precaricamento nella `directory` specificata. Se non viene specificata, ADT include il file SWF di precaricamento di AIR SDK.

directory La `directory` che contiene il file SWF di precaricamento dei profili.

Opzioni per le estensioni native

Le opzioni per le estensioni native specificano le opzioni e i file per la compilazione di un file ANE per un'estensione nativa. Usate queste opzioni con un comando del pacchetto ADT in cui l'opzione `-target` è `ane`.

```
extension-descriptor -swc swcPath  
  -platform platformName  
  -platformoptions path/platform.xml  
  FILE_OPTIONS
```

extension-descriptor Il file descrittore dell'estensione nativa.

-swc Il file SWC che contiene il codice ActionScript e le risorse dell'estensione nativa.

-platform Il nome della piattaforma supportata da questo file ANE. Potete includere varie opzioni `-platform`, ognuna con le proprie `FILE_OPTIONS`.

-platformoptions Il percorso di un file di opzioni della piattaforma (platform.xml). Utilizzate questo file per specificare opzioni linker non predefinite e librerie statiche di terze parti utilizzate dall'estensione. Per ulteriori informazioni ed esempi, vedete Librerie native iOS.

FILE_OPTIONS Identifica i file nativi della piattaforma da includere nel pacchetto, ad esempio librerie statiche da includere nel pacchetto dell'estensione nativa. Le opzioni per i file sono descritte in maniera completa in [“Opzioni per file e percorsi”](#) a pagina 188. (L'opzione -e non può essere usata quando si compila un file ANE.)

Altri argomenti presenti nell’Aiuto

[Compilazione di un'estensione nativa](#)

Messaggi di errore di ADT

Nelle seguenti tabelle sono elencati i possibili errori segnalati dal programma ADT e le probabili cause:

Errori di convalida del descrittore dell'applicazione

Codice di errore	Descrizione	Note
100	Application descriptor cannot be parsed (Impossibile analizzare il descrittore dell'applicazione)	Verificate che il file descrittore dell'applicazione non contenga errori di sintassi XML, ad esempio tag non chiusi.
101	Namespace is missing (Spazio dei nomi mancante)	Aggiungete lo spazio dei nomi mancante.
102	Invalid namespace (Spazio dei nomi non valido)	Verificate l'ortografia dello spazio dei nomi.
103	Unexpected element or attribute (Elemento o attributo imprevisto)	Rimuovete gli elementi e gli attributi errati. Nel file descrittore non sono consentiti valori personalizzati. Verificate l'ortografia dei nomi di elementi e di attributi. Assicuratevi che gli elementi siano inseriti nell'elemento principale corretto e che gli attributi siano usati con gli elementi corretti.
104	Missing element or attribute (Elemento o attributo mancante)	Aggiungete l'elemento o l'attributo richiesto.
105	Element or attribute contains an invalid value (L'elemento o l'attributo contiene un valore non valido)	Correggete il valore errato.
106	Illegal window attribute combination (Combinazione di attributi della finestra non valida)	Alcune impostazioni delle finestra, ad esempio <code>transparency = true</code> e <code>systemChrome = standard</code> non possono essere usate insieme. Modificate una delle impostazioni incompatibili.
107	Window minimum size is larger than the window maximum size (Le dimensioni minime della finestra sono maggiori delle dimensioni massime)	Modificate l'impostazione delle dimensioni minime o massime.

Codice di errore	Descrizione	Note
108	Attribute already used in prior element (Attributo già utilizzato nell'elemento precedente)	
109	Duplicate element (Elemento duplicato).	Rimuovete l'elemento duplicato.
110	At least one element of the specified type is required. (È necessario almeno un elemento del tipo specificato.)	Aggiungete l'elemento mancante.
111	None of the profiles listed in the application descriptor support native extensions. (Nessuno dei profili elencati nel descrittore dell'applicazione supporta le estensioni native.)	Aggiungete all'elenco supportedProfiles un profilo che supporta le estensioni native.
112	The AIR target doesn't support native extensions. (Il target AIR non supporta le estensioni native.)	Scegliete un target che supporta le estensioni native.
113	<nativeLibrary> and <initializer> must be provided together. (<nativeLibrary> e <initializer> devono essere specificati insieme.)	È necessario specificare una funzione di inizializzazione per ogni libreria nativa dell'estensione nativa.
114	Found <finalizer> without <nativeLibrary>. (Trovato <finalizer> senza <nativeLibrary>.)	Non specificate un finalizer a meno che la piattaforma non utilizzi una libreria nativa.
115	The default platform must not contain a native implementation. (La piattaforma predefinita non deve contenere un'implementazione nativa.)	Non specificate una libreria nativa nell'elemento piattaforma predefinita.
116	Browser invocation is not supported for this target. (La chiamata al browser non è supportata da questo target.)	L'elemento <allowBrowserInvocation> non può essere true per il target specificato.
117	This target requires at least namespace n to package native extensions. (Questo target richiede almeno uno spazio dei nomi n per compilare estensioni native.)	Cambiate lo spazio dei nomi di AIR nel descrittore dell'applicazione per supportare il valore.

Per informazioni su spazi dei nomi, elementi, attributi e relativi valori validi, vedete “[File descrittori delle applicazioni AIR](#)” a pagina 212.

Errori relativi all'icona dell'applicazione

Codice di errore	Descrizione	Note
200	Icon file cannot be opened (Impossibile aprire il file di icona)	Verificate che il file sia presente nel percorso specificato. Usate un'altra applicazione per accertarvi che il file possa essere aperto.
201	Icon is the wrong size (Dimensioni dell'icona errate)	Le dimensioni in pixel dell'icona devono corrispondere al tag XML. Dato, ad esempio l'elemento del descrittore dell'applicazione: <image32x32>icon.png</image32x32> Le dimensioni dell'immagine in icon.png devono essere esattamente 32x32 pixel.
202	Icon file contains an unsupported image format (Il file di icona contiene un formato di immagine non supportato)	È supportato solo il formato PNG. Convertite le immagini in altri formati prima di creare il pacchetto dell'applicazione.

Errori relativi al file dell'applicazione

Codice di errore	Descrizione	Note
300	Missing file, or file cannot be opened (File mancante o impossibile aprirlo)	Non è possibile individuare o aprire un file specificato nella riga di comando.
301	Application descriptor file missing or cannot be opened (File del descrittore dell'applicazione mancante o impossibile aprirlo)	Non è possibile individuare il file del descrittore dell'applicazione nel percorso specificato oppure non è possibile aprirlo.
302	Root content file missing from package (File del contenuto principale mancante dal pacchetto)	Il file SWF o HTML a cui fa riferimento l'elemento <content> del descrittore dell'applicazione deve essere aggiunto al pacchetto, includendolo nei file elencati nella riga di comando di ADT.
303	Icon file missing from package (File di icona mancante dal pacchetto)	Il file di icona specificati nel descrittore dell'applicazione devono essere aggiunto al pacchetto, includendoli nei file elencati nella riga di comando di ADT. I file di icona non vengono aggiunti automaticamente.
304	Initial window content is invalid (Il contenuto della finestra iniziale non è valido)	Il file a cui fa riferimento l'elemento <content> del descrittore dell'applicazione non viene riconosciuto come file HTML o SWF valido.

Codice di errore	Descrizione	Note
305	Initial window content SWF version exceeds namespace version (La versione SWF del contenuto della finestra iniziale è maggiore della versione dello spazio dei nomi)	La versione SWF del file a cui fa riferimento l'elemento <content> del descrittore dell'applicazione non è supportata dalla versione di AIR specificata nello spazio dei nomi del descrittore. Questo errore viene generato, ad esempio, se tentate di includere in un pacchetto un file SWF10 (Flash Player 10) come contenuto iniziale di un'applicazione AIR 1.1.
306	Profile not supported (Profilo non supportato)	Il profilo specificato nel file descrittore dell'applicazione non è supportato. Vedete "supportedProfiles" a pagina 248.
307	Namespace must be at least <i>nnn</i> . (Lo spazio dei nomi deve essere almeno <i>nnn</i> .)	Utilizzate lo spazio dei nomi appropriato per le funzioni dell'applicazione (ad esempio lo spazio dei nomi 2.0).

Codici di uscita per altri errori

Codice di uscita	Descrizione	Note
2	Usage error (Errore di utilizzo)	Verificate che gli argomenti della riga di comando siano corretti.
5	Unknown error (Errore sconosciuto)	Questo errore indica una situazione che non può essere spiegata con le comuni condizioni di errore. Le possibili cause principali di questo errore comprendono: incompatibilità tra ADT e Java Runtime Environment, installazioni di ADT o JRE danneggiate ed errori di programmazione all'interno di ADT.
6	Could not write to output directory (Impossibile scrivere nella directory di output)	Accertatevi che la directory di output specificata (o implicita) sia accessibile e che sull'unità che la contiene sia disponibile spazio su disco sufficiente.
7	Could not access certificate (Impossibile accedere al certificato)	Accertatevi che il percorso dell'archivio di chiavi sia specificato correttamente. Controllate che il certificato nell'archivio di chiavi sia accessibile. Per facilitare la risoluzione dei problemi di accesso ai certificati, potete usare l'utilità Java 1.6 Keytool.
8	Invalid certificate (Certificato non valido)	Il file di certificato non è strutturato correttamente, è scaduto o è stato modificato o revocato.
9	Could not sign AIR file (Impossibile firmare il file AIR)	Verificate le opzioni di firma passate ad ADT.
10	Could not create time stamp (Impossibile creare l'indicatore di data e ora)	ADT non è stato in grado di stabilire una connessione al server degli indicatori di data e ora. Se vi connettete a Internet tramite un server proxy, potreste dover configurare le impostazioni del proxy JRE.
11	Certificate creation error (Errore durante la creazione del certificato)	Verificate gli argomenti della riga di comando usati per creare le firme.

Codice di uscita	Descrizione	Note
12	Invalid input (Input non valido)	Verificate i percorsi dei file e gli altri argomenti passati ad ADT nella riga di comando.
13	Missing device SDK (SDK dispositivo assente)	Verificate la configurazione SDK del dispositivo. ADT non è in grado di individuare il kit SDK dispositivo necessario per eseguire il comando specificato.
14	Device error (Errore dispositivo)	ADT non è in grado di eseguire il comando a causa di una restrizione o di un problema del dispositivo. Ad esempio, questo codice di uscita viene generato quando si tenta di disinstallare un'applicazione che in realtà non è installata.
15	No devices (Nessun dispositivo)	Verificate che sia collegato e acceso un dispositivo o che sia in esecuzione un emulatore.
16	Missing GPL components (Componenti GPL mancanti)	La versione corrente di AIR SDK non include tutti i componenti necessari per eseguire l'operazione richiesta.
17	Device packaging tool failed. (Errore dello strumento di creazione pacchetto.)	Non è stato possibile creare il pacchetto perché alcuni componenti previsti del sistema operativo sono mancanti.

Errori Android

Codice di uscita	Descrizione	Note
400	Current Android sdk version doesn't support attribute. (La versione dell'SDK Android corrente non supporta l'attributo.)	Controllate che il nome dell'attributo sia scritto correttamente e che l'attributo sia valido per l'elemento nel quale appare. Potreste dover impostare il flag -platformsdk nel comando ADT se l'attributo è stato introdotto dopo Android 2.2.
401	Current Android sdk version doesn't support attribute value. (La versione dell'SDK Android corrente non supporta il valore di attributo.)	Controllate che il valore dell'attributo sia scritto correttamente e che il valore sia valido per l'attributo. Potreste dover impostare il flag -platformsdk nel comando ADT se il valore dell'attributo è stato introdotto dopo Android 2.2.
402	Current Android sdk version doesn't support XML tag. (La versione dell'SDK Android corrente non supporta il tag XML.)	Controllate che il nome del tag XML sia scritto correttamente e che sia un elemento di documento manifest Android valido. Potreste dover impostare il flag -platformsdk nel comando ADT se l'elemento è stato introdotto dopo Android 2.2.
403	Android tag is not allowed to be overridden (Non è possibile eseguire l'override del tag Android)	L'applicazione sta tentando di eseguire l'override di un elemento manifest Android che è riservato per AIR. Vedete "Impostazioni Android" a pagina 79.

Codice di uscita	Descrizione	Note
404	Android attribute is not allowed to be overridden (Non è possibile eseguire l'override dell'attributo Android)	L'applicazione sta tentando di eseguire l'override di un attributo manifest Android che è riservato per AIR. Vedete "Impostazioni Android" a pagina 79.
405	Android tag %1 must be the first element in manifestAdditions tag (Il tag Android %1 deve corrispondere al primo elemento nel tag manifestAdditions)	Spostate il tag specificato nella posizione richiesta.
406	The attribute %1 of the android tag %2 has invalid value %3. (L'attributo %1 del tag Android %2 presenta un valore %3 non valido.)	Specificate un valore valido per l'attributo.

Variabili d'ambiente ADT

ADT legge i valori delle seguenti variabili d'ambiente (se sono impostati):

AIR_ANDROID_SDK_HOME specifica il percorso della directory principale di Android SDK (la directory che contiene la cartella tools). Il kit SDK AIR 2.6+ include i tool di Android SDK necessari per implementare i comandi ADT richiesti. Impostate questo valore solo per utilizzare una versione differente di Android SDK. Se questa variabile è impostata, l'opzione `-platformsdk` non deve essere specificata quando si eseguono i comandi ADT nei quali è richiesta. Se entrambi i dati sono impostati (la variabile e l'opzione della riga di comando), viene utilizzato il percorso indicato sulla riga di comando.

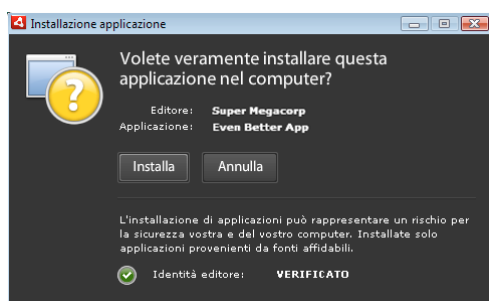
AIR_EXTENSION_PATH specifica un elenco di directory in cui cercare le estensioni native richieste da un'applicazione. La ricerca viene eseguita nelle directory contenute in questo elenco nell'ordine specificato, dopo eventuali directory di estensioni native specificate nella riga di comando ADT. Anche il comando ADL usa questa variabile ambientale.

Nota: su alcuni computer, i caratteri a doppio byte nei percorsi del file system memorizzati in queste variabili d'ambiente possono essere interpretati erroneamente. Se ciò si verifica, provate a impostare il JRE usato per eseguire ADT in modo da utilizzare il set di caratteri UTF-8. Questa impostazione viene effettuata in modo predefinito nello script utilizzato per avviare ADT su Mac e Linux. Nel file `adt.bat` di Windows, oppure quando eseguite ADT direttamente da Java, specificate l'opzione `-Dfile.encoding=UTF-8` nella riga di comando Java.

Capitolo 13: Firma di applicazioni AIR

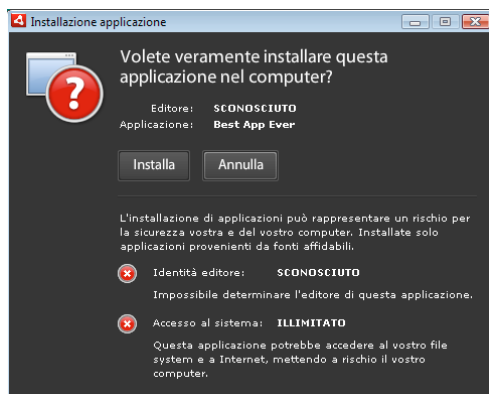
Firma digitale di un file AIR

L'applicazione di una firma digitale ai file di installazione AIR mediante un certificato rilasciato da un'autorità di certificazione (CA) riconosciuta offre una garanzia significativa che l'applicazione in fase di installazione non sia stata alterata in modo casuale o intenzionalmente dannoso e identifica lo sviluppatore come firmatario (editore). Il nome dell'editore viene visualizzato durante l'installazione quando l'applicazione AIR è stata firmata con un certificato attendibile o che risulta *concatenato* a un certificato attendibile sul computer di installazione:



Finestra di conferma dell'installazione per un'applicazione firmata con un certificato attendibile

Se firmate un'applicazione utilizzando un certificato autofirmato (oppure un certificato non concatenato a un certificato attendibile), l'utente deve accettare un rischio di sicurezza maggiore installando l'applicazione. Nelle finestre di dialogo di installazione viene segnalato questo rischio aggiuntivo:



Finestra di conferma dell'installazione per un'applicazione firmata con un certificato autofirmato

Importante: un'entità dannosa che abbia in qualche modo ottenuto il file di archivio chiavi per la firma dell'utente o che ne abbia scoperto la chiave privata può creare un file AIR con la vostra identità.

Certificati per la firma del codice

Le garanzie di sicurezza, i limiti e gli obblighi legali relativi all'utilizzo dei certificati per la firma del codice sono definiti nel CPS (Certificate Practice Statements) e negli accordi di sottoscrizione pubblicati dall'autorità di certificazione emittente. Per ulteriori informazioni sugli accordi per le autorità di certificazione che rilasciano certificati per la firma di codice AIR, vedete:

[ChosenSecurity](http://www.chosensecurity.com/products/tc_publisher_id_adobe_air.htm) (http://www.chosensecurity.com/products/tc_publisher_id_adobe_air.htm)

[ChosenSecurity CPS](http://www.chosensecurity.com/resource_center/repository.htm) (http://www.chosensecurity.com/resource_center/repository.htm)

[GlobalSign](http://www.globalsign.com/code-signing/index.html) (www.globalsign.com/code-signing/index.html)

[GlobalSign CPS](http://www.globalsign.com/repository/index.htm) (<http://www.globalsign.com/repository/index.htm>)

[Thawte CPS](http://www.thawte.com/cps/index.html) (CPS Thawte) (<http://www.thawte.com/cps/index.html>)

[VeriSign CPS](http://www.verisign.com/repository/CPS/) (<http://www.verisign.com/repository/CPS/>)

[Verisign Subscriber's Agreement](https://www.verisign.com/repository/subscriber/SUBAGR.html) (Contratto di abbonamento a Verisign)
(<https://www.verisign.com/repository/subscriber/SUBAGR.html>)

Informazioni sulla firma del codice AIR

Quando un file AIR viene firmato, nel file di installazione viene inclusa una firma digitale. Tale firma comprende un digest del pacchetto, usato per verificare che il file AIR non sia stato alterato dal momento della firma, e informazioni sulla firma del certificato, che consentono di verificare l'identità dell'editore.

Per stabilire se un certificato può essere ritenuto attendibile, AIR utilizza l'infrastruttura PKI (public key infrastructure, infrastruttura a chiave pubblica) supportata nell'archivio certificati del sistema operativo per stabilire l'attendibilità di un certificato. Il computer su cui l'applicazione AIR è installata deve ritenere direttamente attendibile il certificato utilizzato per la firma dell'applicazione AIR oppure deve considerare attendibile una catena di certificati che collegano il certificato stesso a un'autorità CA che verifichi le informazioni dell'editore.

Se un file AIR è firmato mediante un certificato non concatenato a uno dei certificati principali affidabili (e normalmente in questa categoria sono inclusi tutti i certificati autofirmati), le informazioni sull'editore non possono essere verificate. Anche se AIR può determinare che il pacchetto AIR non è stato alterato dal momento della firma, non c'è modo di sapere chi abbia effettivamente creato e firmato il file.

***Nota:** un utente può scegliere di ritenere affidabile un certificato autofirmato e quindi in qualsiasi applicazione AIR firmata con quel certificato viene visualizzato il valore del campo del nome comune come nome dell'editore. In AIR non viene fornito alcun sistema per consentire a un utente di designare un certificato come attendibile. Il certificato (che non include la chiave privata) deve essere fornito all'utente separatamente, e quest'ultimo deve utilizzare uno dei meccanismi forniti dal sistema operativo oppure uno strumento appropriato per importare il certificato nella posizione adatta all'interno dell'archivio certificati del sistema.*

Informazioni sugli ID editore AIR

***Importante:** a partire da AIR 1.5.3 l'ID editore è stato dichiarato obsoleto e non viene più calcolato in base al certificato di firma del codice. Per le nuove applicazioni l'ID editore non è necessario e non deve essere specificato. Quando aggiornate un'applicazione esistente, dovete specificare l'ID editore originale nel file descrittore dell'applicazione.*

Prima di AIR 1.5.3, il programma di installazione dell'applicazione AIR generava un ID editore durante l'installazione di un file AIR. Si trattava di un codice di identificazione univoco per il certificato utilizzato per firmare il file AIR. Se si riutilizzava lo stesso certificato per più applicazioni AIR, esse avevano tutte lo stesso ID editore. La firma di un aggiornamento dell'applicazione con un certificato differente e talvolta anche con un'istanza rinnovata del certificato originale modificava l'ID editore.

A partire da AIR 1.5.3, l'ID editore non viene più assegnato da AIR. Un'applicazione pubblicata con AIR 1.5.3 può specificare una stringa di ID editore nel descrittore dell'applicazione. Dovete specificare un ID editore solo quando pubblicate aggiornamenti per applicazioni originariamente pubblicate per versioni di AIR anteriori alla 1.5.3. Se non specificate l'ID originale nel descrittore dell'applicazione, il nuovo pacchetto AIR non viene considerato come aggiornamento dell'applicazione esistente.

Per determinare l'ID editore originale, individuate il file `publisherid` nella sottodirectory `META-INF/AIR` in cui è installata l'applicazione originale. La stringa contenuta in questo file è l'ID editore. Per specificare manualmente l'ID editore, dovete specificare il runtime AIR 1.5.3 (o successivo) nella dichiarazione namespace del file descrittore dell'applicazione.

L'ID editore, se presente, viene utilizzato per i seguenti scopi:

- Come parte della chiave di crittografia per l'archivio locale crittografato
- Come parte del percorso della directory di archiviazione dell'applicazione
- Come parte della stringa di connessione per le connessioni locali
- Come parte della stringa di identità utilizzata per chiamare un'applicazione con l'API AIR interna al browser
- Come parte dell'identificatore OSID (utilizzato per la creazione di programmi di installazione/disinstallazione personalizzati)

Quando un ID editore viene modificato, cambia anche il comportamento delle funzioni AIR associate all'ID. Ad esempio, non è più possibile accedere ai dati presenti nell'archivio locale crittografato, ed eventuali istanze Flash o AIR che creano una connessione locale all'applicazione devono usare il nuovo ID nella stringa di connessione. L'ID editore per un'applicazione installata non può essere modificato in AIR 1.5.3 o versioni successive. Se utilizzate un ID editore diverso quando pubblicate un pacchetto AIR, il programma di installazione tratta il nuovo pacchetto come se fosse un'applicazione differente anziché un aggiornamento.

Informazioni sui formati dei certificati

Gli strumenti di firma di AIR accettano qualsiasi archivio di chiavi accessibile mediante Java Cryptography Architecture (JCA). Sono quindi inclusi gli archivi di chiavi basati su file quali i file in formato PKCS12 (che di solito hanno un'estensione `pfx` o `p12`), i file Java keystore, gli archivi di chiavi hardware PKCS11 e gli archivi di chiavi di sistema. Il formato degli archivi di chiavi a cui può accedere ADT variano in base alla versione e configurazione del runtime Java utilizzato per l'esecuzione di ADT. L'accesso ad alcuni tipi di archivi di chiavi, come ad esempio i token hardware PKCS1, potrebbe richiedere l'installazione e la configurazione di plugin JCA e di driver software aggiuntivi.

Per firmare i file AIR, potete utilizzare la maggior parte dei certificati esistenti per la firma di codice oppure ottenerne uno nuovo rilasciato appositamente per la firma di applicazioni AIR. Potete ad esempio utilizzare uno dei tipi di certificati seguenti di VeriSign, Thawte, GlobalSign o ChosenSecurity:

- [ChosenSecurity](#)
 - TC Publisher ID per Adobe AIR
- [GlobalSign](#)
 - Certificato per la firma del codice di ObjectSign

- **Thawte:**
 - Certificato per sviluppatori AIR
 - Certificato per sviluppatori Apple
 - Certificato per sviluppatori JavaSoft
 - Certificato Microsoft Authenticode
- **VeriSign:**
 - ID digitale di Adobe AIR
 - ID digitale Microsoft Authenticode
 - ID digitale Sun Java Signing

***Nota:** il certificato deve essere creato per la firma di codice. Non potete utilizzare un certificato SSL o di altro tipo per firmare file AIR.*

Indicatori di data e ora

Quando si appone la firma a un file AIR, lo strumento di creazione dei pacchetti interroga il server di un'autorità di indicazione di data e ora per ottenere una data e ora della firma verificabile a livello indipendente. L'indicatore di data e ora ottenuto viene incorporato nel file AIR. Fino a quando il certificato è valido per l'ora della firma, il file AIR può essere installato, anche se tale certificato è scaduto. D'altro canto, se non viene ottenuto alcun indicatore di data e ora, il file AIR non può più essere installato dopo la scadenza o la revoca del certificato.

Per impostazione predefinita, gli strumenti di creazione pacchetti di AIR ottengono un indicatore di data e ora. Tuttavia, per consentire la creazione di pacchetti delle applicazioni anche quando il servizio di indicazione di data e ora non è disponibile, è possibile disattivare la richiesta del servizio. Adobe raccomanda la presenza di indicatori di data e ora su tutti i file AIR distribuiti a livello pubblico.

L'autorità predefinita per ottenere gli indicatori di data e ora per gli strumenti di creazione pacchetti AIR è Geotrust.

Ottenere un certificato

Per ottenere un certificato, normalmente si visita il sito Web dell'autorità di certificazione per completare il processo di approvvigionamento della società. Gli strumenti utilizzati per produrre l'archivio di chiavi richiesto dagli strumenti AIR varia in base al certificato acquistato, alla modalità di memorizzazione del certificato sul computer dell'utente e, in alcuni casi, al browser impiegato per ottenere il certificato. Per ottenere ed esportare, ad esempio, un certificato di Adobe Developer da Thawte, è necessario utilizzare Mozilla Firefox. Il certificato può quindi essere esportato come file .p12 o .pfx direttamente dall'interfaccia di Firefox.

***Nota:** le versioni Java dalla 1.5 in poi non accettano i caratteri high-ASCII nelle password utilizzate per proteggere i file di certificato PKCS12. Java viene utilizzato dagli strumenti di sviluppo di AIR per creare i pacchetti AIR firmati. Quando esportate il certificato come file .p12 o .pfx, utilizzate solo caratteri ASCII standard nella password.*

Potete generare un certificato autofirmato mediante ADT (Air Development Tool), lo strumento utilizzato per creare pacchetti di file di installazione AIR. È possibile utilizzare anche alcuni strumenti di terzi.

Per istruzioni su come generare un certificato autofirmato e sulla firma di un file AIR, vedete “[ADT \(AIR Developer Tool\)](#)” a pagina 171. Potete inoltre esportare e firmare file AIR mediante Flash Builder, Dreamweaver e l'aggiornamento di AIR per Flash.

Nell'esempio seguente viene illustrato come ottenere un certificato per sviluppatori AIR dall'autorità di certificazione Thawte e pararlo all'utilizzo in ADT.

Esempio: ottenere un certificato per sviluppatori AIR da Thawte

Nota: nell'esempio viene illustrato solo uno dei vari sistemi per ottenere e preparare all'uso un certificato per la firma del codice. Ogni autorità di certificazione dispone di politiche e procedure distinte.

Per acquistare un certificato per sviluppatori AIR sul sito Web di Thawte, è necessario utilizzare Mozilla Firefox. La chiave privata per il certificato è memorizzata nell'archivio di chiavi del browser. Accertarsi che l'archivio di chiavi di Firefox sia protetto con una password master e che il computer stesso sia fisicamente sicuro. Una volta terminato il processo di approvvigionamento, è possibile esportare e rimuovere il certificato e la chiave privata dall'archivio di chiavi del browser.

Come parte del processo di registrazione del certificato, viene generata una coppia di chiavi pubblica/privata. La chiave privata viene memorizzata automaticamente nell'archivio di chiavi di Firefox. Per richiedere e quindi recuperare il certificato dal sito Web di Thawte, è necessario utilizzare lo stesso computer.

- 1 Visitate il sito Web di Thawte e andate alla [pagina del prodotto per i certificati per la firma del codice](#).
- 2 Dall'elenco dei certificati per firma del codice, selezionate il certificato per sviluppatori AIR (Adobe AIR Developer Certificate).
- 3 Completate il processo di registrazione in tre fasi. Sarà necessario fornire informazioni di contatto e relative all'azienda. A questo punto, Thawte esegue il proprio processo di identificazione e potrebbe richiedere ulteriori informazioni. Al termine della verifica, poi, Thawte vi invierà un messaggio di posta elettronica con istruzioni su come recuperare il certificato.

Nota: ulteriori informazioni sul tipo di documentazione richiesta sono disponibili qui: https://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/enroll_codesign_eng.pdf.

- 4 Recuperate il certificato rilasciato dal sito Thawte. Esso viene salvato automaticamente nell'archivio di chiavi di Firefox.
- 5 Esportate un file dell'archivio di chiavi contenente la chiave privata e il certificato dall'archivio di chiavi di Firefox effettuando la seguente procedura:

Nota: la chiave privata e il certificato in Firefox vengono esportati in un formato p12 (pfx) che anche ADT, Flex, Flash e Dreamweaver sono in grado di utilizzare.

- a Aprite la finestra di dialogo *Gestione certificati* di Firefox:
- b In Windows: aprite Strumenti -> Opzioni -> Avanzate-> Crittografia -> Mostra certificati
- c In Mac OS: aprite Firefox -> Preferenze -> Avanzate -> Crittografia -> Mostra certificati
- d In Linux: aprite Modifica -> Preferenze -> Avanzate -> Crittografia -> Mostra certificati
- e Selezionate il certificato per la firma del codice di AIR (Adobe AIR Code Signing Certificate) dall'elenco dei certificati, quindi fate clic sul pulsante **Backup**.
- f Immettete un nome file e il percorso in cui esportare il file dell'archivio di certificati, quindi fate clic su **Salva**.
- g Se utilizzate la password master di Firefox, per l'esportazione del file vi verrà richiesto di immettere la password per il dispositivo di sicurezza del software. Questa password è utilizzata solo da Firefox.
- h Nella *finestra di dialogo di scelta di una password di backup per il certificato* create una password per la protezione del file dell'archivio di chiavi.

Importante: questa password protegge il file dell'archivio di chiavi ed è necessaria quando il file deve essere utilizzato per la firma di applicazioni AIR. Cercate di utilizzare una password sicura.

- i Fate clic su OK. Verrà visualizzato un messaggio di riuscita della creazione della password di backup. Il file dell'archivio di chiavi contenente la chiave privata e il certificato viene salvato con un'estensione p12 (in formato PKCS12).

- 6 Potete utilizzare il file dell'archivio di chiavi esportato con ADT, Flash Builder, Flash Professional o Dreamweaver. La password creata per il file è necessaria ogniqualvolta viene apposta la firma a un'applicazione AIR.

Importante: la chiave privata e il certificato vengono memorizzati nell'archivio di chiavi di Firefox. Ciò consente di esportare copie aggiuntive del file di certificato, ma implica anche la creazione di un ulteriore punto di accesso da proteggere per garantire la sicurezza di certificato e chiave privata.

Modifica dei certificati

In alcune circostanze dovete modificare il certificato utilizzato per firmare gli aggiornamenti dell'applicazione AIR. Ciò può avvenire, ad esempio, nei seguenti casi:

- Rinnovo del certificato di firma originale.
- Aggiornamento da un certificato autofirmato a un certificato rilasciato da un'autorità di certificazione
- Passaggio da un certificato autofirmato sul punto di scadere a un altro
- Passaggio da un certificato commerciale a un altro, ad esempio in caso di modifica della ragione sociale

Per consentire a AIR di riconoscere un file AIR come aggiornamento, dovete firmare sia i file originali che i file AIR di aggiornamento con lo stesso certificato oppure applicare una firma di migrazione certificato all'aggiornamento. Per firma di migrazione si intende una seconda firma apposta al pacchetto AIR di aggiornamento utilizzando il certificato originale. La firma di migrazione utilizza il certificato originale per stabilire che il firmatario è l'editore originale dell'applicazione.

Dopo che un file AIR con una firma di migrazione viene installato, il nuovo certificato diventa il certificato principale. Gli aggiornamenti successivi non richiedono una firma di migrazione. Tuttavia, è necessario applicare le firme di migrazione il più a lungo possibile per includere gli utenti che ignorano gli aggiornamenti.

Importante: dovete sostituire il certificato e applicare una firma di migrazione all'aggiornamento con il certificato originale prima della sua scadenza. In caso contrario, gli utenti devono disinstallare la versione corrente dell'applicazione prima di installare una nuova versione. Con AIR 1.5.3 o versioni successive, potete applicare una firma di migrazione utilizzando un certificato scaduto entro un periodo di tolleranza di 365 giorni dalla scadenza. Non è tuttavia possibile utilizzare il certificato scaduto per applicare la firma principale dell'applicazione.

Per cambiare i certificati:

- 1 Create un aggiornamento dell'applicazione
- 2 Create un pacchetto e firmate il file di aggiornamento di AIR con il **nuovo** certificato.
- 3 Firmate nuovamente il file AIR con il certificato **originale**, mediante il comando `-migrate` di ADT.

Per altri aspetti, un file con firma di migrazione è un normale file di AIR. Se l'applicazione è installata su un sistema senza la versione originale, la nuova versione viene installata nella maniera consueta.

Nota: prima di AIR 1.5.3, per firmare un'applicazione AIR con un certificato non era sempre necessario utilizzare una firma di migrazione. A partire da AIR 1.5.3, è sempre obbligatorio usare una firma di migrazione per i certificati rinnovati.

Per applicare una firma di migrazione utilizzate il “[Comando ADT migrate](#)” a pagina 179, come descritto in “[Firma di una versione aggiornata di un'applicazione AIR](#)” a pagina 206.

Nota: il comando ADT migrate non può essere utilizzato con applicazioni desktop AIR che includono estensioni native, in quanto tali applicazioni sono inserite in un pacchetto come programmi di installazione nativi, non come file .air. Per modificare i certificati per un'applicazione desktop AIR che include un'estensione nativa, inserite in un pacchetto l'applicazione mediante il “[Comando ADT package](#)” a pagina 172 con il flag `-migrate`.

Modifiche dell'identità dell'applicazione

Prima di AIR 1.5.3, l'identità di un'applicazione AIR cambiava quando veniva installato un aggiornamento firmato con una firma di migrazione. La modifica dell'identità di un'applicazione ha varie ripercussioni, tra cui:

- La nuova versione dell'applicazione non è in grado di accedere ai dati nell'archivio locale crittografato esistente.
- Il percorso della directory di memorizzazione dell'applicazione cambia. I dati nel vecchio percorso non vengono copiati nella nuova directory (la nuova applicazione, tuttavia, è in grado di individuare la directory originale in base al vecchio ID editore).
- L'applicazione non è più in grado di aprire le connessioni locali mediante il vecchio ID editore.
- La stringa di identità utilizzata per accedere a un'applicazione da una pagina Web cambia.
- L'identificatore OSID dell'applicazione cambia. (L'OSID viene utilizzato per scrivere programmi di installazione/disinstallazione personalizzati.)

Quando si pubblica un aggiornamento con AIR 1.5.3 o versioni successive, l'identità dell'applicazione non può cambiare. Nel descrittore dell'applicazione del file AIR di aggiornamento è necessario specificare gli ID applicazione e editore originali. In caso contrario, il nuovo pacchetto non viene riconosciuto come aggiornamento.

Nota: quando pubblicate una nuova applicazione AIR con AIR 1.5.3 o successivo, non dovete specificare un ID editore.

Terminologia

In questa sezione viene fornito un glossario della terminologia chiave, che è fondamentale conoscere per prendere decisioni in merito alla firma delle applicazioni per la pubblica distribuzione.

Termine	Descrizione
Autorità di certificazione (CA, Certification Authority)	Entità di una rete di infrastrutture di chiavi pubbliche che funge da terza parte attendibile e che certifica l'identità del proprietario di una chiave pubblica. Di norma, una CA rilascia certificati digitali firmati mediante la propria chiave privata per attestare l'avvenuta verifica dell'identità del possessore del certificato.
CPS (Certificate Practice Statement)	Documento che stabilisce le pratiche e le politiche dell'autorità di certificazione adottate in merito al rilascio e alla verifica dei certificati. Il CPS fa parte del contratto tra CA e le parti coinvolte. Definisce inoltre le politiche per la verifica dell'identità e il livello di garanzia offerto dai certificati forniti.
CRL (Certificate Revocation List)	Elenco di certificati rilasciati e successivamente revocati, che non possono più essere ritenuti affidabili. Nel momento in cui un'applicazione AIR viene firmata, AIR controlla l'elenco CRL. Se non è presente un indicatore di data e ora, il controllo viene ripetuto al momento dell'installazione dell'applicazione.
Catena di certificati	Per catena di certificati si intende una sequenza di certificati in cui ciascun certificato presente è stato firmato dal certificato successivo.
Certificato digitale	Documento digitale che contiene informazioni sull'identità del proprietario, la sua chiave pubblica e l'identità del certificato stesso. Un certificato rilasciato da un'autorità di certificazione è firmato a sua volta da un certificato appartenente alla CA che lo ha rilasciato.
Firma digitale	Messaggio o digest crittografato che è possibile decrittare solo mediante la chiave pubblica che costituisce metà della coppia di chiavi pubblica-privata. In un PKI, una firma digitale contiene uno o più certificati digitali di cui è possibile tenere traccia e che è possibile attribuire all'autorità di certificazione. Una firma digitale consente di convalidare la dichiarazione che un messaggio o un file non è stato alterato dal momento della firma (entro i limiti di garanzia forniti dall'algoritmo di crittografia utilizzato), nonché, presupponendo che l'autorità di certificazione sia considerata attendibile, l'identità del firmatario.
Archivio di chiavi	Database contenente certificati digitali nonché, in alcuni casi, le chiavi private correlate.

Termine	Descrizione
JCA (Java Cryptography Architecture)	Architettura estensibile per la gestione e l'accesso degli archivi di chiavi. Per ulteriori informazioni, vedete il documento Java Cryptography Architecture Reference Guide (Guida di riferimento per la Java Cryptography Architecture).
PKCS #11	Standard per l'interfaccia di token di crittografia creato da RSA Laboratories. Un archivio di chiavi basato su token hardware.
PKCS #12	Standard per la sintassi di scambio di informazioni personali creato da RSA Laboratories. Un archivio di chiavi basato su file che di solito contiene una chiave privata e il certificato digitale associato.
Chiave privata	La metà privata di una coppia di chiavi pubblica-privata a crittografia asimmetrica. La chiave privata deve essere tenuta segreta e non deve mai essere trasmessa su una rete. I messaggi con firma digitale vengono crittografati mediante chiave privata dal firmatario.
Chiave pubblica	La metà pubblica di una coppia di chiavi pubblica-privata a crittografia asimmetrica. La chiave pubblica è disponibile a tutti e viene usata per decrittare i messaggi crittografati mediante la chiave privata.
PKI (Public Key Infrastructure)	Sistema di attendibilità nel quale le autorità di certificazione attestano l'identità dei proprietari delle chiavi pubbliche. I client sulla rete si affidano ai certificati digitali rilasciati da una CA attendibile per verificare l'identità del firmatario di un messaggio digitale o file.
Indicatore di data e ora	Dato di riferimento a firma digitale contenente l'ora e il giorno in cui si è verificato un evento. ADT può includere un indicatore di data e ora proveniente da server temporale conforme alle specifiche RFC 3161 in un pacchetto AIR. Quando è presente, AIR utilizza l'indicatore di data e ora per stabilire la validità di un certificato al momento della firma, consentendo in tal modo l'installazione di un'applicazione AIR anche dopo che il certificato di firma è scaduto.
Autorità di rilascio degli indicatori di data e ora	Autorità che rilascia indicatori di data e ora. Per essere riconosciuto da AIR, l'indicatore di data e ora deve essere conforme alle specifiche RFC 3161 , mentre la firma dell'indicatore di data e ora deve concatenarsi a un certificato principale attendibile presente sul computer di installazione.

Certificati iOS

I certificati di firma del codice emessi da Apple vengono utilizzati per firmare le applicazioni iOS, comprese quelle sviluppate con Adobe AIR. L'applicazione di una firma mediante un certificato di sviluppo Apple è necessaria per installare un'applicazione su dispositivi di prova. L'applicazione di una firma mediante un certificato di distribuzione è invece richiesta per la distribuzione dell'applicazione finita.

Per firmare un'applicazione, ADT richiede l'accesso sia al certificato di firma del codice che alla relativa chiave privata. Il file del certificato non include la chiave privata. Dovete creare un archivio di chiavi sotto forma di file Personal Information Exchange (.p12 o .pfx) che contenga sia il certificato che la chiave privata. Vedete [“Conversione di un certificato per sviluppatori in un file di archivio chiavi P12”](#) a pagina 204.

Generazione di una richiesta di firma del certificato

Per ottenere un certificato per sviluppatori, dovete generare una richiesta di firma del certificato e inviarla al sito Apple iOS Provisioning Portal.

Il processo di richiesta della firma certificato genera una coppia di chiavi pubblica-privata. La chiave privata rimane sul vostro computer. Dovete inviare ad Apple, che svolge ruolo di autorità di certificazione, la richiesta di firma contenente la chiave pubblica e i vostri dati identificativi. Apple firma il vostro certificato con il proprio certificato World Wide Developer Relations.

Generare una richiesta di firma del certificato in Mac OS

In Mac OS, potete utilizzare l'applicazione Accesso Portachiavi per generare una richiesta di firma del certificato. L'applicazione Accesso Portachiavi si trova nella sottodirectory Utility della directory Applicazioni. Le istruzioni per la generazione della richiesta di firma certificato sono disponibili nel sito Apple iOS Provisioning Portal.

Generare una richiesta di firma del certificato in Windows

Per gli sviluppatori Windows, potrebbe essere più semplice ottenere il certificato per sviluppatori iPhone su un computer Mac. Tuttavia, è possibile ottenere un certificato su un computer Windows. Create, innanzitutto, un file CSR (Certificate Signing Request) utilizzando OpenSSL:

- 1 Installate OpenSSL sul computer Windows. (Visitate <http://www.openssl.org/related/binaries.html>.)

Potrebbe anche essere necessario installare i file ridistribuibili di Visual C++ 2008, elencati nella pagina di download di Open SSL. (*Non* è necessario che Visual C++ sia installato sul computer.)

- 2 Aprite una sessione di comandi Windows e passate alla directory bin OpenSSL (ad esempio, c:\OpenSSL\bin\).
- 3 Create la chiave privata immettendo l'istruzione seguente nella riga di comando:

```
openssl genrsa -out mykey.key 2048
```

Salvate questo file della chiave privata per utilizzarlo in seguito.

Quando utilizzate OpenSSL, non ignorate i messaggi di errore. Se OpenSSL genera un messaggio di errore, gli eventuali file creati non sono utilizzabili. In caso di errori, controllate la sintassi ed eseguite nuovamente il comando.

- 4 Create il file CSR immettendo l'istruzione seguente nella riga di comando:

```
openssl req -new -key mykey.key -out CertificateSigningRequest.certSigningRequest -subj  
"/emailAddress=yourAddress@example.com, CN=John Doe, C=US"
```

Sostituite l'indirizzo e-mail e i valori CN (nome certificato) e C (paese) con quelli personali.

- 5 Caricate il file CSR in Apple sul [sito per sviluppatori di iPhone](#). (Vedete "Richiesta di un certificato per sviluppatori iPhone e creazione di un file di provisioning".)

Conversione di un certificato per sviluppatori in un file di archivio chiavi P12

Per creare un archivio di chiavi P12, dovete combinare il certificato per sviluppatori Apple e la relativa chiave privata in un unico file. La procedura per la creazione del file di archivio chiavi dipende dal metodo che avete utilizzato per generare la richiesta di firma certificato originale e da dove è archiviata la chiave privata.

Convertire il certificato per sviluppatori iPhone in un file P12 in Mac OS

Dopo aver scaricato il certificato iPhone Apple da Apple, esportatelo nel formato di archivio chiavi P12. Per eseguire questa operazione in Mac® OS:

- 1 Aprite l'applicazione Accesso Portachiavi (nella cartella Applicazioni/Utility).
- 2 Se non avete già aggiunto il certificato a Portachiavi, selezionate File > Importa. Quindi passate al file di certificato (.cer) ottenuto da Apple.
- 3 Selezionate la categoria Chiavi nell'Accesso Portachiavi.
- 4 Selezionate la chiave privata associata al certificato di sviluppo iPhone.

La chiave privata è identificata dallo sviluppatore iPhone: <Nome> <Cognome> certificato pubblico con cui è associato.

- 5 Fate clic sul certificato di sviluppo iPhone tenendo premuto il tasto Comando e selezionate *Esporta "Sviluppatore iPhone: Nome..."*.
- 6 Salvate l'archivio di chiavi nel formato di file Personal Information Exchange (.p12).
- 7 Vi verrà richiesto di creare una password da specificare quando utilizzate l'archivio di chiavi per firmare applicazioni o trasferire la chiave e il certificato in questo archivio di chiavi o in un altro.

Convertire un certificato per sviluppatori di Apple in un file P12 in Windows

Per sviluppare applicazioni AIR for iOS, dovete usare un file di certificato P12. Questo file viene generato in base al certificato per sviluppatori iPhone Apple ricevuto da Apple.

- 1 Convertite il file di certificato per sviluppatori ricevuto da Apple in un file di certificato PEM. Eseguite l'istruzione della riga di comando seguente dalla directory bin di OpenSSL:

```
openssl x509 -in developer_identity.cer -inform DER -out developer_identity.pem -outform PEM
```

- 2 Se state utilizzando la chiave privata del portachiavi su un computer Mac, convertitela in una chiave PEM:

```
openssl pkcs12 -nocerts -in mykey.p12 -out mykey.pem
```

- 3 Potete ora generare un file P12 valido, basato sulla chiave e la versione PEM del certificato per sviluppatori iPhone:

```
openssl pkcs12 -export -inkey mykey.key -in developer_identity.pem -out iphone_dev.p12
```

Se state utilizzando una chiave del portachiavi Mac OS, utilizzate la versione PEM generata nel passaggio precedente. In caso contrario, utilizzate la chiave OpenSSL generata in precedenza (in Windows).

Creazione di un file AIR intermedio non firmato mediante ADT

Il comando `-prepare` consente di creare un file AIR intermedio non firmato, che è necessario firmare mediante il comando `-sign ADT` per produrre un file di installazione AIR valido.

Il comando `-prepare` utilizza gli stessi flag e parametri del comando `-package`, ad eccezione delle opzioni di firma. L'unica differenza è che il file di output non è firmato. Il file intermedio è generato mediante l'estensione del nome file: `airi`.

Per firmare un file AIR intermedio, utilizzate il comando `-sign` di ADT (Vedete ["Comando ADT prepare"](#) a pagina 178.)

Esempio di comando ADT `-prepare`

```
adt -prepare unsignedMyApp.airi myApp.xml myApp.swf components.swc
```

Firma di un file AIR intermedio mediante ADT

Per firmare un file AIR intermedio mediante ADT, utilizzate il comando `-sign`. Il comando di firma funziona solo con file AIR intermedi (estensione `airi`). Non è possibile firmare un file AIR una seconda volta.

Per creare un file AIR intermedio, utilizzate il comando `adt -prepare`. (Vedete ["Comando ADT prepare"](#) a pagina 178.)

Firmare un file AIRI

- ❖ Utilizzate il comando ADT `-sign` con la seguente sintassi:

```
adt -sign SIGNING_OPTIONS airi_file air_file
```

SIGNING_OPTIONS: le opzioni di firma identificano la chiave privata e il certificato da utilizzare per firmare il file AIR. Queste opzioni sono descritte in “Opzioni di firma codice ADT” a pagina 185.

airi_file: il percorso al file AIR intermedio non firmato da firmare.

air_file: il nome del file AIR da creare.

Esempio di comando ADT -sign

```
adt -sign -storetype pkcs12 -keystore cert.p12 unsignedMyApp.airi myApp.air
```

Per ulteriori informazioni, vedete “Comando ADT sign” a pagina 179.

Firma di una versione aggiornata di un'applicazione AIR

Ogni volta che create una versione aggiornata di un'applicazione AIR esistente, firmate l'applicazione aggiornata. Nel caso migliore, potete utilizzare lo stesso certificato per firmare la versione aggiornata utilizzata per firmare la versione precedente. In tal caso, la firma è esattamente uguale a quella utilizzata per la prima volta nell'applicazione.

Se il certificato utilizzato per firmare la versione precedente dell'applicazione è scaduto ed è stato rinnovato o sostituito, potete utilizzare il certificato rinnovato o nuovo (in sostituzione) per firmare la versione aggiornata. Per fare questo, firmate l'applicazione con il nuovo certificato e applicate una firma di migrazione utilizzando il certificato originale. La firma di migrazione conferma che il titolare del certificato originale ha pubblicato l'aggiornamento.

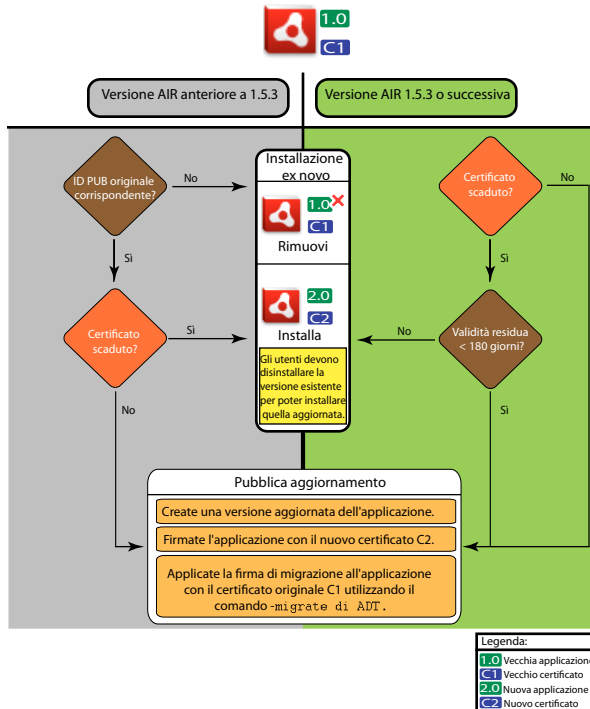
Prima di applicare una firma di migrazione, considerate i punti seguenti:

- Per poter applicare una firma di migrazione, il certificato originale deve essere ancora valido o comunque deve essere scaduto da non più di 365 giorni. Questo periodo di tempo è denominato "periodo di tolleranza" e la sua durata potrebbe cambiare in futuro.
Nota: fino a AIR 2.6, il periodo di tolleranza era di 180 giorni.
- Non è possibile applicare una firma di migrazione dopo che il certificato è scaduto ed è trascorso anche il periodo di tolleranza di 365 giorni. In questo caso, gli utenti devono disinstallare la versione esistente prima di installare quella aggiornata.
- Il periodo di tolleranza di 365 giorni vale solo per le applicazioni nelle quali è specificata una versione di AIR 1.5.3 (o superiore) nello spazio dei nomi del descrittore applicazione.

Importante: la firma di aggiornamenti con firme di migrazione associate a certificati scaduti è una soluzione solo temporanea. Per una soluzione completa, create un flusso di lavoro di firma standardizzato per gestire la distribuzione degli aggiornamenti delle applicazioni. Ad esempio, potete firmare ciascun aggiornamento con il certificato più recente e applicare un certificato di migrazione utilizzando il certificato utilizzato per firmare l'aggiornamento precedente (se disponibile). Caricate ciascun aggiornamento nella relativa URL dalla quale gli utenti possono scaricare l'applicazione. Per ulteriori informazioni, vedete “Flusso di lavoro di firma per gli aggiornamenti delle applicazioni” a pagina 269.

L'immagine e la tabella seguenti offrono un riepilogo del flusso di lavoro per le firme di migrazione:

Scenario	Stato del certificato originale	Azione dello sviluppatore	Azione dell'utente
Applicazione basata sul runtime Adobe AIR versione 1.5.3 o successiva	Valido	Pubblicare l'ultima versione dell'applicazione AIR	Nessuna azione richiesta Upgrade automatico dell'applicazione
	Scaduto, ma entro il periodo di tolleranza di 365 giorni	Firmare l'applicazione con il nuovo certificato. Applicare una firma di migrazione utilizzando il certificato scaduto.	Nessuna azione richiesta Upgrade automatico dell'applicazione
	Scaduto e non nel periodo di tolleranza	Non è possibile applicare la firma di migrazione all'aggiornamento dell'applicazione AIR. Dovete pubblicare un'altra versione dell'applicazione AIR utilizzando un nuovo certificato. Gli utenti possono installare la nuova versione dopo aver disinstallato la versione esistente dell'applicazione AIR.	Disinstallare la versione corrente dell'applicazione AIR e installare la versione più recente
<ul style="list-style-type: none"> Applicazione basata sul runtime Adobe AIR versione 1.5.2 o precedente L'ID editore nel descrittore dell'applicazione e dell'aggiornamento corrisponde all'ID editore della versione precedente 	Valido	Pubblicare l'ultima versione dell'applicazione AIR	Nessuna azione richiesta Upgrade automatico dell'applicazione
	Scaduto e non nel periodo di tolleranza	Non è possibile applicare la firma di migrazione all'aggiornamento dell'applicazione AIR. Dovete pubblicare un'altra versione dell'applicazione AIR utilizzando un nuovo certificato. Gli utenti possono installare la nuova versione dopo aver disinstallato la versione esistente dell'applicazione AIR.	Disinstallare la versione corrente dell'applicazione AIR e installare la versione più recente
<ul style="list-style-type: none"> Applicazione basata sul runtime Adobe AIR versione 1.5.2 o precedente L'ID editore nel descrittore dell'applicazione e dell'aggiornamento non corrisponde all'ID editore della versione precedente 	Qualsiasi	Firmare l'applicazione AIR tramite un certificato valido e pubblicare la versione più recente dell'applicazione AIR	Disinstallare la versione corrente dell'applicazione AIR e installare la versione più recente



Flusso di lavoro di firma per gli aggiornamenti

Migrazione di un'applicazione AIR per l'utilizzo di un nuovo certificato

Per migrare un'applicazione AIR in un nuovo certificato durante l'aggiornamento dell'applicazione:

- 1 Create un aggiornamento dell'applicazione
- 2 Create un pacchetto e firmate il file di aggiornamento di AIR con il **nuovo** certificato.
- 3 Firmate nuovamente il file AIR con il certificato **originale**, mediante il comando `-migrate`.

Un file AIR firmato con il comando `-migrate` può inoltre essere utilizzato per installare una nuova versione dell'applicazione, oltre a essere utilizzato per aggiornare qualsiasi versione precedente firmata con il vecchio certificato.

Nota: quando aggiornate un'applicazione pubblicata per una versione di AIR anteriore alla 1.5.3, specificate l'ID editore originale nel descrittore dell'applicazione. In caso contrario, gli utenti dell'applicazione dovranno disinstallare la versione precedente prima di installare l'aggiornamento.

Utilizzate il comando ADT `-migrate` con la seguente sintassi:

```
adt -migrate SIGNING_OPTIONS air_file_in air_file_out
```

- **SIGNING_OPTIONS:** le opzioni di firma identificano la chiave privata e il certificato da utilizzare per firmare il file AIR. Queste opzioni devono identificare il certificato per la firma **originale** e sono descritte in “Opzioni di firma codice ADT” a pagina 185.
- **air_file_in:** il file AIR per l'aggiornamento, firmato mediante il **nuovo** certificato.
- **air_file_out:** il file AIR da creare.

Nota: i nomi di file utilizzati per i file AIR di input e di output devono essere diversi.

Nell'esempio seguente viene illustrata la chiamata ad ADT con il flag `-migrate` per applicare una firma di migrazione a una versione aggiornata di un'applicazione AIR:

```
adt -migrate -storetype pkcs12 -keystore cert.p12 myAppIn.air myApp.air
```

Nota: il comando `-migrate` è stato aggiunto ad ADT nella versione AIR 1.1.

Migrazione di un'applicazione AIR del programma di installazione nativo per l'utilizzo di un nuovo certificato

In un'applicazione AIR pubblicata come programma di installazione nativo (ad esempio, un'applicazione in cui viene utilizzato l'API di estensione nativa) non può essere eseguita la firma tramite il comando ADT `-migrate`, in quanto si tratta di un'applicazione nativa specifica della piattaforma, non di un file `.air`. Invece, per migrare un'applicazione AIR pubblicata come estensione nativa in un nuovo certificato, procedere come descritto di seguito:

- 1 Create un aggiornamento dell'applicazione.
- 2 Assicuratevi che nel vostro file descrittore dell'applicazione (`app.xml`), il tag `<supportedProfiles>` includa il profilo `desktop` e il profilo `extendedDesktop` (oppure rimuovete il tag `<supportedProfiles>` dal descrittore dell'applicazione).
- 3 Create un pacchetto e firmate l'applicazione di aggiornamento **come file `.air`** mediante il comando ADT `-package` con il **nuovo** certificato.
- 4 Applicate il certificato di migrazione al file `.air` mediante il comando ADT `-migrate` con il certificato **originale** (come descritto in precedenza in [“Migrazione di un'applicazione AIR per l'utilizzo di un nuovo certificato”](#) a pagina 208).
- 5 Create un pacchetto del file `.air` in un programma di installazione nativo mediante il comando ADT `-package` con il flag `-target native`. Poiché l'applicazione è già firmata, non è necessario specificare un certificato di firma in questo passaggio.

Nel seguente esempio sono illustrati i punti 3-5 di questo processo. Il codice chiama ADT con i comandi `-package`, `-migrate`, quindi nuovamente con il comando `-package` per creare un pacchetto di una versione aggiornata di un'applicazione AIR come programma di installazione nativo:

```
adt -package -storetype pkcs12 -keystore new_cert.p12 myAppUpdated.air myApp.xml myApp.swf
adt -migrate -storetype pkcs12 -keystore original_cert.p12 myAppUpdated.air myAppMigrate.air
adt -package -target native myApp.exe myAppMigrate.air
```

Migrazione di un'applicazione AIR che utilizza un'estensione nativa per l'utilizzo di un nuovo certificato

Un'applicazione AIR che utilizza un'estensione nativa non può essere firmata mediante il comando ADT `-migrate`. Non può inoltre essere migrata tramite la procedura per la migrazione di un'applicazione AIR del programma di installazione nativo, in quanto non può essere pubblicata come file `.air` intermedio. Se invece si desidera migrare un'applicazione AIR che utilizza un'estensione nativa in un nuovo certificato, procedere come descritto di seguito:

- 1 Create un aggiornamento dell'applicazione
- 2 Create un pacchetto e firmate il programma di installazione nativo di aggiornamento mediante il comando ADT `-package`. Create un pacchetto dell'applicazione con il **nuovo** certificato e includete il flag `-migrate`, specificando il certificato **originale**.

Utilizzate la seguente sintassi per chiamare il comando ADT `-package` con il flag `-migrate`:

```
adt -package AIR_SIGNING_OPTIONS -migrate MIGRATION_SIGNING_OPTIONS -target package_type  
NATIVE_SIGNING_OPTIONS output app_descriptor FILE_OPTIONS
```

- **AIR_SIGNING_OPTIONS**: le opzioni di firma identificano la chiave privata e il certificato da utilizzare per firmare il file AIR. Queste opzioni identificano il **nuovo** certificato di firma e sono descritte in “Opzioni di firma codice ADT” a pagina 185.
- **MIGRATION_SIGNING_OPTIONS**: le opzioni di firma identificano la chiave privata e il certificato da utilizzare per firmare il file AIR. Queste opzioni identificano il certificato di firma **originale** e sono descritte in “Opzioni di firma codice ADT” a pagina 185.
- Le altre opzioni sono le stesse utilizzate per la creazione di pacchetti di un'applicazione AIR di un programma di installazione nativo e sono descritte in “Comando ADT package” a pagina 172.

Nel seguente esempio viene illustrata la chiamata ad ADT con il comando `-package` e il flag `-migrate` per creare un pacchetto di una versione aggiornata di un'applicazione AIR che utilizza un'estensione nativa e applicare una firma di migrazione all'aggiornamento:

```
adt -package -storetype pkcs12 -keystore new_cert.p12 -migrate -storetype pkcs12 -keystore  
original_cert.p12 -target native myApp.exe myApp.xml myApp.swf
```

Nota: il flag `-migrate` del comando `-package` è disponibile in ADT in AIR 3.6 e versioni successive.

Creazione di un certificato autofirmato mediante ADT

Potete utilizzare i certificati autofirmati per produrre un file di installazione AIR valido. I certificati autofirmati, tuttavia, forniscono agli utenti rassicurazioni limitate in materia di sicurezza, dal momento che la loro autenticità non può essere verificata. Quando viene installato un file AIR autofirmato, le informazioni sull'editore vengono visualizzate dall'utente come Sconosciuto. Un certificato generato da ADT è valido per cinque anni.

Se si crea un aggiornamento per un'applicazione AIR che era stata firmata mediante un certificato autofirmato, è necessario utilizzare lo stesso certificato per firmare entrambi i file AIR, originale e aggiornamento. I certificati prodotti da ADT sono sempre unici, anche se vengono utilizzati gli stessi parametri. Pertanto, se desiderate autofirmare gli aggiornamenti con un certificato generato da ADT, conservate il certificato originale in una posizione sicura. Inoltre, dopo che il certificato originale generato da ADT è scaduto, non sarà possibile produrre un file AIR aggiornato, quindi potrete pubblicare nuove applicazioni con un certificato diverso, ma non nuove versioni della stessa applicazione.

Importante: a causa delle limitazioni dei certificati autofirmati, per la firma di applicazioni AIR rilasciate pubblicamente Adobe consiglia di utilizzare un certificato commerciale rilasciato da un'autorità di certificazione considerata attendibile.

Il certificato e la chiave privata associata generata da ADT sono memorizzati in un file archivio di chiavi di tipo PKCS12. La password specificata è impostata sulla chiave stessa, non sull'archivio di chiavi.

Esempi di creazione di certificati

```
adt -certificate -cn SelfSign -ou QE -o "Example, Co" -c US 2048-RSA newcert.p12 39#wnetx3tl  
adt -certificate -cn ADigitalID 1024-RSA SigningCert.p12 39#wnetx3tl
```

Per firmare i file AIR mediante questi certificati, si utilizzano le seguenti opzioni di firma mediante i comandi -package o -prepare di ADT:

```
-storetype pkcs12 -keystore newcert.p12 -storepass 39#wnetx3t1  
-storetype pkcs12 -keystore SigningCert.p12 -storepass 39#wnetx3t1
```

Nota: le versioni Java dalla 1.5 in poi non accettano i caratteri high-ASCII nelle password utilizzate per proteggere i file di certificato PKCS12. Utilizzate quindi solo caratteri ASCII standard nella password.

Capitolo 14: File descrittore delle applicazioni AIR

Ogni applicazione AIR richiede un file descrittore dell'applicazione. Il file descrittore dell'applicazione è un documento XML che definisce le proprietà di base dell'applicazione.

In molti ambienti di sviluppo che supportano AIR viene generato automaticamente un descrittore di applicazione quando create un progetto. In caso contrario, dovete creare un file descrittore personalizzato. Un file descrittore di esempio, `descriptor-sample.xml`, è disponibile nella directory `samples` di AIR e Flex SDK.

Potete utilizzare un nome file qualsiasi per il file descrittore dell'applicazione. Quando create un pacchetto dell'applicazione, il file descrittore dell'applicazione viene rinominato in `application.xml` e inserito in una directory speciale all'interno del pacchetto AIR.

Esempio di descrittore applicazione

Il seguente documento descrittore dell'applicazione imposta le proprietà di base utilizzate dalla maggior parte delle applicazioni AIR:

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/3.0">
  <id>example.HelloWorld</id>
  <versionNumber>1.0.1</versionNumber>
  <filename>Hello World</filename>
  <name>Example Co. AIR Hello World</name>
  <description>
    <text xml:lang="en">This is an example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
  </description>
  <copyright>Copyright (c) 2010 Example Co.</copyright>
  <initialWindow>
    <title>Hello World</title>
    <content>
      HelloWorld.swf
    </content>
  </initialWindow>
  <icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.png</image32x32>
    <image48x48>icons/bigIcon.png</image48x48>
    <image128x128>icons/biggerIcon.png</image128x128>
  </icon>
</application>
```

Se come contenuto principale dell'applicazione viene utilizzato un file HTML anziché un file SWF, solo l'elemento `<content>` è diverso:

```
<content>
  HelloWorld.html
</content>
```

Modifiche del descrittore dell'applicazione

Il descrittore dell'applicazione AIR è cambiato nelle seguenti release di AIR.

Modifiche del descrittore in AIR 1.1

Possibilità di localizzare gli elementi `name` e `description` dell'applicazione utilizzando l'elemento `text`.

Modifiche del descrittore in AIR 1.5

`contentType` è diventato un elemento secondario obbligatorio di `fileType`.

Modifiche del descrittore in AIR 1.5.3

Elemento `publisherID` aggiunto per consentire alle applicazioni di specificare un valore di ID editore.

Modifiche del descrittore in AIR 2.0

Aggiunti:

- `aspectRatio`
- `autoOrients`
- `fullScreen`
- `image29x29` (vedete `imageNxN`)
- `image57x57`
- `image72x72`
- `image512x512`
- `iPhone`
- `renderMode`
- `supportedProfiles`

Modifiche del descrittore in AIR 2.5

Rimosso: `version`

Aggiunti:

- `android`
- `extensionID`
- `extensions`
- `image36x36` (vedete `imageNxN`)
- `manifestAdditions`
- `versionLabel`
- `versionNumber`

Modifiche del descrittore in AIR 2.6

Aggiunti:

- `image114x114` (vedete `imageNxN`)
- `requestedDisplayResolution`
- `softKeyboardBehavior`

Modifiche del descrittore in AIR 3.0

Aggiunti:

- `colorDepth`
- `direct` come valore valido per `renderMode`
- `renderMode` non viene più ignorato per le piattaforme desktop
- L'elemento Android `<uses-sdk>` può essere specificato. (In precedenza non era consentito.)

Modifiche del descrittore in AIR 3.1

Aggiunti:

- “`Entitlements`” a pagina 227

Modifiche del descrittore in AIR 3.2

Aggiunti:

- `depthAndStencil`
- `supportedLanguages`

Modifiche del descrittore in AIR 3.3

Aggiunti:

- `aspectRatio` ora include l'opzione `ANY`.

Modifiche del descrittore in AIR 3.4

Aggiunti:

- `image50x50` (vedete “`imageNxN`” a pagina 235)
- `image58x58` (vedete “`imageNxN`” a pagina 235)
- `image100x100` (vedete “`imageNxN`” a pagina 235)
- `image1024x1024` (vedete “`imageNxN`” a pagina 235)

Modifiche del descrittore in AIR 3.6

Aggiunti:

- L'elemento “`requestedDisplayResolution`” a pagina 245 in “`iPhone`” a pagina 239 include ora l'attributo `excludeDevices` che consente di specificare quali iOS di destinazione utilizzano la risoluzione elevata o standard

- Il nuovo elemento “[requestedDisplayResolution](#)” a pagina 245 in “[initialWindow](#)” a pagina 236 specifica se utilizzare la risoluzione elevata o standard sulle piattaforme desktop come Mac con display ad alta risoluzione

Modifiche del descrittore in AIR 3.7

Aggiunti:

- L'elemento “[iPhone](#)” a pagina 239 include ora un elemento “[externalSwfs](#)” a pagina 228, che consente di specificare un elenco di file SWF caricabili in fase di runtime.
- L'elemento “[iPhone](#)” a pagina 239 include ora un elemento “[forceCPURenderModeForDevices](#)” a pagina 232, che consente di imporre la modalità di rendering CPU per un insieme di dispositivi specificato.

Struttura del file descrittore dell'applicazione

Il file descrittore dell'applicazione è un documento XML che presenta la struttura seguente:

```
<application xmlns="http://ns.adobe.com/air/application/3.0">
  <allowBrowserInvocation>...</allowBrowserInvocation>
  <android>
    <colorDepth>...</colorDepth>
    <manifestAdditions
      <manifest>...</manifest>
    ]]>
  </android>
  <copyright>...</copyright>
  <customUpdateUI>...</
  <description>
    <text xml:lang="...">...</text>
  </description>
  <extensions>
    <extensionID>...</extensionID>
  </extensions>
  <filename>...</filename>
  <fileTypes>
    <fileType>
      <contentType>...</contentType>
      <description>...</description>
      <extension>...</extension>
      <icon>
        <imageNxN>...</imageNxN>
      </icon>
      <name>...</name>
    </fileType>
  </fileTypes>
  <icon>
    <imageNxN>...</imageNxN>
  </icon>
  <id>...</id>
  <initialWindow>
    <aspectRatio>...</aspectRatio>
    <autoOrients>...</autoOrients>
    <content>...</content>
```

```
<depthAndStencil>...</depthAndStencil>
<fullScreen>...</fullScreen>
<height>...</height>
<maximizable>...</maximizable>
<maxSize>...</maxSize>
<minimizable>...</minimizable>
<minSize>...</minSize>
<renderMode>...</renderMode>
<requestedDisplayResolution>...</requestedDisplayResolution>
<resizable>...</resizable>
<softKeyboardBehavior>...</softKeyboardBehavior>
<systemChrome>...</systemChrome>
<title>...</title>
<transparent>...</transparent>
<visible>...</visible>
<width>...</width>
<x>...</x>
<y>...</y>
</initialWindow>
<installFolder>...</installFolder>
<iPhone>
  <Entitlements>...</Entitlements>
  <InfoAdditions>...</InfoAdditions>
  <requestedDisplayResolution>...</requestedDisplayResolution>
  <forceCPURenderModeForDevices>...</forceCPURenderModeForDevices>
  <externalSwfs>...</externalSwfs>
</iPhone>
<name>
  <text xml:lang="...">...</text>
</name>
<programMenuFolder>...</programMenuFolder>
<publisherID>...</publisherID>
<"supportedLanguages" a pagina 247>...</"supportedLanguages" a pagina 247>
<supportedProfiles>...</supportedProfiles>
<versionNumber>...</versionNumber>
<versionLabel>...</versionLabel>
</application>
```

Elementi del descrittore dell'applicazione AIR

Il seguente dizionario di elementi descrive ciascuno degli elementi validi di un file descrittore dell'applicazione AIR.

allowBrowserInvocation

Adobe AIR 1.0 e versioni successive - opzionale

Consente all'API AIR interna al browser di rilevare e avviare l'applicazione.

Se impostate questo valore su `true`, tenete presenti le implicazioni di sicurezza descritte in [Chiamata di un'applicazione AIR dal browser](#) (sviluppatori ActionScript) e [Invoking an AIR application from the browser](#) (sviluppatori HTML).

Per maggiori informazioni, vedete “[Avvio dal browser di un'applicazione AIR installata](#)” a pagina 265.

Elemento superiore: [“application”](#) a pagina 217

Elementi secondari: nessuno

Contenuto

true o false (predefinito)

Esempio

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

android

Adobe AIR 2.5 e versioni successive - opzionale

Consente di aggiungere elementi al file manifest di Android. AIR crea il file AndroidManifest.xml per ogni pacchetto APK. Potete usare l'elemento android nel descrittore dell'applicazione AIR per aggiungere ad esso ulteriori voci. Ignorato su tutte le piattaforme eccetto Android.

Elemento superiore: [“application”](#) a pagina 217

Elementi secondari:

- [“colorDepth”](#) a pagina 222
- [“manifestAdditions”](#) a pagina 240

Contenuto

Elementi che definiscono le proprietà specifiche di Android da aggiungere al manifest dell'applicazione Android.

Esempio

```
<android>  
  <manifestAdditions>  
    ...  
  </manifestAdditions>  
</android>
```

Altri argomenti presenti nell’Aiuto

[“Impostazioni Android”](#) a pagina 79

[Il file AndroidManifest.xml](#)

application

Adobe AIR 1.0 e versioni successive - obbligatorio

L'elemento principale (root) di un documento descrittore dell'applicazione AIR.

Elemento superiore: nessuno

Elementi secondari:

- [“allowBrowserInvocation”](#) a pagina 216
- [“android”](#) a pagina 217

- “copyright” a pagina 224
- “customUpdateUI” a pagina 224
- “description” a pagina 225
- “extensions” a pagina 228
- “filename” a pagina 229
- “fileTypes” a pagina 230
- “icon” a pagina 233
- “id” a pagina 234
- “initialWindow” a pagina 236
- “installFolder” a pagina 238
- “iPhone” a pagina 239
- “name” a pagina 242
- “programMenuFolder” a pagina 244
- “publisherID” a pagina 244
- “supportedLanguages” a pagina 247
- “supportedProfiles” a pagina 248
- “version” a pagina 250
- “versionLabel” a pagina 251
- “versionNumber” a pagina 251

Attributi

minimumPatchLevel - Il livello di patch minimo del runtime AIR richiesto per questa applicazione.

xmlns - L'attributo dello spazio dei nomi XML determina la versione del runtime AIR richiesta per l'applicazione.

Lo spazio dei nomi cambia a ogni nuova versione principale di AIR che viene rilasciata (ma non con le patch secondarie). L'ultimo segmento dello spazio dei nomi, ad esempio “3.0”, indica la versione del runtime richiesta dall'applicazione.

I valori xmlns delle principali versioni di AIR sono:

```
xmlns="http://ns.adobe.com/air/application/1.0"  
xmlns="http://ns.adobe.com/air/application/1.1"  
xmlns="http://ns.adobe.com/air/application/1.5"  
xmlns="http://ns.adobe.com/air/application/1.5.2"  
xmlns="http://ns.adobe.com/air/application/1.5.3"  
xmlns="http://ns.adobe.com/air/application/2.0"  
xmlns="http://ns.adobe.com/air/application/2.5"  
xmlns="http://ns.adobe.com/air/application/2.6"  
xmlns="http://ns.adobe.com/air/application/2.7"  
xmlns="http://ns.adobe.com/air/application/3.0"  
xmlns="http://ns.adobe.com/air/application/3.1"  
xmlns="http://ns.adobe.com/air/application/3.2"  
xmlns="http://ns.adobe.com/air/application/3.3"  
xmlns="http://ns.adobe.com/air/application/3.4"  
xmlns="http://ns.adobe.com/air/application/3.5"  
xmlns="http://ns.adobe.com/air/application/3.6"  
xmlns="http://ns.adobe.com/air/application/3.7"
```

Per le applicazioni basate su SWF la versione del runtime AIR specificata nel descrittore dell'applicazione determina la versione SWF massima che può essere caricata come contenuto iniziale dell'applicazione. Per le applicazioni per cui è specificato AIR 1.0 o AIR 1.1 potete utilizzare solo file SWF9 (Flash Player 9) come contenuto iniziale, anche se per l'esecuzione utilizzate il runtime AIR 2. Per le applicazioni per cui è specificato AIR 1.5 (o versione successiva) potete utilizzare file SWF9 o SWF10 (Flash Player 10) come contenuto iniziale.

La versione SWF determina la versione disponibile delle API di AIR e Flash Player. Se come contenuto iniziale di un'applicazione AIR 1.5 viene utilizzato un file SWF9, l'applicazione avrà accesso solo alle API di AIR 1.1 e Flash Player 9. Inoltre, le modifiche di comportamento apportate alle API esistenti in AIR 2.0 o Flash Player 10.1 non verranno applicate. Le modifiche importanti relative alla sicurezza apportate alle API rappresentano un'eccezione a questa condizione e possono essere applicate in modo retroattivo in patch presenti o future del runtime.

Per le applicazioni basate su HTML la versione del runtime specificata nel descrittore dell'applicazione determina la versione delle API di AIR e Flash Player disponibili per l'applicazione. Il comportamento del codice HTML, CSS e JavaScript è sempre determinato dalla versione del WebKit utilizzato nel runtime AIR installato e non dal descrittore dell'applicazione.

Quando un'applicazione AIR carica contenuto SWF, la versione delle API di AIR e Flash Player disponibile per il contenuto dipende dalla modalità di caricamento del contenuto. A volte la versione reale è determinata dallo spazio dei nomi del descrittore dell'applicazione, a volte dalla versione del contenuto che esegue il caricamento e a volte dalla versione del contenuto caricato. La tabella seguente illustra in che modo viene determinata la versione delle API in base al metodo di caricamento:

Modalità di caricamento del contenuto	Modalità di determinazione della versione delle API
Contenuto iniziale, applicazione basata su SWF	Versione SWF del file caricato
Contenuto iniziale, applicazione basata su HTML	Spazio dei nomi del descrittore dell'applicazione
SWF caricato da contenuto SWF	Versione del contenuto che esegue il caricamento

Modalità di caricamento del contenuto	Modalità di determinazione della versione delle API
Libreria SWF caricata da contenuto HTML mediante tag <script>	Spazio dei nomi del descrittore dell'applicazione
SWF caricato da contenuto HTML mediante API di AIR o Flash Player (ad esempio flash.display.Loader)	Spazio dei nomi del descrittore dell'applicazione
SWF caricato da contenuto HTML mediante tag <object> o <embed> (o le API JavaScript equivalenti)	Versione SWF del file caricato

Quando caricate un file SWF di versione diversa rispetto al contenuto che esegue il caricamento, potreste incontrare due problemi:

- Caricamento di un SWF di una versione più recente con un SWF di una versione meno recente - I riferimenti alle API aggiunte nelle nuove versioni di AIR e Flash Player nel contenuto caricato non saranno risolti.
- Caricamento di un SWF di una versione precedente da parte di un SWF di una versione più recente - Alcune API modificate nelle ultime versioni di AIR e Flash Player potrebbe comportarsi in modo inatteso per il contenuto caricato.

Contenuto

L'elemento application contiene elementi secondari che definiscono le proprietà di un'applicazione AIR.

Esempio

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/3.0">
  <id>HelloWorld</id>
  <version>2.0</version>
  <filename>Hello World</filename>
  <name>Example Co. AIR Hello World</name>
  <description>
    <text xml:lang="en">This is an example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
  </description>
  <copyright>Copyright (c) 2010 Example Co.</copyright>
  <initialWindow>
    <title>Hello World</title>
    <content>
      HelloWorld.swf
    </content>
    <systemChrome>none</systemChrome>
    <transparent>true</transparent>
    <visible>true</visible>
    <minSize>320 240</minSize>
  </initialWindow>
  <installFolder>Example Co/Hello World</installFolder>
  <programMenuFolder>Example Co</programMenuFolder>
  <icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.png</image32x32>
  </icon>
</application>
```

```
<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>
```

aspectRatio

Adobe AIR 2.0 e versioni successive, iOS e Android - opzionale

Specifica le proporzioni dell'applicazione.

Se non è specificato, l'applicazione si apre con le proporzioni e l'orientamento "naturali" del dispositivo. L'orientamento naturale varia da dispositivo a dispositivo. In genere, è l'orientamento verticale nei dispositivi con display di piccole dimensioni, come i telefoni. Su alcuni dispositivi, come il tablet iPad, l'applicazione si apre con l'orientamento corrente. In AIR 3.3 e versioni successive, questa impostazione si applica all'intera applicazione, non solo alla visualizzazione iniziale.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

portrait, landscape o any

Esempio

```
<aspectRatio>landscape</aspectRatio>
```

autoOrients

Adobe AIR 2.0 e versioni successive, iOS e Android - opzionale

Specifica se il contenuto dell'applicazione si riorienta automaticamente quando l'orientamento fisico del dispositivo cambia. Per maggiori informazioni, consultate [Orientamento stage](#).

Quando utilizzate l'orientamento automatico, valutate se impostare le proprietà `align` e `scaleMode` dello Stage sui valori seguenti:

```
stage.align = StageAlign.TOP_LEFT;
stage.scaleMode = StageScaleMode.NO_SCALE;
```

Queste impostazioni consentono all'applicazione di ruotare intorno all'angolo superiore sinistro e impediscono che il contenuto dell'applicazione venga ridimensionato automaticamente. Anche se le altre modalità di ridimensionamento adattano il contenuto in base alle dimensioni dello stage ruotato, hanno anche l'effetto di troncatura, distorcere o ridurre eccessivamente il contenuto stesso. Il risultato migliore si può ottenere quasi sempre ridisegnando manualmente il contenuto.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

true o false (predefinito)

Esempio

```
<autoOrients>true</autoOrients>
```

colorDepth

Adobe AIR 3 e versioni successive - opzionale

Specifica se utilizzare il colore a 16 bit o a 32 bit.

L'uso del colore a 16 bit può migliorare le prestazioni di rendering, ma con una riduzione della fedeltà dei colori. Prima di AIR 3, il colore a 16 bit veniva sempre utilizzato in Android. In AIR 3, viene utilizzato per impostazione predefinita il colore a 32 bit.

***Nota:** se l'applicazione utilizza la classe `StageVideo`, dovete usare il colore a 32 bit.*

Elemento superiore: “[android](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Uno dei seguenti valori:

- 16 bit
- 32 bit

Esempio

```
<android>  
  <colorDepth>16bit</colorDepth>  
  <manifestAdditions>...</manifestAdditions>  
</android>
```

containsVideo

Specifica se l'applicazione conterrà o meno un contenuto video.

Elemento superiore: “[android](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Uno dei seguenti valori:

- true
- false

Esempio

```
<android>
  <containsVideo>true</containsVideo>
  <manifestAdditions>...</manifestAdditions>
</android>
```

content

Adobe AIR 1.0 e versioni successive - obbligatorio

Il valore specificato per l'elemento `content` è l'URL del file di contenuto principale dell'applicazione. Può trattarsi di un file SWF o HTML. L'URL è specificato in relazione al livello principale della cartella di installazione dell'applicazione. (Quando si esegue un'applicazione AIR con ADL, l'URL è relativo alla cartella contenente il file descrittore dell'applicazione. È possibile utilizzare il parametro `root-dir` di ADL per specificare una directory principale diversa.)

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Un URL relativo alla directory dell'applicazione, poiché il valore dell'elemento `content` viene trattato come URL, i caratteri nel file del contenuto devono essere codificati come URL in base alle regole definite in [RFC 1738](#). Ad esempio, i caratteri di spazio devono essere codificati come `%20`.

Esempio

```
<content>TravelPlanner.swf</content>
```

contentType

Adobe AIR da 1.0 a 1.1 - opzionale; AIR 1.5 e versioni successive - obbligatorio

`contentType` è obbligatorio a partire da AIR 1.5 (era facoltativo in AIR 1.0 e 1.1). La proprietà permette ad alcuni sistemi operativi di individuare più facilmente l'applicazione più indicata per l'apertura di un file. Il valore deve essere il tipo MIME del contenuto del file. Tenete presente che il valore viene ignorato in Linux se il tipo di file è già registrato ed è stato associato a un tipo MIME.

Elemento superiore: “[fileType](#)” a pagina 230

Elementi secondari: nessuno

Contenuto

Il tipo e sottotipo MIME. Vedete [RFC2045](#) per maggiori informazioni sui tipi MIME.

Esempio

```
<contentType>text/plain</contentType>
```

copyright

Adobe AIR 1.0 e versioni successive - opzionale

Le informazioni di copyright per l'applicazione AIR. In Mac OS, il testo di copyright appare nella finestra di dialogo Informazioni su per l'applicazione installata. In Mac OS, le informazioni di copyright sono utilizzate anche nel campo NSHumanReadableCopyright del file Info.plist per l'applicazione.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Una stringa contenente le informazioni sul copyright dell'applicazione.

Esempio

```
<copyright>© 2010, Examples, Inc.All rights reserved.</copyright>
```

customUpdateUI

Adobe AIR 1.0 e versioni successive - opzionale

Indica se un'applicazione fornisce le proprie finestre di aggiornamento. Se è `false`, AIR presenta all'utente le finestre di aggiornamento standard. Solo le applicazioni distribuite come file AIR possono utilizzare il sistema di aggiornamento incorporato di AIR.

Quando la versione installata dell'applicazione ha l'elemento `customUpdateUI` impostato su `true` e l'utente fa doppio clic sul file AIR di una nuova versione o installa un aggiornamento dell'applicazione mediante la funzionalità di installazione invisibile, il runtime apre la versione installata dell'applicazione, anziché il programma di installazione dell'applicazione AIR predefinito. La logica dell'applicazione può quindi determinare come procedere con l'operazione di aggiornamento. (Affinché sia possibile procedere all'aggiornamento, l'ID applicazione e l'ID editore del file AIR devono corrispondere ai valori dell'applicazione installata.)

***Nota:** il meccanismo `customUpdateUI` entra in funzione solo quando l'applicazione è già installata e l'utente fa doppio clic sul file di installazione AIR contenente un aggiornamento oppure installa un aggiornamento dell'applicazione mediante la funzionalità di installazione invisibile. Potete scaricare e avviare un aggiornamento mediante una logica di applicazione personalizzata, visualizzando la vostra interfaccia utente nel modo desiderato, a prescindere dal fatto che `customUpdateUI` sia impostato su `true` o meno.*

Per ulteriori informazioni, vedete “[Aggiornamento di applicazioni AIR](#)” a pagina 267.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

`true` o `false` (predefinito)

Esempio

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

depthAndStencil

Adobe AIR 3.2 e versioni successive - opzionale

Indica che l'applicazione richiede l'uso del buffer di profondità o di stencil. Solitamente questi buffer vengono utilizzati quando si lavora con contenuti 3D. Per impostazione predefinita, il valore di questo elemento è `false`, che disattiva i buffer di profondità e di stencil. L'elemento è necessario perché i buffer devono essere allocati all'avvio dell'applicazione, prima che venga caricato qualsiasi contenuto.

L'impostazione di questo elemento deve corrispondere al valore passato per l'argomento `enableDepthAndStencil` del metodo `Context3D.configureBackBuffer()`. Se i valori non corrispondono, AIR genera un errore.

Questo elemento è applicabile solo quando `renderMode = direct`. Se `renderMode` non è uguale a `direct`, ADT genera l'errore 118:

```
<depthAndStencil> element unexpected for render mode cpu. It requires "direct" render mode.
```

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

`true` o `false` (predefinito)

Esempio

```
<depthAndStencil>true</depthAndStencil>
```

description

Adobe AIR 1.0 e versioni successive - opzionale

La descrizione dell'applicazione visualizzata nel programma di installazione dell'applicazione AIR.

Se si specifica un unico nodo di testo (non più elementi testo), il programma di installazione dell'applicazione AIR utilizza questa descrizione, a prescindere dalla lingua del sistema. In caso contrario, il programma di installazione dell'applicazione AIR utilizza la descrizione che più corrisponde alla lingua dell'interfaccia utente del sistema operativo dell'utente. Ad esempio, considerate un'installazione in cui l'elemento `description` del file descrittore dell'applicazione comprende un valore per la versione locale (inglese). Il programma di installazione dell'applicazione AIR utilizza la descrizione inglese se il sistema dell'utente identifica (inglese) come lingua dell'interfaccia utente. Utilizza la descrizione inglese anche se la lingua dell'interfaccia utente del sistema è en-US (inglese americano). Tuttavia, se la lingua dell'interfaccia utente del sistema è en-US e il file descrittore dell'applicazione definisce i nomi en-US ed en-GB, il programma di installazione dell'applicazione AIR utilizza il valore en-US. Se l'applicazione non definisce alcuna descrizione che corrisponda alla lingua dell'interfaccia utente del sistema, il programma di installazione dell'applicazione AIR utilizza il primo valore `description` definito nel file descrittore dell'applicazione.

Per ulteriori informazioni sullo sviluppo di applicazioni multilingue, vedete “[Localizzazione di applicazioni AIR](#)” a pagina 304.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: “[text](#)” a pagina 249

Contenuto

Lo schema descrittore dell'applicazione AIR 1.0 consente di definire un unico nodo di testo semplice per il nome (non più elementi `text`).

In AIR 1.1 (o versione successiva) potete invece specificare più lingue nell'elemento `description`. L'attributo `xml:lang` di ogni elemento di testo specifica un codice lingua, definito dallo standard [RFC4646](http://www.ietf.org/rfc/rfc4646.txt) (<http://www.ietf.org/rfc/rfc4646.txt>).

Esempio

Descrizione con nodo di testo semplice:

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

Descrizione con elementi di testo localizzati per inglese, francese e spagnolo (valida in AIR 1.1 e versioni successive).

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

description

Adobe AIR 1.0 e versioni successive - obbligatorio

La descrizione del tipo di file viene visualizzata all'utente dal sistema operativo. Non è localizzabile.

Vedete anche: “[description](#)” a pagina 225 come elemento secondario dell'elemento `application`

Elemento superiore: “[fileType](#)” a pagina 230

Elementi secondari: nessuno

Contenuto

Una stringa che descrive il contenuto del file.

Esempio

```
<description>PNG image</description>
```

embedFonts

Consente di utilizzare font personalizzati per StageText nell'applicazione AIR. Questo elemento è opzionale.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: “[font](#)” a pagina 231

Contenuto

L'elemento `embedFonts` può contenere un numero qualsiasi di elementi `font`.

Esempio

```
<embedFonts>
  <font>
    <fontPath>ttf/space_ age.ttf</fontPath>
    <fontName>space_ age</fontName>
  </font>
  <font>
    <fontPath>ttf/xminus.ttf</fontPath>
    <fontName>xminus</fontName>
  </font>
</embedFonts>
```

Entitlements

Adobe AIR 3.1 e versioni successive - solo per iOS, opzionale

iOS impiega proprietà denominate entitlements per consentire all'applicazione di accedere a risorse e funzionalità aggiuntive. Utilizzate l'elemento Entitlements per specificare questa informazione in un'applicazione iOS mobile.

Elemento superiore: “iPhone” a pagina 239

Elementi secondari: elementi iOS Entitlements.plist

Contenuto

Contiene gli elementi secondari che specificano le coppie chiave-valore da utilizzare come impostazioni Entitlements.plist per l'applicazione. Il contenuto dell'elemento Entitlements deve essere racchiuso in un blocco CDATA. Per maggiori informazioni, consultate l'articolo [Entitlement Key Reference](#) della libreria per sviluppatori iOS.

Esempio

```
<iphone>
...
  <Entitlements>
    <![CDATA[
      <key>aps-environment</key>
      <string>development</string>
    ]]>
  </Entitlements>
</iphone>
```

extension

Adobe AIR 1.0 e versioni successive - obbligatorio

La stringa di estensione di un tipo di file.

Elemento superiore: “fileType” a pagina 230

Elementi secondari: nessuno

Contenuto

Una stringa che identifica i caratteri dell'estensione file (senza il punto).

Esempio

```
<extension>png</extension>
```

extensionID

Adobe AIR 2.5 e versioni successive

Specifica l'ID di un'estensione ActionScript utilizzato dall'applicazione. L'ID è definito nel documento descrittore dell'estensione.

Elemento superiore: “[extensions](#)” a pagina 228

Elementi secondari: nessuno

Contenuto

Una stringa che identifica l'ID dell'estensione ActionScript.

Esempio

```
<extensionID>com.example.extendedFeature</extensionID>
```

extensions

Adobe AIR 2.5 e versioni successive - opzionale

Identifica le estensioni ActionScript utilizzate da un'applicazione.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: “[extensionID](#)” a pagina 228

Contenuto

Gli elementi secondari `extensionID` che contengono gli ID estensione ActionScript inclusi nel file descrittore dell'estensione.

Esempio

```
<extensions>
  <extensionID>extension.first</extensionID>
  <extensionID>extension.next</extensionID>
  <extensionID>extension.last</extensionID>
</extensions>
```

externalSwfs

Adobe AIR 3.7 e versioni successive, solo iOS - Opzionale

Specifica il nome di un file di testo contenente l'elenco di file SWF che verranno configurati da ADT per l'hosting remoto. Potete ridurre le dimensioni iniziali di download dell'applicazione creando un pacchetto con un sottoinsieme di SWF utilizzati dall'applicazione, quindi caricando i rimanenti SWF esterni (di sole risorse) in fase di runtime mediante il metodo `Loader.load()`. Per utilizzare questa funzione, create l'applicazione in modo che ADT sposti tutti i componenti ActionScript ByteCode (ABC) dai file SWF caricati esternamente all'SWF dell'applicazione principale, dando origine a un file SWF che contiene soltanto risorse. In tal modo viene rispettata la regola di Apple Store che non consente il download di codice aggiuntivo dopo l'installazione di un'applicazione.

Per ulteriori informazioni, consultate “[Riduzione delle dimensioni di download tramite il caricamento di SWF esterni di sole risorse](#)” a pagina 90.

Elemento superiore: “[iPhone](#)” a pagina 239, “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Nome di un file di testo contenente un elenco delimitato da linee di file SWF destinati all'hosting remoto.

Attributi:

Nessuno.

Esempi

iOS:

```
<iPhone>  
  <externalSwfs>FileContainingListofSWFs.txt</externalSwfs>  
</iPhone>
```

filename

Adobe AIR 1.0 e versioni successive - obbligatorio

La stringa da utilizzare come nome file dell'applicazione (senza estensione) quando viene installata l'applicazione. Il file dell'applicazione avvia l'applicazione AIR nel runtime. Se non viene fornito un valore `name`, `filename` viene utilizzato anche come nome della cartella di installazione.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

La proprietà `filename` può contenere qualsiasi carattere in formato Unicode (UTF-8) a eccezione di quelli seguenti, di cui non è consentito l'uso nei nomi file in vari file system:

Carattere	Codice esadecimale
<i>vari</i>	0x00 – x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E
?	x3F
\	x5C
	x7C

Il valore `filename` non può terminare con un punto.

Esempio

```
<filename>MyApplication</filename>
```

fileType

Adobe AIR 1.0 e versioni successive - opzionale

Descrive un singolo tipo di file per il quale l'applicazione può registrarsi.

Elemento superiore: “fileTypes” a pagina 230

Elementi secondari:

- “contentType” a pagina 223
- “description” a pagina 226
- “extension” a pagina 227
- “icon” a pagina 233
- “name” a pagina 243

Contenuto

Elementi che descrivono un tipo di file.

Esempio

```
<fileType>
  <name>foo.example</name>
  <extension>foo</extension>
  <description>Example file type</description>
  <contentType>text/plain</contentType>
  <icon>
    <image16x16>icons/fooIcon16.png</image16x16>
    <image48x48>icons/fooIcon48.png</image48x48>
  </icon>
</fileType>
```

fileTypes

Adobe AIR 1.0 e versioni successive - opzionale

L'elemento `fileTypes` consente di dichiarare i tipi di file ai quali è possibile associare un'applicazione AIR.

Quando un'applicazione AIR viene installata, qualsiasi tipo di file dichiarato viene registrato nel sistema operativo. Se questi tipi di file non sono già associati a un'altra applicazione, vengono associati all'applicazione AIR. Per sostituire un'associazione esistente tra un tipo di file e un'altra applicazione, utilizzate il metodo `NativeApplication.setAsDefaultApplication()` in fase runtime (preferibilmente con il permesso dell'utente).

Nota: i metodi runtime possono gestire le associazioni solo per i tipi di file dichiarati nel descrittore dell'applicazione.

L'elemento `fileTypes` è opzionale.

Elemento superiore: “application” a pagina 217

Elementi secondari: “fileType” a pagina 230

Contenuto

L'elemento `fileTypes` può contenere un numero qualsiasi di elementi `fileType`.

Esempio

```
<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>
```

font

Descrive un singolo font personalizzato che può essere utilizzato nell'applicazione AIR.

Elemento superiore: “[embedFonts](#)” a pagina 226

Elementi secondari: “[fontName](#)” a pagina 231, “[fontPath](#)” a pagina 231

Contenuto

Elementi che specificano il nome del font personalizzato e il relativo percorso.

Esempio

```
<font>
  <fontPath>ttf/space age.ttf</fontPath>
  <fontName>space age</fontName>
</font>
```

fontName

Specifica il nome di un font personalizzato.

Elemento superiore: “[font](#)” a pagina 231

Elementi secondari: nessuno

Contenuto

Nome del font personalizzato da specificare in `StageText.fontFamily`

Esempio

```
<fontName>space age</fontName>
```

fontPath

Percorso del file del font personalizzato.

Elemento superiore: “font” a pagina 231

Elementi secondari: nessuno

Contenuto

Percorso del file del font personalizzato (rispetto all'origine).

Esempio

```
<fontPath>ttf/space age.ttf</fontPath>
```

forceCPURenderModeForDevices

Adobe AIR 3.7 e versioni successive, solo iOS - Opzionale

Impone la modalità di rendering CPU per un insieme di dispositivi specificato. Questa funzione consente di abilitare in modo selettivo ed efficiente la modalità di rendering GPU per i rimanenti dispositivi iOS.

Aggiungete questo tag come tag secondario del tag `iPhone` e specificate un elenco separato da spazi contenente i nomi di modello dei dispositivi. I nomi di modello dei dispositivi validi includono i seguenti:

iPad 1,1	iPhone1,1	iPod1,1
iPad 2,1	iPhone1,2	iPod2,1
iPad 2,2	iPhone2,1	iPod3,3
iPad 2,3	iPhone3,1	iPod4,1
iPad 2,4	iPhone3,2	iPod5,1
iPad 2,5	iPhone4,1	
iPad 3,1	iPhone5,1	
iPad 3,2		
iPad 3,3		
iPad 3,4		

Elemento superiore: “iPhone” a pagina 239, “initialWindow” a pagina 236

Elementi secondari: nessuno

Contenuto

Elenco separato da spazi di nomi di modello dei dispositivi.

Attributi:

Nessuno.

Esempi

iOS:

```
...
<renderMode>GPU</renderMode>
...
<iPhone>
...
  <forceCPURenderModeForDevices>iPad1,1 iPhone1,1 iPhone1,2 iPod1,1
  </forceCPURenderModeForDevices>
</iPhone>
```

fullScreen

Adobe AIR 2.0 e versioni successive, iOS e Android - opzionale

Specifica se l'applicazione viene avviata in modalità a schermo intero.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

true o false (predefinito)

Esempio

```
<fullscreen>true</fullscreen>
```

height

Adobe AIR 1.0 e versioni successive - opzionale

L'altezza iniziale della finestra principale dell'applicazione.

Se non impostate l'altezza, questa viene determinata dalle impostazioni del file SWF principale oppure, nel caso di un'applicazione AIR basata su HTML, dal sistema operativo.

L'altezza massima di una finestra è cambiata da 2048 pixel a 4096 pixel in AIR 2.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Un numero intero positivo con un valore massimo di 4095.

Esempio

```
<height>4095</height>
```

icon

Adobe AIR 1.0 e versioni successive - opzionale

La proprietà `icon` specifica uno o più file di icona da utilizzare per l'applicazione. L'inclusione di un'icona è opzionale. Se non si specifica una proprietà `icon`, il sistema operativo visualizza un'icona predefinita.

Il percorso specificato è relativo alla directory principale dell'applicazione. I file di icona devono essere in formato PNG. Potete specificare tutte le dimensioni seguenti per le icone:

Se è presente un elemento per una determinata dimensione, l'immagine nel file deve essere esattamente della dimensione specificata. Se non vengono specificate tutte le dimensioni, quelle più simili vengono adattate per un determinato uso dell'icona da parte del sistema operativo.

***Nota:** le icone specificate non vengono aggiunte automaticamente al pacchetto AIR. I file di icona devono essere inclusi nelle posizioni relative corrette nel momento in cui viene creato il pacchetto dell'applicazione.*

Per ottenere un risultato ottimale, specificate un'immagine per ciascuna delle dimensioni disponibili. Inoltre, assicuratevi che le icone siano visualizzabili nelle modalità a colori a 16 e a 32 bit.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: “[imageNxN](#)” a pagina 235

Contenuto

Un elemento `imageNxN` per ogni dimensione di icona desiderata.

Esempio

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

id

Adobe AIR 1.0 e versioni successive - obbligatorio

Una stringa di identificazione per l'applicazione, detta ID applicazione. Spesso viene utilizzato un identificatore in stile DNS inverso, ma non è indispensabile usare questo stile.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Il valore ID è limitato ai seguenti caratteri:

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

Il valore deve contenere da 1 a 212 caratteri. Questo elemento è obbligatorio.

Esempio

```
<id>org.example.application</id>
```

imageNxN

Adobe AIR 1.0 e versioni successive - opzionale

Definisce il percorso di un'icona rispetto alla directory dell'applicazione.

Possono essere utilizzate le seguenti immagini di icone, ciascuna con una dimensione icona differente:

- image16x16
- image29x29 (AIR 2+)
- image32x32
- image36x36 (AIR 2.5+)
- image48x48
- image50x50 (AIR 3.4+)
- image57x57 (AIR 2+)
- image58x58 (AIR 3.4+)
- image72x72 (AIR 2+)
- image100x100 (AIR 3.4+)
- image114x114 (AIR 2.6+)
- image128x128
- image144x144 (AIR 3.4+)
- image512x512 (AIR 2+)
- image1024x1024 (AIR 3.4+)

L'icona deve essere un'immagine PNG esattamente delle dimensioni indicate dall'elemento image. I file di icone devono essere inclusi nel pacchetto dell'applicazione; le icone associate a riferimenti nel documento descrittore dell'applicazione non vengono incluse automaticamente.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Il percorso di file dell'icona può contenere qualsiasi carattere in formato Unicode (UTF-8) a eccezione di quelli seguenti, di cui non è consentito l'uso nei nomi file in vari file system:

Carattere	Codice esadecimale
<i>vari</i>	0x00 – x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E

Carattere	Codice esadecimale
?	x3F
\	x5C
	x7C

Esempio

```
<image32x32>icons/icon32.png</image32x32>
```

InfoAdditions

Adobe AIR 1.0 e versioni successive - opzionale

Permette di specificare proprietà aggiuntive in un'applicazione iOS.

Elemento superiore: [“iPhone”](#) a pagina 239

Elementi secondari: elementi iOS Info.plist

Contenuto

Contiene gli elementi secondari che specificano le coppie chiave-valore da utilizzare come impostazioni Info.plist per l'applicazione. Il contenuto dell'elemento InfoAdditions deve essere racchiuso in un blocco CDATA.

Vedete [Information Property List Key Reference](#) nella Apple iPhone Reference Library per informazioni sulle coppie chiave-valore e come esprimerle in XML.

Esempio

```
<InfoAdditions>
  <![CDATA [
    <key>UIStatusBarStyle</key>
    <string>UIStatusBarStyleBlackOpaque</string>
    <key>UIRequiresPersistentWiFi</key>
    <string>NO</string>
  ]]>
</InfoAdditions>
```

Altri argomenti presenti nell’Aiuto

[“Impostazioni iOS”](#) a pagina 85

initialWindow

Adobe AIR 1.0 e versioni successive - obbligatorio

Definisce il file di contenuto principale e l'aspetto iniziale dell'applicazione.

Elemento superiore: [“application”](#) a pagina 217

Elementi secondari: tutti gli elementi seguenti possono apparire come elementi secondari dell'elemento initialWindow. Tuttavia, alcuni elementi vengono ignorati, a seconda del supporto fornito da AIR per le finestre di una determinata piattaforma:

Elemento	Desktop	Dispositivi mobili
"aspectRatio" a pagina 221	ignorato	utilizzato
"autoOrients" a pagina 221	ignorato	utilizzato
"content" a pagina 223	utilizzato	utilizzato
"depthAndStencil" a pagina 225	utilizzato	utilizzato
"fullScreen" a pagina 233	ignorato	utilizzato
"height" a pagina 233	utilizzato	ignorato
"maximizable" a pagina 241	utilizzato	ignorato
"maxSize" a pagina 241	utilizzato	ignorato
"minimizable" a pagina 242	utilizzato	ignorato
"minSize" a pagina 242	utilizzato	ignorato
"renderMode" a pagina 244	utilizzato (AIR 3.0 e versioni successive)	utilizzato
"requestedDisplayResolution" a pagina 245	utilizzato (AIR 3.6 e versioni successive)	ignorato
"resizable" a pagina 246	utilizzato	ignorato
"softKeyboardBehavior" a pagina 247	ignorato	utilizzato
"systemChrome" a pagina 249	utilizzato	ignorato
"title" a pagina 250	utilizzato	ignorato
"transparent" a pagina 250	utilizzato	ignorato
"visible" a pagina 252	utilizzato	ignorato
"width" a pagina 252	utilizzato	ignorato
"x" a pagina 252	utilizzato	ignorato
"y" a pagina 253	utilizzato	ignorato

Contenuto

Gli elementi secondari che definiscono l'aspetto e il comportamento dell'applicazione.

Esempio

```
<initialWindow>
  <title>Hello World</title>
  <content>
    HelloWorld.swf
  </content>
  <depthAndStencil>true</depthAndStencil>
  <systemChrome>none</systemChrome>
  <transparent>true</transparent>
  <visible>true</visible>
  <maxSize>1024 800</maxSize>
  <minSize>320 240</minSize>
  <maximizable>false</maximizable>
  <minimizable>false</minimizable>
  <resizable>true</resizable>
  <x>20</x>
  <y>20</y>
  <height>600</height>
  <width>800</width>
  <aspectRatio>landscape</aspectRatio>
  <autoOrients>true</autoOrients>
  <fullScreen>false</fullScreen>
  <renderMode>direct</renderMode>
</initialWindow>
```

installFolder

Adobe AIR 1.0 e versioni successive - opzionale

Identifica la sottodirectory della directory di installazione predefinita.

In Windows, la sottodirectory di installazione predefinita è Programmi. In Mac OS, è la directory /Applicazioni. In Linux è /opt/. Se, ad esempio, la proprietà `installFolder` è impostata su "Acme" e un'applicazione è denominata "ExampleApp", l'applicazione viene installata in C:\Programmi\Acme\ExampleApp in Windows, in /Applicazioni/Acme/Example.app in MacOS e in /opt/Acme/ExampleApp in Linux.

La proprietà `installFolder` è opzionale. Se non viene specificata, l'applicazione viene installata in una sottodirectory della directory di installazione predefinita, in base alla proprietà `name`.

Elemento superiore: "application" a pagina 217

Elementi secondari: nessuno

Contenuto

La proprietà `installFolder` può contenere qualsiasi carattere in formato Unicode (UTF-8) a eccezione di quelli non consentiti come nomi di cartella nei vari file system (fate riferimento alla proprietà `filename` per l'elenco delle eccezioni).

Utilizzate il carattere della barra (/) come separatore delle directory per specificare una sottodirectory nidificata.

Esempio

```
<installFolder>utilities/toolA</installFolder>
```

iPhone

Adobe AIR 2.0, solo iOS - opzionale

Definisce proprietà dell'applicazione specifiche per iOS.

Elemento superiore: [“application”](#) a pagina 217

Elementi secondari:

- [“Entitlements”](#) a pagina 227
- [“externalSwfs”](#) a pagina 228
- [“forceCPURenderModeForDevices”](#) a pagina 232
- [“InfoAdditions”](#) a pagina 236
- [“requestedDisplayResolution”](#) a pagina 245

Altri argomenti presenti nell’Aiuto

[“Impostazioni iOS”](#) a pagina 85

manifest

Adobe AIR 2.5 e versioni successive, solo per Android - opzionale

Specifica informazioni da aggiungere al file manifest di Android per l'applicazione.

Elemento superiore: [“manifestAdditions”](#) a pagina 240

Elementi secondari: definiti da Android SDK.

Contenuto

L'elemento manifest, tecnicamente parlando, non fa parte dello schema del descrittore dell'applicazione AIR. È l'elemento radice del documento XML manifest di Android. Qualsiasi contenuto che inserite nell'elemento manifest deve essere conforme allo schema AndroidManifest.xml. Quando generate un file APK con i tool AIR, le informazioni dell'elemento manifest vengono copiate nella parte corrispondente del file AndroidManifest.xml generato per l'applicazione.

Se specificate valori manifest Android che sono disponibili solo in una versione SDK più recente di quelle direttamente supportate da AIR, dovete impostare il flag `-platformSDK` su ADT quando compilate l'app. Impostate il flag del percorso del file system su una versione di Android SDK che supporti i valori che state aggiungendo.

L'elemento manifest deve essere racchiuso in un blocco CDATA all'interno del descrittore dell'applicazione AIR.

Esempio

```
<![CDATA [  
  <manifest android:sharedUserID="1001">  
    <uses-permission android:name="android.permission.CAMERA"/>  
    <uses-feature android:required="false" android:name="android.hardware.camera"/>  
    <application android:allowClearUserData="true"  
      android:enabled="true"  
      android:persistent="true"/>  
  </manifest>  
]]>
```

Altri argomenti presenti nell’Aiuto

“[Impostazioni Android](#)” a pagina 79

[Il file AndroidManifest.xml](#)

manifestAdditions

Adobe AIR 2.5 e versioni successive, solo per Android

Specifica informazioni da aggiungere al file manifest di Android.

Ogni applicazione Android include un file manifest che definisce le proprietà di base dell'applicazione stessa. Il manifest Android è concettualmente simile al descrittore dell'applicazione AIR. Un'applicazione AIR for Android ha sia un descrittore dell'applicazione che un file manifest Android generato automaticamente. Quando viene creato il pacchetto di un'applicazione AIR for Android, le informazioni presenti in questo elemento `manifestAdditions` vengono aggiunte alle parti corrispondenti del documento manifest Android.

Elemento superiore: “[android](#)” a pagina 217

Elementi secondari: “[manifest](#)” a pagina 239

Contenuto

Le informazioni dell'elemento `manifestAdditions` vengono aggiunte al documento XML `AndroidManifest`.

AIR imposta varie voci del file manifest nel documento `Android manifest` generato, al fine di garantire che le funzioni dell'applicazione e del runtime funzionino correttamente. Non è possibile eseguire l'override delle seguenti impostazioni:

Non è possibile impostare gli attributi seguenti dell'elemento `manifest`:

- `package`
- `android:versionCode`
- `android:versionName`

Non è possibile impostare gli attributi seguenti dell'elemento `activity` principale:

- `android:label`
- `android:icon`

Non è possibile impostare gli attributi seguenti dell'elemento `application`:

- `android:theme`
- `android:name`
- `android:label`
- `android:windowSoftInputMode`
- `android:configChanges`
- `android:screenOrientation`
- `android:launchMode`

Esempio

```
<manifestAdditions>
  <![CDATA [
    <manifest android:installLocation="preferExternal">
      <uses-permission android:name="android.permission.INTERNET"/>
      <application android:allowClearUserData="true"
        android:enabled="true"
        android:persistent="true"/>
    </manifest>
  ]]>
</manifestAdditions>
```

Altri argomenti presenti nell’Aiuto

“Impostazioni Android” a pagina 79

[Il file AndroidManifest.xml](#)

maximizable

Adobe AIR 1.0 e versioni successive - opzionale

Specifica se la finestra può essere ingrandita.

Nota: nei sistemi operativi (ad esempio Mac OS X) in cui l’ingrandimento delle finestre è un’operazione di ridimensionamento, i valori `maximizable` e `resizable` devono essere impostati su `false` per impedire che la finestra venga ingrandita o ridimensionata.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

`true` (predefinito) o `false`

Esempio

```
<maximizable>false</maximizable>
```

maxSize

Adobe AIR 1.0 e versioni successive - opzionale

le dimensioni massime di una finestra. Se non impostate una dimensione massima, questa viene determinata dal sistema operativo.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Due numeri interi che rappresentano la larghezza e l’altezza massime, separati da spazi vuoti.

Nota: le dimensioni massime di una finestra supportate da AIR sono aumentate da 2048x2048 pixel a 4096x4096 pixel in AIR 2. (Poiché le coordinate dello schermo sono a base zero, il valore massimo utilizzabile per la larghezza o l’altezza è 4095.)

Esempio

```
<maxSize>1024 360</maxSize>
```

minimizable

Adobe AIR 1.0 e versioni successive - opzionale

Specifica se la finestra può essere ridotta a icona.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

true (predefinito) o false

Esempio

```
<minimizable>>false</minimizable>
```

minSize

Adobe AIR 1.0 e versioni successive - opzionale

Specifica la dimensione minima consentita per la finestra.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Due numeri interi che rappresentano la larghezza e l'altezza minime, separati da spazi vuoti. Tenete presente che la dimensione minima imposta dal sistema operativo ha la precedenza sul valore impostate nel descrittore dell'applicazione.

Esempio

```
<minSize>120 60</minSize>
```

name

Adobe AIR 1.0 e versioni successive - opzionale

Il titolo dell'applicazione visualizzato nel programma di installazione dell'applicazione AIR.

Se non è specificato alcun elemento name, il programma di installazione dell'applicazione AIR visualizza il valore filename come nome dell'applicazione.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: “[text](#)” a pagina 249

Contenuto

Se specificate un unico nodo di testo (invece che più elementi <text>), il programma di installazione dell'applicazione AIR utilizza questo nome, a prescindere dal linguaggio del sistema.

Lo schema del descrittore dell'applicazione AIR 1.0 consente di definire un unico nodo di testo semplice per il nome (non più elementi `text`). In AIR 1.1 (o versione successiva) potete invece specificare più lingue nell'elemento `name`.

L'attributo `xml:lang` di ogni elemento di testo specifica un codice lingua, definito dallo standard [RFC4646](http://www.ietf.org/rfc/rfc4646.txt) (<http://www.ietf.org/rfc/rfc4646.txt>).

Il programma di installazione dell'applicazione AIR utilizza il nome che più corrisponde alla lingua dell'interfaccia utente del sistema operativo dell'utente. Ad esempio, considerate un'installazione in cui l'elemento `name` del file descrittore dell'applicazione comprende un valore per la versione locale `en` (inglese). Il programma di installazione dell'applicazione AIR utilizza il nome `en` se il sistema operativo identifica `en` (inglese) come lingua dell'interfaccia utente. Utilizza anche il nome `en` se la lingua dell'interfaccia utente del sistema è `en-US` (inglese americano). Tuttavia, se la lingua dell'interfaccia utente è `en-US` e il file descrittore dell'applicazione definisce i nomi `en-US` ed `en-GB`, quindi il programma di installazione dell'applicazione AIR utilizza il valore `en-US`. Se l'applicazione non definisce alcun nome che corrisponda alle lingue dell'interfaccia utente del sistema, programma di installazione dell'applicazione AIR utilizza il primo valore `name` definito nel file descrittore dell'applicazione.

L'elemento `name` specifica solo il titolo dell'applicazione utilizzato nel programma di installazione dell'applicazione AIR. Il programma di installazione dell'applicazione AIR supporta più lingue: cinese tradizionale, cinese semplificato, ceco, olandese, inglese, francese, tedesco, italiano, giapponese, coreano, polacco, portoghese brasiliano, russo, spagnolo, svedese e turco. Il programma di installazione dell'applicazione AIR seleziona la relativa lingua visualizzata (per il testo diverso dal titolo e dalla descrizione dell'applicazione) in base alla lingua dell'interfaccia utente del sistema. Questa selezione di lingua non dipende dalle impostazioni nel file descrittore dell'applicazione.

L'elemento `name` *non* definisce le versioni locali disponibili per l'esecuzione dell'applicazione installata. Per informazioni dettagliate sullo sviluppo di applicazioni multilingue, vedete “[Localizzazione di applicazioni AIR](#)” a pagina 304.

Esempio

L'esempio seguente definisce un nome con un nodo di testo semplice:

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

L'esempio seguente, valido in AIR 1.1 e versioni successive, specifica il nome in tre lingue (inglese, francese e spagnolo) utilizzando nodi di elemento `text`:

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

name

Adobe AIR 1.0 e versioni successive - obbligatorio

Identifica il nome di un tipo di file.

Elemento superiore: “[fileType](#)” a pagina 230

Elementi secondari: nessuno

Contenuto

Una stringa che rappresenta il nome del tipo di file.

Esempio

```
<name>adobe.VideoFile</name>
```

programMenuFolder

Adobe AIR 1.0 e versioni successive - opzionale

Identifica il percorso in cui posizionare i collegamenti all'applicazione nel menu Tutti i programmi del sistema operativo Windows o nel menu Applications di Linux. (Questa impostazione è attualmente ignorata in altri sistemi operativi.)

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

La stringa usata per il valore `programMenuFolder` può contenere qualsiasi carattere in formato Unicode (UTF-8) a eccezione di quelli non consentiti come nomi di cartella nei vari file system (vedete l'elemento `filename` per l'elenco delle eccezioni). *Non* utilizzate il carattere barra (/) come ultimo carattere di questo valore.

Esempio

```
<programMenuFolder>Example Company/Sample Application</programMenuFolder>
```

publisherID

Adobe AIR 1.5.3 e versioni successive - opzionale

Identifica l'ID editore per l'aggiornamento di un'applicazione AIR originariamente creata con AIR versione 1.5.2 o precedente.

Specificate un ID editore solo quando create l'aggiornamento di un'applicazione. Il valore dell'elemento `publisherID` deve corrispondere all'ID editore generato da AIR per la versione precedente dell'applicazione. Per un'applicazione installata, l'ID editore si trova nella cartella di installazione dell'applicazione, nel file `META-INF/AIR/publisherid`.

Per le nuove applicazioni create con AIR 1.5.3 e versioni successive l'ID editore non deve essere specificato.

Per ulteriori informazioni, vedete “[Informazioni sugli ID editore AIR](#)” a pagina 197.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Una stringa di ID editore.

Esempio

```
<publisherID>B146A943FBD637B68C334022D304CEA226D129B4.1</publisherID>
```

renderMode

Adobe AIR 2.0 e versioni successive - opzionale

Specifica se deve essere utilizzata l'accelerazione GPU, se supportata dal dispositivo corrente.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Uno dei seguenti valori:

- `auto` (predefinito) - attualmente corrisponde alla modalità CPU.
- `cpu` - accelerazione hardware non utilizzata.
- `direct` - la composizione del rendering avviene nella CPU; il blitting utilizza la GPU. Disponibile in AIR 3+.

***Nota:** per poter sfruttare l'accelerazione GPU dei contenuti Flash con AIR per le piattaforme mobili, Adobe consiglia di utilizzare `renderMode="direct"` (ovvero, Stage3D) anziché `renderMode="gpu"`. Adobe supporta e consiglia ufficialmente i seguenti framework basati su Stage3D: Starling (2D) e Away3D (3D). Per maggiori dettagli su Stage3D e Starling/Away3D, visitate <http://gaming.adobe.com/getstarted/>.*

- `gpu` - accelerazione hardware utilizzata, se disponibile.

***Importante:** Non utilizzate la modalità di rendering tramite GPU per le applicazioni Flex.*

Esempio

```
<renderMode>direct</renderMode>
```

requestedDisplayResolution

Adobe AIR 2.6 e versioni successive, solo iOS; Adobe AIR 3.6 e versioni successive, OS X - Opzionale

Specifica se l'applicazione richiede la risoluzione standard o elevata in un dispositivo o in un monitor per computer con schermo ad alta risoluzione. Se impostato su *standard* (valore predefinito), lo schermo apparirà all'applicazione come schermo a risoluzione standard. Se la risoluzione impostata è *elevata*, l'applicazione può utilizzare ogni pixel in alta risoluzione.

Ad esempio, su uno schermo iPhone ad alta risoluzione da 640x960, se l'impostazione è *standard* le dimensioni a schermo intero dello stage sono 320x480 e ciascun pixel dell'applicazione è renderizzato utilizzando quattro pixel dello schermo. Se la risoluzione è impostata su *elevata*, le dimensioni a schermo intero dello stage corrispondono a 640x960.

Nei dispositivi con schermi a risoluzione standard, le dimensioni dello stage corrispondono alle dimensioni dello schermo, indipendentemente dall'impostazione utilizzata.

Se l'elemento `requestedDisplayResolution` è nidificato nell'elemento `iPhone`, viene applicato ai dispositivi iOS. In questo caso, l'attributo `excludeDevices` può essere utilizzato per specificare i dispositivi per i quali non è applicata l'impostazione.

Se l'elemento `requestedDisplayResolution` è nidificato nell'elemento `initialWindow`, viene applicato alle applicazioni AIR desktop su computer MacBook Pro che supportano display ad alta risoluzione. Il valore specificato viene applicato a tutte le finestre native utilizzate nell'applicazione. La nidificazione dell'elemento `requestedDisplayResolution` nell'elemento `initialWindow` è supportata in AIR 3.6 e versioni successive.

Elemento superiore: “[iPhone](#)” a pagina 239, “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

standard (valore predefinito) oppure *high*.

Attributo:

`excludeDevices` - un elenco separato da spazi di prefissi di nomi di modelli o nomi di modelli iOS. Consente allo sviluppatore di utilizzare programmi ad alta risoluzione e altri a risoluzione standard. Questo attributo è disponibile solo su dispositivi iOS (l'elemento `requestedDisplayResolution` è nidificato nell'elemento `iPhone`). L'attributo `excludeDevices` è disponibile in AIR 3.6 e versioni successive.

Per ciascun dispositivo il cui nome del modello è specificato in questo attributo, il valore `requestedDisplayResolution` è l'opposto del valore specificato. In altre parole, se il valore `requestedDisplayResolution` è *elevato*, i dispositivi esclusi utilizzano la risoluzione standard. Se il valore `requestedDisplayResolution` è *standard*, i dispositivi esclusi utilizzano la risoluzione elevata.

I valori corrispondono a prefissi di nomi di modelli o nomi di modelli di dispositivi iOS. Ad esempio, il valore `iPad3,1` si riferisce nello specifico a un iPad di terza generazione Wi-Fi (ma non a iPad di terza generazione CDMA o GSM). In alternativa, il valore `iPad3` si riferisce a iPad di terza generazione. Un elenco non ufficiale di nomi modelli iOS è disponibile nella [pagina dei modelli iPhone wiki](#).

Esempi

Desktop:

```
<initialWindow>  
  <requestedDisplayResolution>high</requestedDisplayResolution>  
</initialWindow>
```

iOS:

```
<iPhone>  
  <requestedDisplayResolution excludeDevices="iPad3  
iPad4">high</requestedDisplayResolution>  
</iPhone>
```

resizable

Adobe AIR 1.0 e versioni successive - opzionale

Specifica se la finestra può essere ridimensionata.

Nota: nei sistemi operativi (ad esempio Mac OS X) in cui l'ingrandimento delle finestre è un'operazione di ridimensionamento, i valori `maximizable` e `resizable` devono essere impostati su `false` per impedire che la finestra venga ingrandita o ridimensionata.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari:

Contenuto

true (predefinito) o false

Esempio

```
<resizable>false</resizable>
```

softKeyboardBehavior

Adobe AIR 2.6 e versioni successive, profilo mobile - opzionale

Specifica il comportamento predefinito dell'applicazione quando è visualizzata una tastiera virtuale. Il comportamento predefinito è lo scorrimento verso l'alto dell'applicazione. Il runtime mantiene visibile sullo schermo il campo di testo o oggetto interattivo attualmente attivato. Usate l'opzione *pan* se l'applicazione non fornisce una propria logica per la gestione della tastiera.

Potete anche disattivare il comportamento automatico impostando l'elemento `softKeyboardBehavior` su *none*. In questo caso, i campi di testo e gli oggetti interattivi inviano un evento `SoftKeyboardEvent` quando viene visualizzata la tastiera virtuale, ma il runtime non esegue lo scorrimento o il ridimensionamento dell'applicazione. È quindi compito dell'applicazione mantenere visibile l'area di immissione del testo.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

none o *pan*. Il valore predefinito è *pan*.

Esempio

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

Altri argomenti presenti nell’Aiuto

[SoftKeyboardEvent](#)

supportedLanguages

Adobe AIR 3.2 e versioni successive - opzionale

Identifica le lingue supportate dall'applicazione. Questo elemento è utilizzato solo da iOS, dal runtime autonomo Mac e dalle applicazioni Android. Viene ignorato da tutti gli altri tipi di applicazioni.

Se non specificate questo elemento, l'impostazione predefinita prevede che il compilatore esegua le operazioni seguenti in base al tipo di applicazione:

- iOS - Tutte le lingue supportate dal runtime AIR sono elencate in iOS App Store come lingue supportate dell'applicazione.
- Runtime autonomo Mac - L'applicazione impacchettata con il runtime autonomo non contiene informazioni sulla localizzazione.
- Android - Il pacchetto dell'applicazione contiene le risorse per tutte le lingue supportate dal runtime AIR.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Un elenco delimitato da spazi delle lingue supportate. I valori validi corrispondono ai valori ISO 639-1 delle lingue supportate dal runtime AIR: en, de, es, fr, it, ja, ko, pt, ru, cs, nl, pl, sv, tr, zh, da, nb, iw.

Il compilatore genera un errore in caso di valore vuoto per l'elemento `<supportedLanguages>`.

Nota: i tag localizzati (come il tag del nome) ignorano il valore di una lingua se utilizzate il tag `<supportedLanguages>` e questo non contiene quella lingua. Se un'estensione nativa contiene le risorse per una lingua che non è specificata dal tag `<supportedLanguages>`, viene generato un avviso e le risorse di tale lingua vengono ignorate.

Esempio

```
<supportedLanguages>en ja fr es</supportedLanguages>
```

supportedProfiles

Adobe AIR 2.0 e versioni successive - opzionale

Identifica i profili supportati per l'applicazione.

Elemento superiore: [“application”](#) a pagina 217

Elementi secondari: nessuno

Contenuto

Potete includere uno dei valori seguenti nell'elemento `supportedProfiles`:

- `desktop` Il profilo `desktop` è per le applicazioni AIR installate in un computer desktop mediante un file AIR. Queste applicazioni non hanno accesso alla classe `NativeProcess` (che garantisce la comunicazione con le applicazioni native).
- `extendedDesktop` Il profilo di `desktop` esteso è per le applicazioni AIR installate in un computer desktop mediante un programma di installazione dell'applicazione nativo. Queste applicazioni hanno accesso alla classe `NativeProcess` (che garantisce la comunicazione con le applicazioni native).
- `mobileDevice` - Questo profilo è per le applicazioni destinate ai dispositivi mobili.
- `extendedMobileDevice` - Il profilo dispositivo mobile esteso non è attualmente in uso.

La proprietà `supportedProfiles` è facoltativa. Se non includete questo elemento nel file descrittore dell'applicazione, quest'ultima può essere compilata e distribuita per tutti i profili.

Per specificare più profili, separarli con un carattere di spaziatura. Con l'impostazione seguente ad esempio si specifica che l'applicazione è disponibile solo nei profili per `desktop` ed `estesi`:

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

Nota: quando eseguite un'applicazione con ADL e non specificate un valore per l'opzione `ADL -profile`, viene utilizzato il primo profilo elencato nel descrittore dell'applicazione. (Se nel descrittore non è specificato alcun profilo, viene utilizzato il profilo `desktop`.)

Esempio

```
<supportedProfiles>desktop mobileDevice</supportedProfiles>
```

Altri argomenti presenti nell' Aiuto

[“Profili dispositivo”](#) a pagina 254

[“Profili supportati”](#) a pagina 78

systemChrome

Adobe AIR 1.0 e versioni successive - opzionale

Specifica se la finestra iniziale dell'applicazione viene creata con gli elementi standard (barra del titolo, bordi e controlli) forniti dal sistema operativo.

L'impostazione del chrome di sistema della finestra non può essere modificata in fase runtime.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Uno dei seguenti valori:

- `none` - Chrome di sistema non fornito. Spetta all'applicazione (o un framework di applicazione come Flex) visualizzare il chrome della finestra.
- `standard` (predefinito) - Il chrome di sistema viene fornito dal sistema operativo.

Esempio

```
<systemChrome>standard</systemChrome>
```

text

Adobe AIR 1.1 e versioni successive - opzionale

Specifica una stringa localizzata.

L'attributo `xml:lang` di un elemento di testo specifica un codice lingua, definito dallo standard [RFC4646](http://www.ietf.org/rfc/rfc4646.txt) (<http://www.ietf.org/rfc/rfc4646.txt>).

Il programma di installazione dell'applicazione AIR utilizza l'elemento `text` con il valore dell'attributo `xml:lang` più simile alla lingua dell'interfaccia utente del sistema operativo dell'utente.

Ad esempio, considerate un'installazione in cui un elemento `text` comprende un valore per la versione locale en (inglese). Il programma di installazione dell'applicazione AIR utilizza il nome `en` se il sistema operativo identifica `en` (inglese) come lingua dell'interfaccia utente. Utilizza anche il nome `en` se la lingua dell'interfaccia utente del sistema è `en-US` (inglese americano). Tuttavia, se la lingua dell'interfaccia utente è `en-US` e il file descrittore dell'applicazione definisce i nomi `en-US` ed `en-GB`, quindi il programma di installazione dell'applicazione AIR utilizza il valore `en-US`.

Se l'applicazione non definisce un elemento `text` che corrisponda alle lingue dell'interfaccia utente del sistema, il programma di installazione dell'applicazione AIR utilizza il primo valore `name` definito nel file descrittore dell'applicazione.

Elementi superiori:

- “[name](#)” a pagina 242
- “[description](#)” a pagina 225

Elementi secondari: nessuno

Contenuto

Un attributo `xml:lang` che specifica una versione locale e una stringa di testo localizzato.

Esempio

```
<text xml:lang="fr">Bonjour AIR</text>
```

title

Adobe AIR 1.0 e versioni successive - opzionale

Specifica il titolo visualizzato nella barra del titolo della finestra iniziale dell'applicazione.

Il titolo viene visualizzato solo se l'elemento `systemChrome` è impostato su `standard`.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Una stringa contenente il titolo della finestra.

Esempio

```
<title>Example Window Title</title>
```

transparent

Adobe AIR 1.0 e versioni successive - opzionale

Specifica se la finestra iniziale dell'applicazione esegue una fusione alfa con il desktop.

Una finestra con trasparenza abilitata potrebbe essere visualizzata più lentamente e richiedere più memoria. L'impostazione della trasparenza non può essere modificata in fase runtime.

Importante: potete impostare `transparent` su `true` solo quando `systemChrome` è `none`.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

`true` o `false` (predefinito)

Esempio

```
<transparent>true</transparent>
```

version

Adobe AIR da 1.0 a 2.0 - obbligatorio; non consentito in AIR 2.5 e versioni successive

Specifica le informazioni della versione per l'applicazione.

La stringa di versione è un indicatore definito dall'applicazione. AIR non è in grado di interpretare in alcun modo la stringa relativa alla versione. Pertanto, la versione “3.0” non viene considerata più aggiornata della versione “2.0”.

Esempi: “1.0”, “.4”, “0.5”, “4.9”, “1.3.4a”.

In AIR 2.5 e versioni successive, l'elemento `version` è stato sostituito dagli elementi `versionNumber` e `versionLabel`.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Una stringa contenente la versione dell'applicazione.

Esempio

```
<version>0.1 Alpha</version>
```

versionLabel

Adobe AIR 2.5 e versioni successive - opzionale

Specifica una stringa di versione in formato leggibile.

Il valore dell'etichetta di versione viene visualizzato nelle finestre di installazione al posto del valore dell'elemento `versionNumber`. Se `versionLabel` non viene utilizzato, `versionNumber` viene usato per entrambi.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Una stringa contenente il testo della versione pubblico.

Esempio

```
<versionLabel>0.9 Beta</versionLabel>
```

versionNumber

Adobe AIR 2.5 e versioni successive - obbligatorio

Il numero di versione dell'applicazione.

Elemento superiore: “[application](#)” a pagina 217

Elementi secondari: nessuno

Contenuto

Il numero di versione può contenere una sequenza composta da un massimo di tre numeri interi separati da punti. Ogni numero intero deve essere compreso tra 0 e 999 (inclusi).

Esempi

```
<versionNumber>1.0.657</versionNumber>
```

```
<versionNumber>10</versionNumber>
```

```
<versionNumber>0.01</versionNumber>
```


visible

Adobe AIR 1.0 e versioni successive - opzionale

Indica se la finestra iniziale dell'applicazione è visibile non appena viene creata.

Le finestre di AIR, compresa quella iniziale, vengono create come invisibili per impostazione predefinita. Per visualizzare una finestra, chiamate il metodo `activate()` dell'oggetto `NativeWindow` o impostate la proprietà `visible` su `true`. Potete scegliere di mantenere la finestra principale nascosta inizialmente, in modo che non siano visibili i cambiamenti relativi alla posizione e alle dimensioni della finestra e al layout del relativo contenuto.

Il componente `mx:WindowedApplication` di Flex visualizza e attiva automaticamente la finestra subito prima dell'invio dell'evento `applicationComplete`, a meno che l'attributo `visible` non sia impostato su `false` nella definizione MXML.

Nei dispositivi del profilo mobile, che non supporta le finestre, l'impostazione `visible` viene ignorata.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

`true` o `false` (predefinito)

Esempio

```
<visible>true</visible>
```

width

Adobe AIR 1.0 e versioni successive - opzionale

La larghezza iniziale della finestra principale dell'applicazione.

Se non impostate la larghezza, questa viene determinata dalle impostazioni del file SWF principale oppure, nel caso di un'applicazione AIR basata su HTML, dal sistema operativo.

La larghezza massima di una finestra è cambiata da 2048 pixel a 4096 pixel in AIR 2.

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Un numero intero positivo con un valore massimo di 4095.

Esempio

```
<width>1024</width>
```

X

Adobe AIR 1.0 e versioni successive - opzionale

La posizione orizzontale della finestra iniziale dell'applicazione.

Nella maggior parte dei casi è meglio lasciare che sia il sistema operativo a determinare la posizione iniziale della finestra anziché assegnare un valore fisso.

L'origine del sistema di coordinate dello schermo (0,0) è l'angolo superiore sinistro della schermata principale del desktop (determinata dal sistema operativo).

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Un numero intero.

Esempio

```
<x>120</x>
```

y

Adobe AIR 1.0 e versioni successive - opzionale

La posizione verticale della finestra iniziale dell'applicazione.

Nella maggior parte dei casi è meglio lasciare che sia il sistema operativo a determinare la posizione iniziale della finestra anziché assegnare un valore fisso.

L'origine del sistema di coordinate dello schermo (0,0) è l'angolo superiore sinistro della schermata principale del desktop (determinata dal sistema operativo).

Elemento superiore: “[initialWindow](#)” a pagina 236

Elementi secondari: nessuno

Contenuto

Un numero intero.

Esempio

```
<y>250</y>
```

Capitolo 15: Profili dispositivo

Adobe AIR 2 e versioni successive

I profili sono un meccanismo per definire le classi dei dispositivi informatici sui quali viene eseguita l'applicazione. Un profilo definisce una serie di API e di funzionalità tipicamente supportate in una particolare classe di dispositivo. I profili disponibili sono:

- desktop
- extendedDesktop
- mobileDevice
- extendedMobileDevice

Potete definire i profili per l'applicazione specificandoli nel descrittore dell'applicazione. Gli utenti di computer e dispositivi corrispondenti ai profili specificati possono installare l'applicazione, gli utenti di altri computer e dispositivi no. Ad esempio, se includete solo il profilo desktop nel descrittore dell'applicazione, gli utenti possono installare ed eseguire l'applicazione solo sui computer desktop.

Se includete un profilo che l'applicazione non supporta veramente, l'esperienza dell'utente in tali ambiente potrebbe risentirne. Se non specificate nessun profilo nel descrittore dell'applicazione, AIR non pone alcun limite all'applicazione. Potete creare il pacchetto dell'applicazione in uno qualsiasi dei formati supportati e gli utenti potranno installarla sui dispositivi di qualunque profilo, tuttavia potrebbe non funzionare in fase runtime.

Quando è possibile, le restrizioni relative ai profili vengono applicate al momento della creazione del pacchetto dell'applicazione. Ad esempio, se includete solo il profilo extendedDesktop, non potete creare il pacchetto dell'applicazione in formato AIR, bensì solo come programma di installazione nativo. Analogamente, se non includete il profilo mobileDevice, non potete impacchettare l'applicazione nel formato APK di Android.

Un singolo dispositivo informatico può supportare più di un profilo. Ad esempio, AIR su computer desktop supporta le applicazioni scritte per i profili desktop e extendedDesktop. Tuttavia, un'applicazione associata al profilo extendedDesktop può comunicare con processi nativi e DEVE essere impacchettata come programma di installazione nativo (exe, dmg, deb o rpm). Un'applicazione del profilo desktop, al contrario, non ha la possibilità di comunicare con un processo nativo e può essere impacchettata sia come file AIR che come programma di installazione nativo.

L'inclusione di una determinata funzione in un profilo indica che il supporto di tale funzione è comune nella classe di dispositivi per la quale tale profilo è definito. Tuttavia, non significa che ogni dispositivo del profilo supporta ogni funzione. Ad esempio, alcuni telefoni cellulari, ma non tutti, contengono un accelerometro. Le classi e le funzioni che non dispongono di supporto universale solitamente hanno una proprietà booleana che può essere verificata prima di utilizzare la funzione. Nel caso dell'accelerometro, ad esempio, è possibile testare la proprietà statica `Accelerometer.isSupported` per determinare se il dispositivo corrente è dotato di un accelerometro supportato.

I profili seguenti possono essere assegnati all'applicazione AIR utilizzando l'elemento `supportedProfiles` nel descrittore dell'applicazione:

Desktop Il profilo desktop definisce un set di funzionalità per le applicazioni AIR che vengono installate come file AIR su un computer desktop. Queste applicazioni vengono installate ed eseguite sulle piattaforme desktop supportate (Mac OS, Windows e Linux). Le applicazioni AIR sviluppate nelle versioni di AIR precedenti ad AIR 2 possono essere considerate applicazioni con profilo desktop. Alcune API non funzionano in questo profilo. Ad esempio, le applicazioni desktop non possono comunicare con i processi nativi.

Desktop esteso Il profilo desktop esteso definisce un set di funzionalità per le applicazioni AIR che vengono impacchettate e installate sotto forma di programma di installazione nativo. I programmi di installazione nativi sono file EXE in Windows, DMG in Mac OS e BIN, DEB o RPM in Linux. Le applicazioni con profilo desktop esteso sono dotate di funzionalità aggiuntive non disponibili per le applicazioni con profilo desktop. Per ulteriori informazioni, vedete “[Creazione del pacchetto di un file di installazione nativo desktop](#)” a pagina 59.

Dispositivo mobile Il profilo dispositivo mobile definisce un set di funzionalità per le applicazioni che vengono installate sui dispositivi mobili, ad esempio telefoni cellulari e tablet. Queste applicazioni vengono installate ed eseguite sulle piattaforme mobili supportate (Android, Blackberry Tablet OS e iOS).

Dispositivo mobile esteso Il profilo dispositivo mobile esteso definisce un set di funzionalità esteso per le applicazioni che vengono installate sui dispositivi mobili. Attualmente, nessun dispositivo supporta questo profilo.

Restrizione dei profili target nel file descrittore dell'applicazione

Adobe AIR 2 e versioni successive

A partire da AIR 2, il file descrittore dell'applicazione comprende un elemento `supportedProfiles` che consente di assegnare una restrizione ai profili target. L'impostazione seguente, ad esempio, specifica che l'applicazione è disponibile solo nel profilo desktop:

```
<supportedProfiles>desktop</supportedProfiles>
```

Se questo elemento è impostato, l'applicazione può essere associata solo ai profili elencati. Utilizzate i seguenti valori:

- `desktop` - Profilo desktop
- `extendedDesktop` - Profilo desktop esteso
- `mobileDevice` - Profilo dispositivo mobile

L'elemento `supportedProfiles` è opzionale. Se non includete questo elemento nel file descrittore dell'applicazione, quest'ultima può essere compilata e distribuita con tutti i profili.

Per specificare più profili nell'elemento `supportedProfiles`, separateli con uno spazio come nell'esempio seguente:

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

Funzionalità supportate dai profili

Adobe AIR 2 e versioni successive

La tabella seguente elenca le classi e le funzioni che non sono supportate in tutti i profili.

Classe o funzione	desktop	extendedDesktop	mobileDevice
Accelerometer (Accelerometer.isSupported)	No	No	Controllare
Accessibility (Capabilities.hasAccessibility)	Si	Si	No
Soppressione dell'eco acustica (Microphone.getEnhancedMicrophone())	Si	Si	No

Classe o funzione	desktop	extendedDesktop	mobileDevice
ActionScript 2	Si	Si	No
Matrice CacheAsBitmap	No	No	Si
Camera (Camera.isSupported)	Si	Si	Si
CameraRoll	No	No	Si
CameraUI (CameraUI.isSupported)	No	No	Si
Pacchetti runtime autonomi	Si	Si	Si
ContextMenu (ContextMenu.isSupported)	Si	Si	No
DatagramSocket (DatagramSocket.isSupported)	Si	Si	Si
DockIcon (NativeApplication.supportsDockIcon)	Controllare	Controllare	No
Drag-and-drop (NativeDragManager.isSupported)	Si	Si	Controllare
EncryptedLocalStore (EncryptedLocalStore.isSupported)	Si	Si	Si
Flash Access (DRMManager.isSupported)	Si	Si	No
GameInput (GameInput.isSupported)	No	No	No
Geolocation (Geolocation.isSupported)	No	No	Controllare
HTMLLoader (HTMLLoader.isSupported)	Si	Si	No
IME (IME.isSupported)	Si	Si	Controllare
LocalConnection (LocalConnection.isSupported)	Si	Si	No
Microphone (Microphone.isSupported)	Si	Si	Controllare
Audio multicanale (Capabilities.hasMultiChannelAudio())	No	No	No
Estensioni native	No	Si	Si
NativeMenu (NativeMenu.isSupported)	Si	Si	No
NativeProcess (NativeProcess.isSupported)	No	Si	No
NativeWindow (NativeWindow.isSupported)	Si	Si	No
NetworkInfo (NetworkInfo.isSupported)	Si	Si	Controllare
Apertura di file con l'applicazione predefinita	Limitato	Si	No
PrintJob (PrintJob.isSupported)	Si	Si	No
SecureSocket (SecureSocket.isSupported)	Si	Si	Controllare
ServerSocket (ServerSocket.isSupported)	Si	Si	Si
Shader	Si	Si	Limitato
Stage3D (Stage.stage3Ds.length)	Si	Si	Si

Classe o funzione	desktop	extendedDesktop	mobileDevice
Orientamento dello stage (Stage.supportsOrientationChange)	No	No	Sì
StageVideo	No	No	Controllare
StageWebView (StageWebView.isSupported)	Sì	Sì	Sì
Avvio dell'applicazione dopo il login (NativeApplication.supportsStartAtLogin)	Sì	Sì	No
StorageVolumeInfo (StorageVolumeInfo.isSupported)	Sì	Sì	No
Modalità idle del sistema	No	No	Sì
SystemTrayIcon (NativeApplication.supportsSystemTrayIcon)	Controllare	Controllare	No
Input TLF (Text Layout Framework)	Sì	Sì	No
Updater (Updater.isSupported)	Sì	No	No
XMLSignatureValidator (XMLSignatureValidator.isSupported)	Sì	Sì	No

Le voci della tabella hanno il seguente significato:

- *Controllare* - La funzione è supportata su alcuni dispositivi del profilo, ma non su tutti. Occorre controllare in fase runtime se la funzione è supportata, prima di utilizzarla.
- *Limitato* - La funzione è supportata, ma con dei limiti importanti. Per ulteriori informazioni, vedete la relativa documentazione.
- *No* - La funzione non è supportata nel profilo.
- *Sì* - La funzione supportata nel profilo. Tenete presente che singoli dispositivi informatici potrebbero non essere dotati di tutto l'hardware necessario per l'uso di una determinata funzione. Ad esempio, non tutti i telefoni hanno una fotocamera.

Indicazione dei profili per il debug con ADL

Adobe AIR 2 e versioni successive

ADL verifica se avete specificato profili supportati nell'elemento `supportedProfiles` del file descrittore dell'applicazione. In caso affermativo, ADL utilizza il primo profilo supportato presente nell'elenco in fase di debug.

Potete specificare un profilo per la sessione di debug di ADL utilizzando l'argomento della riga di comando `-profile` (Vedete “[ADL \(AIR Debug Launcher\)](#)” a pagina 165.) Questo argomento può essere utilizzato indipendentemente dal fatto che abbiate specificato un profilo nell'elemento `supportedProfiles` del file descrittore dell'applicazione. Se specificate un elemento `supportedProfiles`, tuttavia, tale elemento deve includere il profilo che specificate nella riga di comando. In caso contrario, ADL genera un errore.

Capitolo 16: API interna al browser AIR.SWF

Personalizzazione del file badge.swf per l'installazione invisibile

Oltre a utilizzare il file badge.swf fornito con SDK, è possibile creare un file SWF personalizzato da utilizzare in una pagina browser. Tale file personalizzato può interagire con il runtime nei modi seguenti:

- Installando un'applicazione AIR. Vedete “[Installazione di un'applicazione AIR dal browser](#)” a pagina 264.
- Verificando se una determinata applicazione AIR è installata. Vedete “[Verificare da una pagina Web se un'applicazione AIR è installata](#)” a pagina 263.
- Verificando se il runtime è installato. Vedete “[Verificare se il runtime è installato](#)” a pagina 262.
- Avviando un'applicazione AIR installata sul sistema dell'utente. Vedete “[Avvio dal browser di un'applicazione AIR installata](#)” a pagina 265.

Queste funzionalità vengono tutte fornite chiamando le API in un file SWF ospitato su adobe.com: air.swf. Potete personalizzare il file badge.swf e chiamare le API air.swf dal vostro file SWF.

Inoltre, un file SWF in esecuzione nel browser può comunicare con un'applicazione AIR in esecuzione mediante la classe LocalConnection. Per ulteriori informazioni, vedete [Comunicazione con altre istanze di Flash Player e Adobe AIR](#) (sviluppatori ActionScript) o [Communicating with other Flash Player and AIR instances](#) (sviluppatori HTML).

Importante: per l'esecuzione delle funzionalità descritte in questa sezione e delle API nel file air.swf, è necessario che nel browser Web dell'utente finale, in Windows o Mac OS, sia installato Adobe® Flash® Player 9 aggiornamento 3 o una versione successiva. In Linux, la funzione di installazione invisibile richiede Flash Player 10 (versione 10,0,12,36 o successiva). È possibile scrivere il codice per verificare la versione installata di Flash Player e fornire all'utente un'interfaccia alternativa se la versione richiesta di Flash Player non è installata. Ad esempio, se è installata una versione precedente di Flash Player, è possibile fornire un collegamento alla versione di download del file AIR invece di utilizzare il file badge.swf o l'API air.swf per installare un'applicazione.

Uso del file badge.swf per l'installazione di un'applicazione AIR

In AIR SDK e Flex SDK è incluso un file badge.swf che consente di utilizzare facilmente la funzionalità di installazione invisibile. Tale file consente di installare il runtime e un'applicazione AIR da un collegamento in una pagina Web. Il file badge.swf file e il relativo codice sorgente vengono forniti per la distribuzione sul vostro sito Web.

Incorporare il file badge.swf in una pagina Web

- 1 Individuate i seguenti file presenti nella directory samples/badge di AIR SDK o di Flex SDK, quindi aggiungeteli al server Web.
 - badge.swf
 - default_badge.html

- AC_RunActiveContent.js

- 2 Aprite la pagina default_badge.html in un editor di testo.
- 3 Nella funzione AC_FL_RunContent () JavaScript della pagina default_badge.html, regolate le definizioni dei parametri FlashVars per i seguenti elementi:

Parametro	Descrizione
appname	Il nome dell'applicazione, visualizzato dal file SWF quando il runtime non è installato.
appurl	(Obbligatorio) L'URL del file AIR da scaricare. Utilizzate un URL assoluto, non relativo.
airversion	(Obbligatorio) Per la versione 1.0 del runtime, impostare il valore su 1.0.
imageurl	L'URL dell'immagine (facoltativo) da visualizzare nel badge.
buttoncolor	Il colore del pulsante di download, specificato come valore esadecimale, ad esempio FFCC00.
messagecolor	Il colore del messaggio di testo visualizzato sotto il pulsante quando il runtime non è installato (specificato come valore esadecimale, ad esempio FFCC00).

- 4 La dimensione minima del file badge.swf è 217 pixel di larghezza per 180 pixel di altezza. Regolate i valori dei parametri width e height della funzione AC_FL_RunContent () in modo da adattarli alle proprie esigenze.
- 5 Rinominate il file default_badge.html e regolatene il codice (oppure includetelo in un'altra pagina HTML) per adattarlo alle vostre esigenze.

Nota: per il tag HTML embed che carica il file badge.swf non impostate l'attributo wmode, ma lasciate il tag impostato sul valore predefinito ("window"). Impostazioni wmode diverse potrebbero impedire l'installazione in alcuni sistemi. L'uso di impostazioni wmode diverse produce inoltre un errore: "Error #2044: Unhandled ErrorEvent" (Errore 2044: evento di errore non gestito). text=Error #2074: The stage is too small to fit the download ui." (lo stage è troppo ridotto per l'interfaccia utente di download).

È anche possibile modificare e ricompilare il file badge.swf. Per informazioni dettagliate, vedete "[Modificare il file badge.swf](#)" a pagina 260.

Installare l'applicazione AIR da un collegamento di installazione invisibile in una pagina Web

Dopo aver aggiunto il collegamento di installazione invisibile in una pagina, l'utente può installare l'applicazione AIR facendo clic sul collegamento nel file SWF.

- 1 Accedete alla pagina HTML tramite un browser Web con installato Flash Player (versione 9 aggiornamento 3 o successiva in Windows e Mac OS o versione 10 in Linux).
- 2 Nella pagina Web, fate clic sul collegamento nel file badge.swf.
 - Se è stato installato il runtime, passate alla fase successiva.
 - In caso contrario, viene visualizzata una finestra di dialogo che chiede se si desidera installare il runtime. Installate il runtime (vedete "[Installazione di Adobe AIR](#)" a pagina 3), quindi procedete con il passaggio successivo.
- 3 Nella finestra Installazione, lasciate selezionate le impostazioni predefinite, quindi fate clic su Continua.
In un computer con sistema operativo Windows, AIR effettua automaticamente le seguenti operazioni:
 - Installa l'applicazione nella directory c:\Programmi\.
 - Crea un collegamento al desktop per l'applicazione.

- Crea un collegamento al menu Start.
- Aggiunge una voce per l'applicazione in Installazione applicazioni di Pannello di controllo.

In Mac OS, il programma di installazione aggiunge l'applicazione alla directory Applicazioni (ad esempio, nella directory /Applicazioni di Mac OS).

In un computer con sistema operativo Linux, AIR effettua automaticamente le seguenti operazioni:

- Installa l'applicazione in /opt.
- Crea un collegamento al desktop per l'applicazione.
- Crea un collegamento al menu Start.
- Aggiunge una voce relativa all'applicazione nello strumento di gestione pacchetti di sistema.

4 Selezionate le opzioni desiderate, quindi fate clic sul pulsante Installa.

5 Al termine dell'installazione, scegliete Fine.

Modificare il file badge.swf

Flex SDK e AIR SDK forniscono i file di origine per il file badge.swf. Tali file sono inclusi nella cartella samples/badge di SDK.

File di origine	Descrizione
badge fla	Il file di origine di Flash utilizzato per la compilazione di badge.swf. Il file badge fla viene compilato in un file SWF 9, che è possibile caricare in Flash Player.
AIRBadge.as	Classe di ActionScript 3.0 che definisce la classe di base utilizzata nel file badge fla.

È possibile utilizzare Flash Professional per riprogettare l'interfaccia visuale del file badge fla.

La funzione di costruzione `AIRBadge()`, definita nella classe `AIRBadge`, carica il file `air.swf` ospitato in <http://airdownload.adobe.com/air/browserapi/air.swf>. Il file `air.swf` include il codice per l'utilizzo della funzionalità di installazione invisibile.

Il metodo `onInit()` nella classe `AIRBadge` viene chiamato dopo che il file `air.swf` è stato caricato:

```
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;
    }
}
```

Il codice imposta la variabile globale `_air` sulla classe principale del file `air.swf` caricato. Tale classe include i seguenti metodi pubblici a cui accede il file `badge.swf` per chiamare la funzionalità di installazione invisibile:

Metodo	Descrizione
<code>getStatus()</code>	<p>Determina se il runtime è installato sul computer o se può essere installato. Per informazioni dettagliate, vedete “Verificare se il runtime è installato” a pagina 262.</p> <ul style="list-style-type: none"> <code>runtimeVersion</code>: una stringa che indica la versione del runtime (ad esempio, "1.0.M6") richiesta dall'applicazione da installare.
<code>installApplication()</code>	<p>Installa l'applicazione specificata sul computer dell'utente. Per informazioni dettagliate, vedete “Installazione di un'applicazione AIR dal browser” a pagina 264.</p> <ul style="list-style-type: none"> <code>url</code>: una stringa che definisce l'URL. Utilizzare un percorso di URL assoluto, non relativo. <code>runtimeVersion</code>: una stringa che indica la versione del runtime (ad esempio, "2.5.") richiesta dall'applicazione da installare. <code>arguments</code>: argomenti da trasmettere all'applicazione se viene avviata all'installazione. L'applicazione viene avviata all'installazione se l'elemento <code>allowBrowserInvocation</code> viene impostato su <code>true</code> nel file descrittore dell'applicazione. (Per ulteriori informazioni sul file descrittore dell'applicazione, vedete “File descrittore delle applicazioni AIR” a pagina 212.) Se l'applicazione viene avviata come risultato di un'installazione invisibile dal browser (con l'utente che sceglie di effettuare l'avvio all'installazione), l'oggetto <code>NativeApplication</code> dell'installazione invia un evento <code>BrowserInvokeEvent</code> solo se vengono trasmessi gli argomenti. Si considerino le implicazioni di sicurezza dei dati che vengono trasmessi all'applicazione. Per informazioni dettagliate, vedete “Avvio dal browser di un'applicazione AIR installata” a pagina 265.

Le impostazioni per `url` e `runtimeVersion` vengono trasmesse nel file SWF tramite le impostazioni di FlashVars nella pagina contenitore HTML.

Se l'applicazione viene avviata automaticamente all'installazione, è possibile utilizzare la comunicazione LocalConnection per fare in modo che l'applicazione installata contatti il file badge.swf alla chiamata. Per ulteriori informazioni, vedete [Comunicazione con altre istanze di Flash Player e Adobe AIR](#) (sviluppatori ActionScript) o [Communicating with other Flash Player and AIR instances](#) (sviluppatori HTML).

È anche possibile chiamare il metodo `getApplicationVersion()` del file `air.swf` per verificare se un'applicazione è installata. La chiamata al metodo può essere effettuata prima del processo di installazione o dopo che tale processo è stato avviato. Per informazioni dettagliate, vedete “[Verificare da una pagina Web se un'applicazione AIR è installata](#)” a pagina 263.

Caricamento del file air.swf

È possibile creare un file SWF personalizzato che utilizza le API nel file `air.swf` per interagire con il runtime e le applicazioni AIR da una pagina Web in un browser. Il file `air.swf` è ospitato in <http://airdownload.adobe.com/air/browserapi/air.swf>. Per fare riferimento alle API di `air.swf` API dal file SWF personalizzato, caricate `air.swf` nello stesso dominio dell'applicazione del file SWF. Nel codice seguente viene mostrato un esempio di caricamento del file `air.swf` nel dominio dell'applicazione del file SWF:

```
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;
}
```

Dopo che il file `air.swf` è stato caricato (quando l'oggetto `contentLoaderInfo` dell'oggetto `Loader` invia l'evento `init`), potete chiamare le API di `air.swf`, descritte nelle sezioni che seguono.

Nota: il file `badge.swf` fornito con AIR SDK e Flex SDK carica automaticamente il file `air.swf`. Vedete “[Uso del file badge.swf per l'installazione di un'applicazione AIR](#)” a pagina 258. Le istruzioni fornite in questa sezione si applicano alla creazione del file SWF personalizzato che carica il file `air.swf`.

Verificare se il runtime è installato

Un file SWF può verificare se il runtime è installato effettuando la chiamata al metodo `getStatus()` nel file `air.swf` caricato da <http://airdownload.adobe.com/air/browserapi/air.swf>. Per informazioni dettagliate, vedete “[Caricamento del file air.swf](#)” a pagina 262.

Dopo che il file `air.swf` è stato caricato, il file SWF può effettuare la chiamata al metodo `getStatus()` del file `air.swf`, come nell'esempio seguente:

```
var status:String = airSWF.getStatus();
```

Il metodo `getStatus()` restituisce uno dei seguenti valori di stringa, in base allo stato del runtime sul computer:

Valore stringa	Descrizione
"available"	Il runtime può essere installato ma attualmente non è installato in questo computer.
"unavailable"	Il runtime non può essere installato in questo computer.
"installed"	Il runtime è installato su questo computer.

Il metodo `getStatus()` genera un errore se la versione richiesta di Flash Player (versione 9 aggiornamento 3 o successiva in Windows e Mac OS o versione 10 in Linux) non è installata nel browser.

Verificare da una pagina Web se un'applicazione AIR è installata

Un file SWF può verificare se un'applicazione AIR (con un ID editore e un ID applicazione corrispondente) è installata effettuando una chiamata al metodo `getApplicationVersion()` nel file `air.swf` file caricato da <http://airdownload.adobe.com/air/browserapi/air.swf>. Per informazioni dettagliate, vedete “[Caricamento del file air.swf](#)” a pagina 262.

Dopo che il file `air.swf` è stato caricato, il file SWF può effettuare la chiamata al metodo `getApplicationVersion()` del file `air.swf`, come nell'esempio seguente:

```
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.
    }
}
```

Il metodo `getApplicationVersion()` presenta i seguenti parametri:

Parametri	Descrizione
appID	L'ID applicazione dell'applicazione. Per informazioni dettagliate, vedete "id" a pagina 234.
pubID	L'ID editore dell'applicazione. Per informazioni dettagliate, vedete "publisherID" a pagina 244. Se l'applicazione in questione non dispone di un ID editore, impostate il parametro pubID su una stringa vuota ("").
callback	Una funzione callback che serve da funzione di gestore. Il metodo <code>getApplicationVersion()</code> funziona in modo asincrono e al rilevamento della versione installata (o della mancanza di una versione installata) viene chiamato questo metodo di callback. La definizione del metodo di callback deve includere un solo parametro, una stringa, impostata sulla stringa di versione dell'applicazione installata. Se l'applicazione non è installata, alla funzione viene trasmesso un valore null, come illustrato nel precedente esempio di codice.

Il metodo `getApplicationVersion()` genera un errore se la versione richiesta di Flash Player (versione 9 aggiornamento 3 o successiva in Windows e Mac OS o versione 10 in Linux) non è installata nel browser.

Nota: a partire da AIR 1.5.3, l'ID editore è stato dichiarato obsoleto. Gli ID editore non vengono più assegnati in automatico alle applicazioni. Per garantire la compatibilità retroattiva, le applicazioni possono comunque continuare a specificare un ID editore.

Installazione di un'applicazione AIR dal browser

Un file SWF può installare un'applicazione AIR effettuando una chiamata al metodo `installApplication()` nel file `air.swf` caricato da `http://airdownload.adobe.com/air/browserapi/air.swf`. Per informazioni dettagliate, vedete "Caricamento del file `air.swf`" a pagina 262.

Dopo che il file `air.swf` è stato caricato, il file SWF può effettuare la chiamata al metodo `installApplication()` del file `air.swf`, come nel codice seguente:

```
var url:String = "http://www.example.com/myApplication.air";
var runtimeVersion:String = "1.0";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.installApplication(url, runtimeVersion, arguments);
```

Il metodo `installApplication()` installa l'applicazione specificata sul computer dell'utente. Il metodo dispone dei parametri seguenti:

Parametro	Descrizione
url	Stringa che definisce l'URL del file AIR da installare. Utilizzare un percorso di URL assoluto, non relativo.
runtimeVersion	Stringa che indica la versione del runtime (ad esempio, "1.0") richiesta dall'applicazione da installare.
arguments	Array di argomenti da trasmettere all'applicazione se viene avviata all'installazione. Negli argomenti vengono riconosciuti solo i caratteri alfanumerici. Per passare valori diversi, è possibile utilizzare uno schema di codifica. L'applicazione viene avviata all'installazione se l'elemento <code>allowBrowserInvocation</code> viene impostato su <code>true</code> nel file descrittore dell'applicazione. (Per ulteriori informazioni sul file descrittore dell'applicazione, vedete "File descrittore delle applicazioni AIR" a pagina 212.) Se l'applicazione viene avviata come risultato di un'installazione invisibile dal browser (con l'utente che sceglie di effettuare l'avvio all'installazione), l'oggetto <code>NativeApplication</code> dell'installazione invia un evento <code>BrowserInvokeEvent</code> solo se sono stati trasmessi gli argomenti. Per informazioni dettagliate, vedete "Avvio dal browser di un'applicazione AIR installata" a pagina 265.

Il metodo `installApplication()` può funzionare solo quando viene chiamato nel gestore di eventi per un evento utente (ad esempio, un clic del mouse).

Il metodo `installApplication()` genera un errore se la versione richiesta di Flash Player (versione 9 aggiornamento 3 o successiva in Windows e Mac OS o versione 10 in Linux) non è installata nel browser.

In Mac OS, per installare una versione aggiornata di un'applicazione sono necessari privilegi di sistema adeguati per effettuare l'installazione nella directory delle applicazioni, nonché privilegi di amministratore se l'applicazione aggiorna il runtime. In Windows, sono richiesti privilegi di amministratore.

È anche possibile chiamare il metodo `getApplicationVersion()` del file `air.swf` per verificare se un'applicazione è già installata. La chiamata al metodo può essere effettuata prima del processo di installazione o dopo che tale processo è stato avviato. Per informazioni dettagliate, vedete [“Verificare da una pagina Web se un'applicazione AIR è installata”](#) a pagina 263. Quando l'applicazione è stata posta in esecuzione è in grado di comunicare con il contenuto SWF del browser mediante la classe `LocalConnection`. Per ulteriori informazioni, vedete [Comunicazione con altre istanze di Flash Player e Adobe AIR](#) (sviluppatori ActionScript) o [Communicating with other Flash Player and AIR instances](#) (sviluppatori HTML).

Avvio dal browser di un'applicazione AIR installata

Per utilizzare la funzionalità di chiamata browser, che consente l'avvio dal browser, il file descrittore dell'applicazione di destinazione deve includere la seguente impostazione:

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

(Per ulteriori informazioni sul file descrittore dell'applicazione, vedete [“File descrittori delle applicazioni AIR”](#) a pagina 212.)

Un file SWF può avviare un'applicazione AIR nel browser effettuando una chiamata al metodo `launchApplication()` nel file `air.swf` caricato da <http://airdownload.adobe.com/air/browserapi/air.swf>. Per informazioni dettagliate, vedete [“Caricamento del file air.swf”](#) a pagina 262.

Dopo che il file `air.swf` è stato caricato, il file SWF può effettuare la chiamata al metodo `launchApplication()` del file `air.swf`, come nel codice seguente:

```
var appID:String = "com.example.air.myTestApplication";  
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";  
var arguments:Array = ["launchFromBrowser"]; // Optional  
airSWF.launchApplication(appID, pubID, arguments);
```

Il metodo `launchApplication()` è definito al primo livello del file `air.swf`, caricato nel dominio dell'applicazione del file SWF dell'interfaccia utente. La chiamata a questo metodo determina l'avvio dell'applicazione specificata (se è installata e se è consentita la chiamata browser tramite l'impostazione `allowBrowserInvocation` nel file descrittore dell'applicazione). Il metodo dispone dei parametri seguenti:

Parametro	Descrizione
<code>appID</code>	L'ID applicazione dell'applicazione da avviare. Per informazioni dettagliate, vedete “id” a pagina 234.
<code>pubID</code>	L'ID editore dell'applicazione da avviare. Per informazioni dettagliate, vedete “publisherID” a pagina 244. Se l'applicazione in questione non dispone di un ID editore, impostate il parametro <code>pubID</code> su una stringa vuota (<code>""</code>).
<code>arguments</code>	Un array di argomenti da passare all'applicazione. L'oggetto <code>NativeApplication</code> dell'applicazione invia un evento <code>BrowserInvokeEvent</code> con una proprietà <code>arguments</code> impostata su questo array. Negli argomenti vengono riconosciuti solo i caratteri alfanumerici. Per passare valori diversi, è possibile utilizzare uno schema di codifica.

Il metodo `launchApplication()` può funzionare solo quando viene chiamato nel gestore di eventi per un evento utente (ad esempio, un clic del mouse).

Il metodo `launchApplication()` genera un errore se la versione richiesta di Flash Player (versione 9 aggiornamento 3 o successiva in Windows e Mac OS o versione 10 in Linux) non è installata nel browser.

Se l'elemento `allowBrowserInvocation` è impostato su `false` nel file descrittore dell'applicazione, la chiamata al metodo `launchApplication()` non ha alcun effetto.

Prima di presentare l'interfaccia utente per l'avvio dell'applicazione, è possibile chiamare il metodo `getApplicationVersion()` nel file `air.swf`. Per informazioni dettagliate, vedete [“Verificare da una pagina Web se un'applicazione AIR è installata”](#) a pagina 263.

Quando l'applicazione viene chiamata tramite la funzionalità di chiamata browser, l'oggetto `NativeApplication` dell'applicazione invia un oggetto `BrowserInvokeEvent`. Per ulteriori informazioni, vedete [Chiamata di un'applicazione AIR dal browser](#) (sviluppatori ActionScript) o [Invoking an AIR application from the browser](#) (sviluppatori HTML).

Se utilizzate la funzionalità di chiamata browser, tenete presenti le implicazioni di sicurezza descritte in [Chiamata di un'applicazione AIR dal browser](#) (sviluppatori ActionScript) e [Invoking an AIR application from the browser](#) (sviluppatori HTML).

Quando l'applicazione è stata posta in esecuzione è in grado di comunicare con il contenuto SWF del browser mediante la classe `LocalConnection`. Per ulteriori informazioni, vedete [Comunicazione con altre istanze di Flash Player e Adobe AIR](#) (sviluppatori ActionScript) o [Communicating with other Flash Player and AIR instances](#) (sviluppatori HTML).

Nota: a partire da AIR 1.5.3, l'ID editore è stato dichiarato obsoleto. Gli ID editore non vengono più assegnati in automatico alle applicazioni. Per garantire la compatibilità retroattiva, le applicazioni possono comunque continuare a specificare un ID editore.

Capitolo 17: Aggiornamento di applicazioni AIR

Gli utenti possono installare o aggiornare un'applicazione AIR facendo doppio clic su un file AIR sul computer o dal browser (utilizzando la funzionalità di installazione invisibile). Il programma di installazione di Adobe® AIR® gestisce l'installazione e avvisa l'utente se sta tentando di aggiornare un'applicazione esistente

Tuttavia, è anche possibile che un'applicazione installata esegua l'aggiornamento automatico a una nuova versione utilizzando la classe Updater (un'applicazione installata può rilevare che una nuova versione è disponibile per essere scaricata e installata). La classe Updater include un metodo `update()` che consente di selezionare un file AIR sul computer dell'utente e di eseguire l'aggiornamento a quella versione. Il pacchetto dell'applicazione deve essere creato in formato di file AIR per poter utilizzare la classe Updater. Le applicazioni impacchettate come eseguibili o pacchetti nativi devono utilizzare le funzionalità di aggiornamento fornite dalla piattaforma nativa.

L'ID applicazione e l'ID editore di un file AIR di aggiornamento devono corrispondere all'applicazione da aggiornare. L'ID editore viene ricavato dal certificato di firma. Sia l'aggiornamento che l'applicazione da aggiornare devono essere firmati con lo stesso certificato.

Per AIR 1.5.3 e versioni successive, il file descrittore dell'applicazione include un elemento `<publisherID>`. È necessario utilizzare questo elemento se esistono versioni dell'applicazione sviluppate con AIR 1.5.2 o versioni precedenti. Per ulteriori informazioni, vedete “[publisherID](#)” a pagina 244.

A partire da AIR 1.1 e versioni successive, potete migrare un'applicazione per utilizzare un nuovo certificato per la firma del codice. La migrazione di un'applicazione per utilizzare una nuova firma implica la firma del file AIR aggiornato con i certificati nuovo e originale. La migrazione del certificato è un processo unidirezionale. Dopo la migrazione, solo i file AIR firmati con il nuovo certificato (o con entrambi i certificati) verranno riconosciuti come aggiornamenti di un'installazione esistente.

La gestione degli aggiornamenti delle applicazioni può essere un'operazione complessa. AIR 1.5 include il nuovo *framework di aggiornamento per le applicazioni Adobe AIR*. In questo framework sono disponibili le API che consentono agli sviluppatori di includere funzioni di aggiornamento adeguate nelle applicazioni AIR.

Potete usare la migrazione del certificato per passare da un certificato autofirmato a un certificato firmato commerciale o da un certificato autofirmato o commerciale a un altro. Se non eseguite la migrazione del certificato, gli utenti esistenti devono rimuovere la versione corrente dell'applicazione prima di installare la nuova versione. Per ulteriori informazioni, vedete “[Modifica dei certificati](#)” a pagina 201.

È consigliabile includere un meccanismo di aggiornamento nella vostra applicazione. Se create una nuova versione dell'applicazione, questo meccanismo può richiedere all'utente di installare la nuova versione.

Il programma di installazione dell'applicazione AIR crea dei file di registro quando l'applicazione viene installata, aggiornata o rimossa. Potete consultare questi registri per individuare la causa di eventuali problemi di installazione. Vedete [Installation logs](#).

Nota: le nuove versioni del runtime di Adobe AIR possono includere versioni aggiornate di WebKit. Una versione aggiornata di WebKit potrebbe determinare modifiche impreviste nel contenuto HTML di un'applicazione AIR distribuita. In alcuni casi dovrete aggiornare l'applicazione per via di queste modifiche. Un meccanismo di aggiornamento può segnalare all'utente la disponibilità della nuova versione dell'applicazione. Per ulteriori informazioni, vedete [Informazioni sull'ambiente HTML](#) (sviluppatori ActionScript) o [About the HTML environment](#) (sviluppatori HTML).

Informazioni sull'aggiornamento delle applicazioni

La classe `Updater` (del pacchetto `flash.desktop`) include un unico metodo, `update()`, che può essere utilizzato per aggiornare l'applicazione in esecuzione a una versione diversa. Ad esempio, se l'utente dispone di una versione del file AIR ("Sample_App_v2.air") che si trova sul desktop, il codice seguente aggiorna l'applicazione.

Esempio ActionScript:

```
var updater:Updater = new Updater();
var airFile:File = File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version:String = "2.01";
updater.update(airFile, version);
```

Esempio JavaScript:

```
var updater = new air.Updater();
var airFile = air.File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version = "2.01";
updater.update(airFile, version);
```

Prima che un'applicazione possa utilizzare la classe `Updater`, l'utente o l'applicazione deve scaricare la versione aggiornata del file AIR sul computer. Per ulteriori informazioni, vedete [“Scaricamento di un file AIR nel computer dell'utente”](#) a pagina 270.

Risultati della chiamata del metodo `Updater.update()`

Quando un'applicazione nel runtime chiama il metodo `update()`, il runtime chiude l'applicazione e quindi tenta di installare la nuova versione dal file AIR. Il runtime controlla che l'ID applicazione e l'ID editore specificati nel file AIR corrispondano a quelli dell'applicazione che chiama il metodo `update()`. Per informazioni sull'ID applicazione e sull'ID editore, vedete [“File descrittore delle applicazioni AIR”](#) a pagina 212. Il runtime controlla, inoltre, che la stringa della versione corrisponda alla stringa `version` passata al metodo `update()`. Se l'installazione viene effettuata correttamente, il runtime apre la nuova versione dell'applicazione. In caso contrario, viene riaperta la versione esistente (preinstallazione) dell'applicazione.

In Mac OS, per installare una versione aggiornata di un'applicazione, l'utente deve disporre dei privilegi di sistema appropriati per l'installazione nella directory dell'applicazione. In Windows e Linux, sono richiesti privilegi di amministratore.

Se la versione aggiornata dell'applicazione richiede una versione aggiornata del runtime, viene installata la nuova versione runtime. Per aggiornare il runtime, l'utente deve disporre dei privilegi amministrativi per il computer in uso.

Durante il test di un'applicazione mediante ADL, una chiamata al metodo `update()` determina un'eccezione di runtime.

Informazioni sulla stringa della versione

La stringa specificata come parametro `version` del metodo `update()` deve corrispondere alla stringa nell'elemento `version` o `versionNumber` del file descrittore dell'applicazione per il file AIR da installare. L'indicazione del parametro `version` è richiesta per motivi di sicurezza. Richiedendo all'applicazione di verificare il numero di versione nel file AIR, l'applicazione evita di installare inavvertitamente una versione vecchia, che potrebbe contenere vulnerabilità di sicurezza successivamente risolte. L'applicazione deve anche confrontare la stringa della versione nel file AIR con quella nell'applicazione installata per evitare attacchi di downgrade.

Prima di AIR 2.5, la stringa di versione può essere in qualsiasi formato. Può essere, ad esempio, "2.01" o "versione 2". A partire da AIR 2.5, la stringa di versione deve essere una sequenza composta da fino a tre serie con un massimo di tre cifre ciascuna, separate da punti. Ad esempio, ".0", "1.0" e "67.89.999" sono tutti numeri di versione validi. Dovete convalidare la stringa della versione di aggiornamento prima di aggiornare l'applicazione.

Se un'applicazione Adobe AIR scarica un file AIR tramite il Web, è opportuno disporre di un meccanismo che consente al servizio Web di inviare una notifica riguardo la versione scaricata. L'applicazione può quindi utilizzare questa stringa come parametro `version` del metodo `update()`. Se il file AIR file è ottenuto in modo diverso e la versione del file AIR non è nota, l'applicazione AIR può esaminare il file AIR per determinare la versione (un file AIR è un archivio compresso in formato ZIP e il file descrittore dell'applicazione è il secondo record nell'archivio).

Per ulteriori informazioni sul file descrittore dell'applicazione, vedete [“File descrittori delle applicazioni AIR”](#) a pagina 212.

Flusso di lavoro di firma per gli aggiornamenti delle applicazioni

La pubblicazione di aggiornamenti ad-hoc complica il compito di gestire più versioni delle applicazioni e le date di scadenza dei relativi certificati. I certificati potrebbero infatti scadere prima che possiate pubblicare un aggiornamento.

Il runtime Adobe AIR tratta un aggiornamento di un'applicazione pubblicato senza una firma di migrazione come se fosse una nuova applicazione. Gli utenti devono quindi disinstallare l'applicazione AIR corrente per poter installare l'aggiornamento.

Per risolvere il problema, caricate ogni applicazione aggiornata con il certificato più recente su un URL di distribuzione distinto. Includete un meccanismo che vi ricordi di applicare firme di migrazione quando il certificato entra nel periodo di tolleranza di 180 giorni. Per ulteriori informazioni, vedete [“Firma di una versione aggiornata di un'applicazione AIR”](#) a pagina 206.

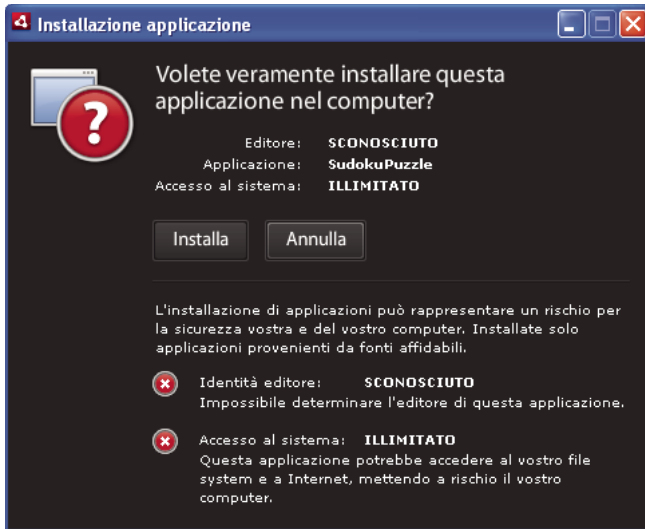
Vedete [“Comandi ADT”](#) a pagina 171 per informazioni su come applicare le firme.

Effettuate le seguenti operazioni per semplificare il processo di applicazione delle firme di migrazione:

- Caricate ogni applicazione aggiornata su un URL di distribuzione distinto.
- Caricate sullo stesso URL il file XML descrittore dell'aggiornamento e il certificato più recente.
- Firmate l'applicazione aggiornate con il certificato più recente.
- Applicate una firma di migrazione all'applicazione aggiornata con il certificato utilizzato per firmare la versione precedente caricata su un URL diverso.

Presentazione di un'interfaccia di aggiornamento applicazioni personalizzata

AIR include un'interfaccia di aggiornamento predefinita:



Questa interfaccia è sempre utilizzata la prima volta che un utente installa una versione dell'applicazione su una macchina. Tuttavia, potete definire un'interfaccia personalizzata da utilizzare per istanze successive. Se nell'applicazione è definita un'interfaccia di aggiornamento personalizzata, specificate un elemento `customUpdateUI` nel file descrittore dell'applicazione per l'applicazione attualmente installata:

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

Quando l'applicazione è installata e l'utente apre un file AIR con un ID applicazione e un ID editore che corrispondono all'applicazione installata, il runtime apre l'applicazione, anziché il programma di installazione dell'applicazione AIR predefinita. Per ulteriori informazioni, vedete “[customUpdateUI](#)” a pagina 224.

L'applicazione può decidere, quando viene eseguita (quando l'oggetto `NativeApplication.nativeApplication` invia un evento `load`), se aggiornare l'applicazione (utilizzando la classe `Updater`). Se decide di eseguire l'aggiornamento, può presentare all'utente la propria interfaccia di installazione (che è diversa dall'interfaccia di esecuzione standard).

Scaricamento di un file AIR nel computer dell'utente

Per utilizzare la classe `Updater`, l'utente o l'applicazione deve prima salvare un file AIR localmente sul computer dell'utente.

Nota: in AIR 1.5 è incluso un framework di aggiornamento che consente agli sviluppatori di includere funzioni di aggiornamento adeguate nelle applicazioni AIR. L'uso di questo framework può risultare molto più facile rispetto all'uso diretto del metodo `update()` della classe `Update`. Per ulteriori dettagli, vedete “[Uso del framework di aggiornamento](#)” a pagina 274.

Il codice seguente legge un file AIR da un URL (http://example.com/air/updates/Sample_App_v2.air) e salva il file AIR nella directory di memorizzazione dell'applicazione.

Esempio ActionScript:

```
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.");
}
```

Esempio JavaScript:

```
var urlString = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq = new air.URLRequest(urlString);
var urlStream = new air.URLStream();
var fileData = new air.ByteArray();
urlStream.addEventListener(air.Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event) {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile() {
    var file = air.File.desktopDirectory.resolvePath("My App v2.air");
    var fileStream = new air.FileStream();
    fileStream.open(file, air.FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

Per ulteriori informazioni, vedete:

- [Flusso di lavoro per lettura e scrittura di file](#) (sviluppatori ActionScript)
- [Workflow for reading and writing files](#) (sviluppatori HTML)

Verificare se un'applicazione viene eseguita per la prima volta

Dopo aver completato l'aggiornamento di un'applicazione, potete inviare all'utente un messaggio "operazioni preliminari" o "benvenuto". All'avvio, l'applicazione esegue un controllo per verificare se è la prima volta che viene eseguita, in modo da poter determinare se visualizzare il messaggio.

***Nota:** in AIR 1.5 è incluso un framework di aggiornamento che consente agli sviluppatori di includere funzioni di aggiornamento adeguate nelle applicazioni AIR. Questo framework fornisce dei metodi semplici per controllare se una versione di un'applicazione viene eseguita per la prima volta. Per ulteriori dettagli, vedete ["Uso del framework di aggiornamento"](#) a pagina 274.*

Un modo per eseguire questa operazione è salvare un file nella directory di memorizzazione dell'applicazione durante l'inizializzazione dell'applicazione. A ogni avvio dell'applicazione, è necessario verificare l'esistenza di questo file. Se il file non esiste, allora è la prima volta che l'applicazione è in esecuzione per l'utente corrente. Se il file esiste, l'applicazione è stata già eseguita almeno una volta. Se il file esiste e contiene un numero di versione precedente a quello corrente, allora è la prima volta che l'utente esegue la nuova versione.

Nell'esempio seguente di Flex viene dimostrato il concetto:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    title="Sample Version Checker Application"
    applicationComplete="system extension()">
  <mx:Script>
    <![CDATA [
      import flash.filesystem.*;
      public var file:File;
      public var currentVersion:String = "1.2";
      public function system extension():void {
        file = File.applicationStorageDirectory;
        file = file.resolvePath("Preferences/version.txt");
        trace(file.nativePath);
        if(file.exists) {
          checkVersion();
        } else {
          firstRun();
        }
      }
      private function checkVersion():void {
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var reversion:String = stream.readUTFBytes(stream.bytesAvailable);
        stream.close();
        if (reversion != currentVersion) {
          log.text = "You have updated to version " + currentVersion + ".\n";
        }
      }
    ]]>
  </mx:Script>
</mx:WindowedApplication>
```

```
        } else {
            saveFile();
        }
        log.text += "Welcome to the application.";
    }
    private function firstRun():void {
        log.text = "Thank you for installing the application. \n"
            + "This is the first time you have run it.";
        saveFile();
    }
    private function saveFile():void {
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.WRITE);
        stream.writeUTFBytes(currentVersion);
        stream.close();
    }
    ]]>
</mx:Script>
<mx:TextArea ID="log" width="100%" height="100%" />
</mx:WindowedApplication>
```

Il seguente esempio dimostra il concetto in JavaScript:

```
<html>
  <head>
    <script src="AIRAliases.js" />
    <script>
      var file;
      var currentVersion = "1.2";
      function system extension() {
        file = air.File.appStorageDirectory.resolvePath("Preferences/version.txt");
        air.trace(file.nativePath);
        if(file.exists) {
          checkVersion();
        } else {
          firstRun();
        }
      }
      function checkVersion() {
        var stream = new air.FileStream();
        stream.open(file, air.FileMode.READ);
        var reversion = stream.readUTFBytes(stream.bytesAvailable);
        stream.close();
        if (reversion != currentVersion) {
          window.document.getElementById("log").innerHTML
            = "You have updated to version " + currentVersion + ".\n";
        } else {
          saveFile();
        }
        window.document.getElementById("log").innerHTML
          += "Welcome to the application.";
      }
    </script>
  </head>
  <body>
    <div id="log">
    </div>
  </body>
</html>
```

```
    }  
    function firstRun() {  
        window.document.getElementById("log").innerHTML  
            = "Thank you for installing the application. \n"  
            + "This is the first time you have run it.";  
        saveFile();  
    }  
    function saveFile() {  
        var stream = new air.FileStream();  
        stream.open(file, air.FileMode.WRITE);  
        stream.writeUTFBytes(currentVersion);  
        stream.close();  
    }  
    </script>  
</head>  
<body onLoad="system extension()">  
    <textarea ID="log" rows="100%" cols="100%" />  
</body>  
</html>
```

Se l'applicazione salva i dati localmente (ad esempio, nella directory di memorizzazione dell'applicazione), è opportuno verificare la presenza di eventuali dati salvati (dalle versioni precedenti) al momento del primo avvio.

Uso del framework di aggiornamento

La gestione degli aggiornamenti delle applicazioni può essere un'operazione laboriosa. Il *framework di aggiornamento per applicazioni Adobe AIR* fornisce delle API che permettono agli sviluppatori di integrare solide funzionalità di aggiornamento nelle applicazioni AIR. Il framework di aggiornamento AIR effettua le seguenti operazioni per gli sviluppatori:

- Controllare regolarmente la disponibilità di aggiornamenti in base a un intervallo di tempo prestabilito o alle richieste degli utenti
- Scaricare i file AIR (aggiornamenti) da un'origine Web
- Avvisare l'utente alla prima esecuzione di una nuova versione installata
- Verificare che l'utente desidera controllare la disponibilità di aggiornamenti
- Visualizzare all'utente informazioni relative alla nuova versione di aggiornamento
- Visualizzazione all'utente informazioni sull'avanzamento del download e sugli eventuali errori

Nel framework di aggiornamento di AIR è disponibile un'interfaccia utente di esempio che potete usare per la vostra applicazione. Fornisce all'utente le informazioni di base e le opzioni di configurazione relative agli aggiornamenti dell'applicazione. Nell'applicazione potete anche definire un'interfaccia utente personalizzata da usare con il framework di aggiornamento.

Il framework di aggiornamento di AIR consente di archiviare le informazioni relative alla versione di aggiornamento di un'applicazione AIR in semplici file di configurazione XML. Per la maggior parte delle applicazioni, l'impostazione di questi file di configurazione con l'inclusione di codice di base fornisce all'utente delle buone funzionalità di aggiornamento.

Anche senza utilizzare il framework di aggiornamento, Adobe AIR include una classe Updater che può essere usata dalle applicazioni AIR per l'aggiornamento alle nuove versioni. La classe Updater consente l'aggiornamento di un'applicazione a una versione contenuta in un file AIR sul computer dell'utente. La gestione degli aggiornamenti può tuttavia comportare operazioni più complesse del semplice aggiornamento di un'applicazione basato su un file AIR archiviato localmente.

File del framework di aggiornamento di AIR

Il framework di aggiornamento di AIR si trova nella directory `frameworks/libs/air` dell'SDK di AIR 2 e comprende i seguenti file:

- `applicationupdater.swc` - Definisce la funzionalità di base della libreria di aggiornamenti (da utilizzare in ActionScript). Questa versione non contiene alcuna interfaccia utente.
- `applicationupdater.swf` - Definisce la funzionalità di base della libreria di aggiornamenti (da utilizzare in JavaScript). Questa versione non contiene alcuna interfaccia utente.
- `applicationupdater_ui.swc` - Definisce una versione Flex 4 della funzionalità di base della libreria di aggiornamenti, includendo un'interfaccia utente che può essere usata dall'applicazione per visualizzare le opzioni di aggiornamento.
- `applicationupdater_ui.swf` - Definisce una versione JavaScript della funzionalità di base della libreria di aggiornamenti, includendo un'interfaccia utente che può essere usata dall'applicazione per visualizzare le opzioni di aggiornamento.

Per ulteriori informazioni, vedete le seguenti sezioni:

- [“Impostazione dell'ambiente di sviluppo Flex”](#) a pagina 275
- [“Inclusione dei file del framework in un'applicazione AIR basata su HTML”](#) a pagina 276
- [“Esempio di base: Uso della versione ApplicationUpdaterUI”](#) a pagina 276

Impostazione dell'ambiente di sviluppo Flex

I file SWC contenuti nella directory `frameworks/libs/air` dell'SDK di AIR 2 definiscono le classi che potete utilizzare per lo sviluppo in Flex e in Flash.

Per usare il framework di aggiornamento quando effettuate la compilazione con Flex SDK, includete il file `ApplicationUpdater.swc` o `ApplicationUpdater_UI.swc` nella chiamata al compilatore `amxmlc`. Nell'esempio riportato di seguito, il compilatore carica il file `ApplicationUpdater.swc` nella sottodirectory `lib` della directory Flex SDK:

```
amxmlc -library-path+=lib/ApplicationUpdater.swc -- myApp.mxml
```

Nell'esempio riportato di seguito, il compilatore carica il file `ApplicationUpdater_UI.swc` nella sottodirectory `lib` della directory Flex SDK:

```
amxmlc -library-path+=lib/ApplicationUpdater_UI.swc -- myApp.mxml
```

Quando sviluppate un'applicazione con Flash Builder, aggiungete il file SWC nella scheda relativa al percorso della libreria nelle impostazioni del percorso di Flex Builder nella finestra di dialogo delle proprietà.

Assicuratevi di copiare i file SWC nella directory a cui farete riferimento nel compilatore `amxmlc` (usando Flex SDK) o in Flash Builder.

Inclusione dei file del framework in un'applicazione AIR basata su HTML

Nella directory `frameworks/html` del framework di aggiornamento sono inclusi i seguenti file SWF:

- `applicationupdater.swf` - Definisce la funzionalità di base della libreria di aggiornamenti, senza interfaccia utente
- `applicationupdater_ui.swf` - Definisce la funzionalità di base della libreria di aggiornamenti, includendo un'interfaccia utente che può essere usata dall'applicazione per visualizzare le opzioni di aggiornamento

Il codice JavaScript nelle applicazioni AIR può usare le classi definite nei file SWF.

Per usare il framework di aggiornamento, includete il file `applicationupdater.swf` o `applicationupdater_ui.swf` nella directory (o in una sottodirectory) dell'applicazione. Quindi, nel file HTML che userà il framework (nel codice JavaScript), includete un tag `script` che carica il file:

```
<script src="applicationUpdater.swf" type="application/x-shockwave-flash"/>
```

In alternativa, usate questo tag `script` per caricare il file `applicationupdater_ui.swf`:

```
<script src="applicationupdater_ui.swf" type="application/x-shockwave-flash"/>
```

L'API definita in questi due file viene descritta nella parte rimanente di questo documento.

Esempio di base: Uso della versione ApplicationUpdaterUI

La versione `ApplicationUpdaterUI` del framework di aggiornamento fornisce un'interfaccia di base che potete usare facilmente nella vostra applicazione. Di seguito è riportato un esempio di base.

Create innanzitutto un'applicazione AIR che chiama il framework di aggiornamento:

- 1 Se la vostra applicazione è un'applicazione AIR basata su HTML, caricate il file `applicationupdaterui.swf`:

```
<script src="ApplicationUpdater_UI.swf" type="application/x-shockwave-flash"/>
```

- 2 Nella logica della vostra applicazione AIR, create un'istanza dell'oggetto `ApplicationUpdaterUI`.

In `ActionScript`, usate il seguente codice:

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

In `JavaScript`, usate il seguente codice:

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

Potete aggiungere questo codice a una funzione di inizializzazione che viene eseguita dopo il caricamento dell'applicazione.

- 3 Create un file di testo denominato `updateConfig.xml` e aggiungetevi quanto riportato di seguito:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
  <url>http://example.com/updates/update.xml</url>
  <delay>1</delay>
</configuration>
```

Modificate l'elemento `URL` del file `updateConfig.xml` in modo che corrisponda alla posizione finale del file descrittore dell'aggiornamento sul server Web (vedete la procedura successiva).

Il valore `delay` è il numero di giorni che l'applicazione attende prima di controllare se sono presenti aggiornamenti.

- 4 Aggiungete il file `updateConfig.xml` alla directory `project` dell'applicazione AIR.
- 5 Impostate l'oggetto `updater` in modo che faccia riferimento al file `updateConfig.xml` e chiamate il metodo `initialize()` dell'oggetto.

In ActionScript, usate il seguente codice:

```
appUpdater.configurationFile = new File("app:/updateConfig.xml");  
appUpdater.initialize();
```

In JavaScript, usate il seguente codice:

```
appUpdater.configurationFile = new air.File("app:/updateConfig.xml");  
appUpdater.initialize();
```

- 6 Create una seconda versione dell'applicazione AIR con una versione diversa rispetto alla prima applicazione. (La versione è specificata nel file descrittore dell'applicazione, nell'elemento `version`).

Aggiungete quindi la versione di aggiornamento dell'applicazione AIR al server Web:

- 1 Copiate la versione di aggiornamento del file AIR sul server Web.
- 2 Create un file di testo denominato `updateDescriptor.2.5.xml` e aggiungetevi il contenuto riportato di seguito:

```
<?xml version="1.0" encoding="utf-8"?>  
  <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">  
    <versionNumber>1.1</versionNumber>  
    <url>http://example.com/updates/sample_1.1.air</url>  
    <description>This is the latest version of the Sample application.</description>  
  </update>
```

Modificate gli elementi `versionNumber`, URL e `description` del file `updateDescriptor.xml` in modo che corrispondano al file AIR di aggiornamento. Questo formato del descrittore di aggiornamento viene utilizzato dalle applicazioni che usano il framework di aggiornamento incluso in AIR 2.5 SDK (e versioni successive).

- 3 Create un file di testo denominato `updateDescriptor.1.0.xml` e aggiungetevi il contenuto riportato di seguito:

```
<?xml version="1.0" encoding="utf-8"?>  
  <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">  
    <version>1.1</version>  
    <url>http://example.com/updates/sample_1.1.air</url>  
    <description>This is the latest version of the Sample application.</description>  
  </update>
```

Modificate gli elementi `version`, URL e `description` del file `updateDescriptor.xml` in modo che corrispondano al file AIR di aggiornamento. Questo formato del descrittore di aggiornamento viene utilizzato dalle applicazioni che usano il framework di aggiornamento incluso in AIR 2 SDK (e versioni precedenti).

Nota: è necessario creare questo secondo file descrittore di aggiornamento solo se dovete supportare gli aggiornamenti ad applicazioni create prima di AIR 2.5.

- 4 Aggiungete i file `updateDescriptor.2.5.xml` e `updateDescriptor.1.0.xml` alla stessa directory del server web che contiene il file AIR di aggiornamento.

Questo è un esempio di base che tuttavia fornisce una funzionalità di aggiornamento sufficiente per molte applicazioni. Nella parte restante di questo documento viene descritto come utilizzare il framework di aggiornamento per soddisfare le vostre esigenze specifiche.

Per un altro esempio di utilizzo del framework di aggiornamento, consultate la seguente applicazione di esempio in Adobe AIR Developer Center:

- [Framework di aggiornamento in un'applicazione Flash](http://www.adobe.com/go/learn_air_qs_update_framework_flash_it)
(http://www.adobe.com/go/learn_air_qs_update_framework_flash_it)

Aggiornamento ad AIR 2.5

Poiché le regole per assegnare numeri di versione alle applicazioni sono cambiate in AIR 2.5, il framework di aggiornamento AIR non è in grado di analizzare le informazioni di versione di un descrittore AIR 2.5. Questa incompatibilità comporta la necessità di aggiornare l'applicazione utilizzando il nuovo framework di aggiornamento PRIMA di aggiornare l'applicazione per l'uso di AIR 2.5 SDK. In altre parole, l'aggiornamento dell'applicazione ad AIR 2.5 o successivo da qualunque versione di AIR precedente alla 2.5 richiede in realtà DUE aggiornamenti. Il primo aggiornamento deve usare lo spazio dei nomi AIR 2 e includere la libreria del framework di aggiornamento AIR 2.5 (potete comunque creare il pacchetto dell'applicazione utilizzando AIR 2.5 SDK). Per il secondo aggiornamento potete usare lo spazio dei nomi AIR 2.5 e includere le nuove funzioni dell'applicazione.

Potete inoltre fare in modo che l'aggiornamento intermedio non abbia alcun effetto oltre a quello di aggiornare l'applicazione AIR 2.5 utilizzando direttamente la classe AIR Updater.

L'esempio seguente mostra come aggiornare un'applicazione dalla versione 1.0 a 2.0. La versione 1.0 usa il vecchio spazio dei nomi 2.0. La versione 2.0 usa invece lo spazio dei nomi 2.5 e include nuove funzioni implementate con le API AIR 2.5.

1 Create una versione intermedia dell'applicazione (1.0.1), basata sulla versione 1.0.

a Usate il framework Application Updater di AIR 2.5 per creare l'applicazione.

***Nota:** utilizzate `applicationupdater.swc` o `applicationupdater_ui.swc` per le applicazioni AIR basate sulla tecnologia Flash e `applicationupdater.swf` o `applicationupdater_ui.swf` per quelle basate su HTML.*

b Create un file descrittore di aggiornamento per la versione 1.0.1 utilizzando il vecchio spazio dei nomi e la versione così come nell'esempio seguente:

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/2.0">
    <version>1.0.1</version>
    <url>http://example.com/updates/sample_1.0.1.air</url>
    <description>This is the intermediate version.</description>
  </update>
```

2 Create la versione 2.0 dell'applicazione che utilizza le API AIR 2.5 e lo spazio dei nomi 2.5.

3 Create un descrittore di aggiornamento per aggiornare l'applicazione dalla versione 1.0.1 alla versione 2.0.

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
    <version>2.0</version>
    <url>http://example.com/updates/sample_2.0.air</url>
    <description>This is the intermediate version.</description>
  </update>
```

Definizione dei file descrittori dell'aggiornamento e aggiunta del file AIR al server Web

Quando usate il framework di aggiornamento di AIR, definite le informazioni di base sull'aggiornamento disponibile nei file descrittori dell'aggiornamento archiviati sul server Web. Un file descrittore dell'aggiornamento è un semplice file XML. Il framework di aggiornamento incluso nell'applicazione controlla questo file per verificare se è stata caricata una nuova versione.

Il formato del file descrittore dell'aggiornamento è cambiato in AIR 2.5. Il nuovo formato usa uno spazio dei nomi diverso. Lo spazio dei nomi originale è "http://ns.adobe.com/air/framework/update/description/1.0". Lo spazio dei nomi AIR 2.5 è "http://ns.adobe.com/air/framework/update/description/2.5".

Le applicazioni AIR create prima di AIR 2.5 possono leggere solo la versione 1.0 del descrittore di aggiornamento. Le applicazioni AIR create con il framework di aggiornamento incluso in AIR 2.5 o successivo possono leggere solo la versione 2.5 del descrittore di aggiornamento. A causa di questa incompatibilità tra versioni, spesso è necessario creare due file descrittore di aggiornamento. La logica di aggiornamento nelle versioni AIR 2.5 dell'applicazione deve scaricare un descrittore di aggiornamento che utilizza il nuovo formato, mentre le versioni dell'applicazione AIR devono continuare a utilizzare il formato originale. Entrambi i file devono essere modificati per ogni aggiornamento che rilascerete (finché non cesserete di supportare le versioni create prima di AIR 2.5).

Il file descrittore dell'aggiornamento contiene i seguenti dati:

- `versionNumber` - La nuova versione dell'applicazione AIR. Usate l'elemento `versionNumber` nei descrittori di aggiornamento utilizzati per aggiornare le applicazioni AIR 2.5. Il valore deve corrispondere alla stringa inclusa nell'elemento `versionNumber` del nuovo file descrittore dell'applicazione AIR. Se il numero di versione nel file descrittore dell'applicazione non corrisponde a quello del file AIR di aggiornamento, il framework di aggiornamento genera un'eccezione.
- `version` - La nuova versione dell'applicazione AIR. Usate l'elemento `version` nei descrittori di aggiornamento utilizzati per aggiornare le applicazioni create prima di AIR 2.5. Il valore deve corrispondere alla stringa inclusa nell'elemento `version` del nuovo file descrittore dell'applicazione AIR. Se la versione nel file descrittore dell'applicazione non corrisponde alla versione del file AIR di aggiornamento, il framework di aggiornamento genera un'eccezione.
- `versionLabel` - La stringa di versione in formato leggibile all'uomo da visualizzare agli utenti. `versionLabel` è opzionale, ma può essere specificato solo nei file descrittore di aggiornamento 2.5 o successivi. Utilizzatelo se specificate anche l'elemento `versionLabel` nel descrittore dell'applicazione e impostatelo sullo stesso valore.
- `url` - La posizione del file AIR di aggiornamento. Si tratta del file che contiene la versione di aggiornamento dell'applicazione AIR.
- `description` - Dettagli relativi alla nuova versione. Queste informazioni possono essere visualizzate all'utente durante il processo di aggiornamento.

Gli elementi `version` e `url` sono obbligatori. L'elemento `description` è opzionale.

Di seguito è riportato un esempio di file descrittore dell'aggiornamento in versione 2.5:

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
    <versionNumber>1.1.1</versionNumber>
    <url>http://example.com/updates/sample_1.1.1.air</url>
    <description>This is the latest version of the Sample application.</description>
  </update>
```

E questo è un esempio di file descrittore dell'aggiornamento in versione 1.0:

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
    <version>1.1.1</version>
    <url>http://example.com/updates/sample_1.1.1.air</url>
    <description>This is the latest version of the Sample application.</description>
  </update>
```

Se desiderate definire il tag `description` in più lingue, usate più elementi `text` che definiscono un attributo `lang`:

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
    <versionNumber>1.1.1</versionNumber>
    <url>http://example.com/updates/sample_1.1.1.air</url>
    <description>
      <text xml:lang="en">English description</text>
      <text xml:lang="fr">French description</text>
      <text xml:lang="ro">Romanian description</text>
    </description>
  </update>
```

Copiate il file descrittore dell'aggiornamento, insieme al file AIR di aggiornamento, sul server Web.

Nella directory templates inclusa nel descrittore dell'aggiornamento sono disponibili esempi di file descrittori dell'aggiornamento. Sono incluse versioni sia monolingua che multilingua.

Creazione di un'istanza di un oggetto updater

Dopo avere caricato il framework di aggiornamento di AIR nel codice (vedete [“Impostazione dell'ambiente di sviluppo Flex”](#) a pagina 275 e [“Inclusione dei file del framework in un'applicazione AIR basata su HTML”](#) a pagina 276), dovete creare un'istanza di un oggetto updater, come nell'esempio seguente.

Esempio ActionScript:

```
var appUpdater:ApplicationUpdater = new ApplicationUpdater();
```

Esempio JavaScript:

```
var appUpdater = new runtime.air.update.ApplicationUpdater();
```

Nel codice precedente viene usata la classe `ApplicationUpdater` (che non fornisce un'interfaccia utente). Se desiderate usare la classe `ApplicationUpdaterUI` (che fornisce un'interfaccia utente), usate quanto segue.

Esempio ActionScript:

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

Esempio JavaScript:

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

Negli altri esempi di codice presenti in questo documento si presuppone che abbiate creato un'istanza di un oggetto updater denominato `appUpdater`.

Configurazione delle impostazioni di aggiornamento

Potete configurare sia `ApplicationUpdater` che `ApplicationUpdaterUI` tramite un file di configurazione fornito con l'applicazione oppure tramite ActionScript o JavaScript nell'applicazione.

Definizione delle impostazioni di aggiornamento in un file di configurazione XML

Il file di configurazione dell'aggiornamento è un file in formato XML che può contenere i seguenti elementi:

- `updateURL` - Un oggetto String, che rappresenta la posizione del descrittore dell'aggiornamento sul server remoto. È consentita qualsiasi posizione URLRequest valida. Dovete definire la proprietà `updateURL` tramite il file di configurazione oppure tramite uno script (consultate [“Definizione dei file descrittori dell'aggiornamento e aggiunta del file AIR al server Web”](#) a pagina 278). Questa proprietà deve essere definita prima di usare l'oggetto updater (prima di chiamare il metodo `initialize()` dell'oggetto updater, descritto in [“Inizializzazione del framework di aggiornamento”](#) a pagina 283).

- `delay` - Un numero che rappresenta un intervallo di tempo espresso in giorni (sono consentiti valori come 0.25) per il controllo degli aggiornamenti. Un valore 0 (che corrisponde al valore predefinito) specifica che l'oggetto `updater` non esegue automaticamente un controllo periodico.

Il file di configurazione per `ApplicationUpdaterUI` può contenere il seguente elemento, oltre agli elementi `updateURL` e `delay`:

- `defaultUI`: un elenco di elementi `dialog`. Ogni elemento `dialog` ha un attributo `name` che corrisponde alla finestra di dialogo nell'interfaccia utente. Ogni elemento `dialog` ha un attributo `visible` che definisce se la finestra di dialogo è visibile. Il valore predefinito è `true`. di seguito sono riportati i possibili valori per l'attributo `name`:
 - `"checkForUpdate"` - Corrisponde alle finestre di dialogo Check for Update, No Update e Update Error (Verifica disponibilità aggiornamenti, Nessun aggiornamento ed Errore durante l'aggiornamento)
 - `"downloadUpdate"` - Corrisponde alla finestra di dialogo Download Update (Scarica aggiornamento)
 - `"downloadProgress"` - Corrisponde alle finestre di dialogo Download Progress e Download Error (Stato scaricamento ed Errore durante lo scaricamento)
 - `"installUpdate"` - Corrisponde alla finestra di dialogo Install Update (Installa aggiornamento)
 - `"fileUpdate"` - Corrisponde alle finestre di dialogo File Update, File No Update e File Error (Aggiornamento file, Nessun aggiornamento del file ed Errore del file)
- `"unexpectedError"` - Corrisponde alla finestra di dialogo Unexpected Error (Errore imprevisto)

Quando è impostato su `false`, la finestra di dialogo corrispondente non viene visualizzata durante la procedura di aggiornamento.

Di seguito è riportato un esempio del file di configurazione per il framework `ApplicationUpdater`:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
  <url>http://example.com/updates/update.xml</url>
  <delay>1</delay>
</configuration>
```

Di seguito è riportato un esempio del file di configurazione per il framework `ApplicationUpdaterUI`, che include una definizione per l'elemento `defaultUI`:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
  <url>http://example.com/updates/update.xml</url>
  <delay>1</delay>
  <defaultUI>
    <dialog name="checkForUpdate" visible="false" />
    <dialog name="downloadUpdate" visible="false" />
    <dialog name="downloadProgress" visible="false" />
  </defaultUI>
</configuration>
```

Puntate la proprietà `configurationFile` verso la posizione di quel file:

Esempio ActionScript:

```
appUpdater.configurationFile = new File("app:/cfg/updateConfig.xml");
```

Esempio JavaScript:

```
appUpdater.configurationFile = new air.File("app:/cfg/updateConfig.xml");
```

Nella directory `templates` del framework di aggiornamento è incluso il file di configurazione di esempio `config-template.xml`.

Definizione delle impostazioni di aggiornamento nel codice ActionScript o JavaScript

Questi parametri di configurazione possono essere impostati anche usando il codice nell'applicazione, come nell'esempio seguente:

```
appUpdater.updateURL = " http://example.com/updates/update.xml ";  
appUpdater.delay = 1;
```

Le proprietà dell'oggetto `updater` sono `updateURL` e `delay`. Queste proprietà definiscono le stesse impostazioni degli elementi `updateURL` e `delay` nel file di configurazione: l'URL del file descrittore dell'aggiornamento e l'intervallo per il controllo degli aggiornamenti. Se specificate un file di configurazione e le impostazioni nel codice, le proprietà impostate usando il codice hanno la precedenza sulle impostazioni corrispondenti nel file di configurazione.

Dovete definire la proprietà `updateURL` tramite il file di configurazione oppure tramite uno script (consultate [“Definizione dei file descrittore dell'aggiornamento e aggiunta del file AIR al server Web”](#) a pagina 278), prima di utilizzare l'oggetto `updater` (prima di chiamare il metodo `initialize()` dell'oggetto `updater`, descritto in [“Inizializzazione del framework di aggiornamento”](#) a pagina 283).

Il framework `ApplicationUpdaterUI` definisce queste proprietà aggiuntive dell'oggetto `updater`:

- `isCheckForUpdateVisible` - Corrisponde alle finestre di dialogo `Check for Update`, `No Update` e `Update Error` (Verifica disponibilità aggiornamenti, Nessun aggiornamento ed Errore durante l'aggiornamento)
- `isDownloadUpdateVisible` - Corrisponde alla finestra di dialogo `Download Update` (Scarica aggiornamento)
- `isDownloadProgressVisible` - Corrisponde alle finestre di dialogo `Download Progress` e `Download Error` (Stato scaricamento ed Errore durante lo scaricamento)
- `isInstallUpdateVisible` - Corrisponde alla finestra di dialogo `Install Update` (Installa aggiornamento)
- `isFileUpdateVisible` - Corrisponde alle finestre di dialogo `File Update`, `File No Update` e `File Error` (Aggiornamento file, Nessun aggiornamento del file ed Errore del file)
- `isUnexpectedErrorVisible` - Corrisponde alla finestra di dialogo `Unexpected Error` (Errore imprevisto)

Ogni proprietà corrisponde a una o più finestre di dialogo nell'interfaccia utente `ApplicationUpdaterUI`. Ogni proprietà è un valore booleano, con un valore predefinito `true`. Quando è impostato su `false`, le finestre di dialogo corrispondenti non vengono visualizzate durante la procedura di aggiornamento.

Queste proprietà delle finestre di dialogo hanno la precedenza sulle impostazioni nel file di configurazione dell'aggiornamento.

Processo di aggiornamento

Il framework di aggiornamento di AIR completa il processo di aggiornamento nei seguenti passaggi:

- 1 L'inizializzazione dell'oggetto `updater` verifica se è stato effettuato un controllo per rilevare la presenza di un aggiornamento entro l'intervallo di tempo definito (vedete [“Configurazione delle impostazioni di aggiornamento”](#) a pagina 280). Se è previsto un controllo dell'aggiornamento, il processo continua.
- 2 L'oggetto `updater` scarica e interpreta il file descrittore dell'aggiornamento.
- 3 L'oggetto `updater` scarica il file AIR di aggiornamento.
- 4 L'oggetto `updater` installa la versione aggiornata dell'applicazione.

L'oggetto `updater` invia degli eventi al completamento di ognuno di questi passaggi. Nella versione `ApplicationUpdater`, potete annullare gli eventi che indicano l'avvenuto completamento di un passaggio del processo. Se annullate uno di questi eventi, il passaggio successivo del processo viene annullato. Nella versione `ApplicationUpdaterUI`, l'oggetto `updater` presenta una finestra di dialogo che consente all'utente di annullare ogni singolo passaggio del processo o di procedere.

Se annullate l'evento, potete chiamare i metodi dell'oggetto `updater` per riprendere il processo.

Mentre la versione `ApplicationUpdater` procede nel processo di aggiornamento, registra il relativo stato corrente nella proprietà `currentState`. Questa proprietà è impostata su una stringa con i seguenti valori possibili:

- "UNINITIALIZED" - L'oggetto `updater` non è stato inizializzato.
- "INITIALIZING" - L'oggetto `updater` è in corso di inizializzazione.
- "READY" - L'oggetto `updater` è stato inizializzato.
- "BEFORE_CHECKING" - L'oggetto `updater` non ha ancora effettuato la verifica del file descrittore dell'aggiornamento.
- "CHECKING" - L'oggetto `updater` sta effettuando la verifica del file descrittore dell'applicazione.
- "AVAILABLE" - Il file descrittore di `updater` è disponibile.
- "DOWNLOADING" - L'oggetto `updater` sta scaricando il file AIR.
- "DOWNLOADED" - L'oggetto `updater` ha scaricato il file AIR.
- "INSTALLING" - L'oggetto `updater` sta installando il file AIR.
- "PENDING_INSTALLING" - L'oggetto `updater` ha effettuato l'inizializzazione e vi sono aggiornamenti in sospeso.

Alcuni metodi dell'oggetto `updater` vengono eseguiti solo in presenza di un determinato stato dell'oggetto stesso.

Inizializzazione del framework di aggiornamento

Dopo l'impostazione delle proprietà di configurazione (consultate ["Esempio di base: Uso della versione ApplicationUpdaterUI"](#) a pagina 276), chiamate il metodo `initialize()` per inizializzare l'aggiornamento:

```
appUpdater.initialize();
```

Questo metodo esegue le seguenti operazioni:

- Inizializza il framework di aggiornamento, installa automaticamente in modo sincrono gli aggiornamenti in sospeso. È necessario chiamare questo metodo durante l'avvio dell'applicazione, in quanto potrebbe riavviare l'applicazione quando viene chiamato.
- Verifica se è presente un aggiornamento posticipato e lo installa.
- Se durante il processo di aggiornamento si verifica un errore, cancella il file di aggiornamento e le informazioni sulla versione dall'area di memorizzazione dell'applicazione.
- Se l'intervallo è scaduto, avvia il processo di aggiornamento. In caso contrario riavvia il timer.

La chiamata a questo metodo può attivare l'invio dei seguenti eventi da parte dell'oggetto `updater`:

- `UpdateEvent.INITIALIZED` - Inviato quando l'inizializzazione è completa.
- `ErrorEvent.ERROR` - Inviato quando si verifica un errore durante l'inizializzazione.

All'invio dell'evento `UpdateEvent.INITIALIZED`, il processo di aggiornamento viene completato.

Quando chiamate il metodo `initialize()`, l'oggetto `updater` avvia il processo di aggiornamento e completa tutti i passaggi in base all'impostazione del ritardo specificato per il timer. Potete comunque avviare il processo di aggiornamento in qualsiasi momento, chiamando il metodo `checkNow()` dell'oggetto `updater`:

```
appUpdater.checkNow();
```

Questo metodo non attiva alcuna operazione se il processo di aggiornamento è già in esecuzione. Altrimenti, avvia il processo di aggiornamento.

L'oggetto `updater` può inviare il seguente evento a seguito della chiamata del metodo `checkNow()`:

- `UpdateEvent.CHECK_FOR_UPDATE`, subito prima del tentativo di scaricare il file descrittore dell'aggiornamento.

Se annullate l'evento `checkForUpdate`, potete chiamare il metodo `checkForUpdate()` dell'oggetto `updater`. (Vedete la sezione successiva.) Se non annullate l'evento, il processo di aggiornamento continua con la verifica del file descrittore dell'aggiornamento.

Gestione del processo di aggiornamento nella versione `ApplicationUpdaterUI`

Nella versione `ApplicationUpdaterUI`, l'utente può annullare il processo tramite i pulsanti `Annulla` presenti nelle finestre di dialogo dell'interfaccia utente. Potete inoltre annullare il processo di aggiornamento a livello di codice, chiamando il metodo `cancelUpdate()` dell'oggetto `ApplicationUpdaterUI`.

Potete impostare le proprietà dell'oggetto `ApplicationUpdaterUI` o definire gli elementi nel file di configurazione dell'aggiornamento per specificare quali conferme vengono visualizzate nelle finestre di dialogo dall'oggetto `updater`. Per ulteriori dettagli, vedete [“Configurazione delle impostazioni di aggiornamento”](#) a pagina 280.

Gestione del processo di aggiornamento nella versione `ApplicationUpdater`

Potete chiamare il metodo `preventDefault()` degli oggetti evento inviati dall'oggetto `ApplicationUpdater` per annullare i passaggi del processo di aggiornamento (vedete [“Processo di aggiornamento”](#) a pagina 282).

L'annullamento del comportamento predefinito consente all'applicazione di visualizzare un messaggio per chiedere all'utente se desidera continuare.

Nelle seguenti sezioni viene descritto come continuare il processo di aggiornamento quando viene annullato un passaggio del processo.

Scaricamento e interpretazione del file descrittore dell'aggiornamento

L'oggetto `ApplicationUpdater` invia l'evento `checkForUpdate` prima dell'inizio del processo di aggiornamento, subito prima che l'oggetto `updater` tenti di scaricare il file descrittore dell'aggiornamento. Se annullate il comportamento predefinito dell'evento `checkForUpdate`, l'oggetto `updater` non scarica il file descrittore dell'aggiornamento. Per riprendere il processo di aggiornamento, potete chiamare il metodo `checkForUpdate()`:

```
appUpdater.checkForUpdate();
```

Se viene chiamato il metodo `checkForUpdate()`, l'oggetto `updater` scarica in modo asincrono e interpreta il file descrittore dell'aggiornamento. A seguito della chiamata del metodo `checkForUpdate()`, l'oggetto `updater` può inviare i seguenti eventi:

- `StatusUpdateEvent.UPDATE_STATUS` - L'oggetto `updater` ha scaricato e interpretato correttamente il file descrittore dell'aggiornamento. Questo evento presenta le seguenti proprietà:
 - `available` - Un valore booleano. Impostato su `true` se è disponibile una versione diversa rispetto a quella dell'applicazione corrente; in caso contrario `false` (la versione è la stessa).
 - `version` - Un oggetto `String`. La versione rilevata dal file descrittore dell'aggiornamento del file di aggiornamento.
 - `details` - Un oggetto `Array`. Se non sono presenti versioni localizzate della descrizione, questo oggetto array restituisce una stringa vuota (" ") come primo elemento e la descrizione come secondo elemento.

Se sono presenti più versioni della descrizione (nel file descrittore dell'aggiornamento), l'array contiene più array secondari. Ogni array presenta due elementi: il primo è un codice di lingua (ad esempio "en"), mentre il secondo è la descrizione corrispondente (una stringa) per quella lingua. Consultate [“Definizione dei file descrittore dell'aggiornamento e aggiunta del file AIR al server Web”](#) a pagina 278.

- `StatusUpdateErrorEvent.UPDATE_ERROR` - Si è verificato un errore e l'oggetto `updater` non ha potuto scaricare o interpretare il file descrittore dell'aggiornamento.

Scaricamento del file AIR di aggiornamento

L'oggetto `ApplicationUpdater` invia l'evento `updateStatus` dopo che il file descrittore dell'aggiornamento è stato scaricato e interpretato correttamente. Il comportamento predefinito prevede di avviare lo scaricamento del file di aggiornamento, se è disponibile. Se annullate il comportamento predefinito, potete chiamare il metodo `downloadUpdate()` per riprendere il processo di aggiornamento:

```
appUpdater.downloadUpdate();
```

Se viene chiamato questo metodo, l'oggetto `updater` scarica in modo asincrono la versione di aggiornamento del file AIR.

Il metodo `downloadUpdate()` può inviare i seguenti eventi:

- `UpdateEvent.DOWNLOAD_START` - La connessione al server è stata stabilita. Quando usate la libreria di `ApplicationUpdaterUI`, questo evento visualizza una finestra di dialogo con una barra di avanzamento che traccia lo stato dello scaricamento.
- `ProgressEvent.PROGRESS` - Inviato periodicamente mentre avanza lo scaricamento del file.
- `DownloadErrorEvent.DOWNLOAD_ERROR` - Inviato se si verifica un errore durante la connessione o lo scaricamento del file di aggiornamento. Viene inviato anche in caso di stati HTTP non validi (ad esempio "404 - File non trovato"). Questo evento presenta una proprietà `errorID`, un valore intero che definisce le informazioni aggiuntive sull'errore. Una proprietà `subErrorID` aggiuntiva può contenere ulteriori informazioni sull'errore.
- `UpdateEvent.DOWNLOAD_COMPLETE` - L'oggetto `updater` ha scaricato e interpretato correttamente il file descrittore dell'aggiornamento. Se non annullate questo evento, la versione `ApplicationUpdater` continua con l'installazione della versione di aggiornamento. Nella versione `ApplicationUpdaterUI`, viene visualizzata all'utente una finestra di dialogo con l'opzione per continuare.

Aggiornamento dell'applicazione

L'oggetto `ApplicationUpdater` invia l'evento `downloadComplete` dopo avere completato lo scaricamento del file di aggiornamento. Se annullate il comportamento predefinito, potete chiamare il metodo `installUpdate()` per riprendere il processo di aggiornamento:

```
appUpdater.installUpdate(file);
```

Se viene chiamato questo metodo, l'oggetto `updater` installa una versione di aggiornamento del file AIR. Il metodo include il parametro `file`; un oggetto `File` che fa riferimento al file AIR da utilizzare come aggiornamento.

L'oggetto `ApplicationUpdater` può inviare l'evento `beforeInstall` a seguito della chiamata al metodo `installUpdate()`:

- `UpdateEvent.BEFORE_INSTALL` - Inviato subito prima dell'installazione dell'aggiornamento. A volte, può essere utile impedire l'installazione dell'aggiornamento a questo punto, in modo che l'utente possa completare il lavoro in corso prima di procedere all'aggiornamento. la chiamata al metodo `preventDefault()` dell'oggetto `Event` posticipa l'installazione fino al riavvio successivo; non possono essere avviati altri processi di aggiornamento. (Sono inclusi gli aggiornamenti che dovessero risultare della chiamata al metodo `checkNow()` o dal controllo periodico).

Installazione da un file AIR arbitrario

Potete chiamare il metodo `installFromAIRFile()` per installare la versione di aggiornamento da un file AIR sul computer dell'utente:

```
appUpdater.installFromAIRFile();
```

Con questo metodo, l'oggetto `updater` installa una versione di aggiornamento dell'applicazione dal file AIR.

Il metodo `installFromAIRFile()` può inviare i seguenti eventi:

- `StatusFileUpdateEvent.FILE_UPDATE_STATUS` - Inviato dopo che `ApplicationUpdater` ha completato la convalida del file inviato usando il metodo `installFromAIRFile()`. Questo evento presenta le seguenti proprietà:
 - `available` - Impostato su `true` se è disponibile una versione diversa rispetto a quella dell'applicazione corrente; in caso contrario `false` (le versioni sono le stesse).
 - `version` - La stringa che rappresenta la nuova versione disponibile.
 - `path` - Rappresenta il percorso nativo del file di aggiornamento.

Potete annullare questo evento se la proprietà `available` dell'oggetto `StatusFileUpdateEvent` è impostata su `true`. L'annullamento dell'evento comporta l'interruzione dell'aggiornamento. Per continuare l'aggiornamento annullato, chiamate il metodo `installUpdate()`.

- `StatusFileUpdateErrorEvent.FILE_UPDATE_ERROR` - Si è verificato un errore e l'oggetto `updater` non ha potuto installare l'applicazione AIR.

Annullamento del processo di aggiornamento

Per annullare il processo di aggiornamento, potete chiamare il metodo `cancelUpdate()`:

```
appUpdater.cancelUpdate();
```

Questo metodo annulla le eventuali operazioni di scaricamento in sospeso, eliminando i file scaricati incompleti, e riavvia il timer di controllo periodico.

Questo metodo non attiva alcuna operazione se l'oggetto `updater` è in corso di inizializzazione.

Localizzazione dell'interfaccia `ApplicationUpdaterUI`

La classe `ApplicationUpdaterUI` fornisce un'interfaccia utente predefinita per il processo di aggiornamento. Sono incluse le finestre di dialogo che consentono all'utente di avviare o annullare il processo e di eseguire altre azioni correlate.

L'elemento `description` del file descrittore dell'aggiornamento consente di definire la descrizione dell'applicazione in più lingue. Potete usare più elementi `text` che definiscono gli attributi `lang`, come nel seguente esempio:

```
<?xml version="1.0" encoding="utf-8"?>
  <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
    <version>1.1a1</version>
    <url>http://example.com/updates/sample_1.1a1.air</url>
    <description>
      <text xml:lang="en">English description</text>
      <text xml:lang="fr">French description</text>
      <text xml:lang="ro">Romanian description</text>
    </description>
  </update>
```

Il framework di aggiornamento utilizza la descrizione che meglio si adatta alla sequenza di localizzazione dell'utente finale. Per ulteriori informazioni, vedete [Definizione del file descrittore dell'aggiornamento e aggiunta del file AIR al server Web](#).

Gli sviluppatori in Flex possono aggiungere direttamente una nuova lingua al pacchetto `"ApplicationUpdaterDialogs"`.

Gli sviluppatori in JavaScript possono chiamare il metodo `addResources()` dell'oggetto `updater`. Questo metodo aggiunge dinamicamente un nuovo pacchetto di risorse per una lingua. Nel pacchetto di risorse sono definite le stringhe localizzate per una lingua. Queste stringhe vengono usate in diversi campi di testo delle finestre di dialogo.

Gli sviluppatori in JavaScript possono usare la proprietà `localeChain` della classe `ApplicationUpdaterUI` per definire la sequenza di versioni locali usate dall'interfaccia utente. In genere, questa proprietà viene usata solo dagli sviluppatori in JavaScript (HTML). Gli sviluppatori in Flex possono usare `ResourceManager` per gestire la sequenza di versioni locali.

Il seguente codice JavaScript definisce, ad esempio, pacchetti di risorse per le lingue rumeno e ungherese:

```
appUpdater.addResources("ro_RO",
    {titleCheck: "Titlu", msgCheck: "Mesaj", btnCheck: "Buton"});
appUpdater.addResources("hu", {titleCheck: "Cím", msgCheck: "Üzenet"});
var languages = ["ro", "hu"];
languages = languages.concat(air.Capabilities.languages);
var sortedLanguages = air.Localizer.sortLanguagesByPreference(languages,
    air.Capabilities.language,
    "en-US");
sortedLanguages.push("en-US");
appUpdater.localeChain = sortedLanguages;
```

Per ulteriori dettagli, vedete la descrizione del metodo `addResources()` della classe nella Guida di riferimento.

Capitolo 18: Visualizzazione del codice sorgente

Esattamente come è possibile visualizzare il codice sorgente di una pagina HTML in un browser Web, è possibile anche visualizzare il codice sorgente di un'applicazione AIR basata su HTML. Adobe® AIR® SDK include un file JavaScript AIRSourceViewer.js che potete utilizzare nell'applicazione per mostrare con facilità il codice sorgente agli utenti finali.

Caricamento, configurazione e apertura del visualizzatore del codice sorgente

Il codice del visualizzatore del codice sorgente è incluso nel file JavaScript AIRSourceViewer.js, che si trova nella directory frameworks di AIR SDK. Per utilizzare il visualizzatore del codice sorgente nell'applicazione, copiate AIRSourceViewer.js nella directory di progetto dell'applicazione e caricate il file tramite un tag script nel file HTML principale dell'applicazione:

```
<script type="text/javascript" src="AIRSourceViewer.js"></script>
```

Nel file AIRSourceViewer.js viene definita una classe, SourceViewer, a cui potete accedere dal codice JavaScript chiamando `air.SourceViewer`.

La classe SourceViewer definisce tre metodi: `getDefault()`, `setup()` e `viewSource()`.

Metodo	Descrizione
<code>getDefault()</code>	Metodo statico. Restituisce un'istanza di SourceViewer che potete utilizzare per chiamare gli altri metodi.
<code>setup()</code>	Applica le impostazioni di configurazione al visualizzatore del codice sorgente. Per informazioni dettagliate, vedete "Configurazione del visualizzatore del codice sorgente" a pagina 288.
<code>viewSource()</code>	Apri una nuova finestra nella quale l'utente può sfogliare e aprire i file di origine dell'applicazione host.

Nota: il codice per cui viene utilizzato il visualizzatore del codice sorgente deve trovarsi nella sandbox di sicurezza dell'applicazione (in un file nella directory dell'applicazione).

Il codice JavaScript seguente, ad esempio, consente di creare un'istanza di un oggetto SourceViewer e di aprire la finestra del visualizzatore del codice sorgente con l'elenco di tutti i file di origine:

```
var viewer = air.SourceViewer.getDefault();
viewer.viewSource();
```

Configurazione del visualizzatore del codice sorgente

Il metodo `config()` applica le impostazioni al visualizzatore del codice sorgente. Questo metodo accetta un parametro: `configObject`. L'oggetto `configObject` ha proprietà che definiscono le impostazioni di configurazione per il visualizzatore del codice sorgente, ovvero `default`, `exclude`, `initialPosition`, `modal`, `typesToRemove` e `typesToAdd`.

default

Stringa che specifica il percorso relativo del file iniziale da visualizzare nel visualizzatore del codice sorgente.

Il codice JavaScript seguente, ad esempio, consente di aprire la finestra del visualizzatore del codice sorgente con il file `index.html` come file iniziale:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.default = "index.html";
viewer.viewSource(configObj);
```

exclude

Array di stringhe che specifica i file o le directory da escludere dall'elenco del visualizzatore del codice sorgente. I percorsi sono relativi alla directory dell'applicazione. Non sono supportati caratteri jolly.

Il codice JavaScript seguente, ad esempio, consente di aprire la finestra del visualizzatore del codice sorgente con l'elenco di tutti i file di origine, ad eccezione del file `AIRSourceViewer.js` e dei file contenuti nelle sottodirectory `Images` e `Sounds`:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.exclude = ["AIRSourceViewer.js", "Images" "Sounds"];
viewer.viewSource(configObj);
```

initialPosition

Array che include due numeri e specifica le coordinate x e y iniziali della finestra del visualizzatore del codice sorgente.

Il codice JavaScript seguente ad esempio consente di aprire la finestra del visualizzatore del codice sorgente con le coordinate `[40, 60]` (X = 40, Y = 60):

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.initialPosition = [40, 60];
viewer.viewSource(configObj);
```

modal

Valore booleano che specifica se il visualizzatore del codice sorgente deve essere una finestra modale (`true`) o non modale (`false`). Per impostazione predefinita, la finestra del visualizzatore del codice sorgente è modale.

Il codice JavaScript seguente, ad esempio, consente di aprire la finestra del visualizzatore del codice sorgente in modo che l'utente possa interagire sia con essa che con le finestre di altre applicazioni:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.modal = false;
viewer.viewSource(configObj);
```

typesToAdd

Array di stringhe che specifica i tipi di file da includere nell'elenco del visualizzatore del codice sorgente oltre ai tipi predefiniti inclusi.

Per impostazione predefinita, nel visualizzatore del codice sorgente vengono elencati i tipi di file seguenti:

- File di testo: TXT, XML, MXML, HTM, HTML, JS, AS, CSS, INI, BAT, PROPERTIES, CONFIG
- File di immagine: JPG, JPEG, PNG, GIF

Se non viene specificato alcun valore vengono inclusi tutti i tipi predefiniti (ad eccezione di quelli specificati nella proprietà `typesToExclude`).

Con il codice JavaScript seguente, ad esempio, viene aperta la finestra del visualizzatore del codice sorgente che include file VCF e VCARD:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToAdd = ["text.vcf", "text.vcard"];
viewer.viewSource(configObj);
```

Per ogni tipo di file che elencate, dovete specificare "text" (per i file di testo) o "image" per i file di immagine.

typesToExclude

Array di stringhe che specifica i tipi di file da escludere dal visualizzatore del codice sorgente.

Per impostazione predefinita, nel visualizzatore del codice sorgente vengono elencati i tipi di file seguenti:

- File di testo: TXT, XML, MXML, HTM, HTML, JS, AS, CSS, INI, BAT, PROPERTIES, CONFIG
- File di immagine: JPG, JPEG, PNG, GIF

Con il codice JavaScript seguente, ad esempio, viene aperta la finestra del visualizzatore del codice sorgente senza elencare file GIF o XML:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToExclude = ["image.gif", "text.xml"];
viewer.viewSource(configObj);
```

Per ogni tipo di file che elencate, dovete specificare "text" (per i file di testo) o "image" per i file di immagine.

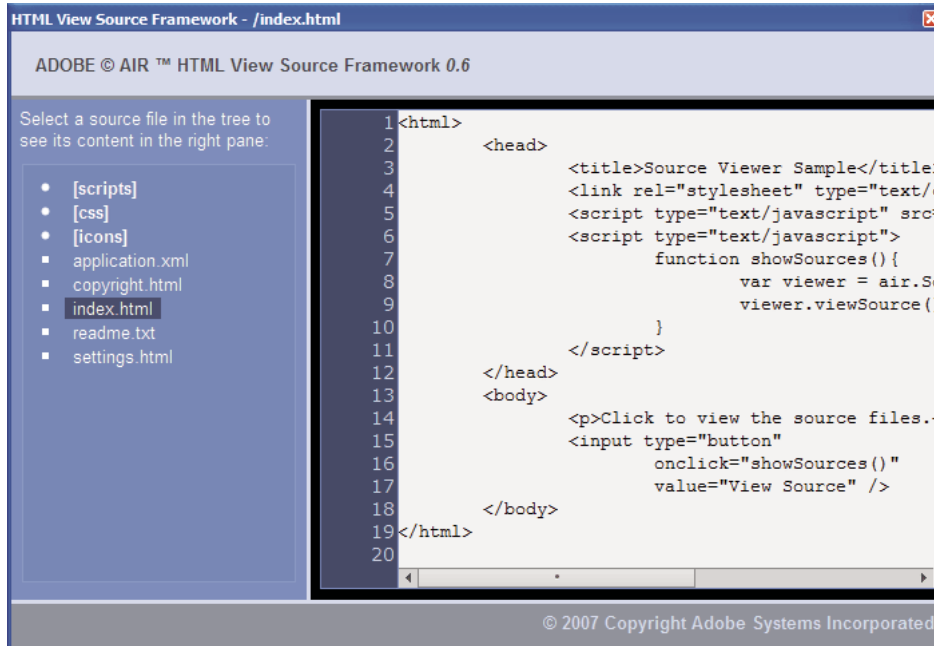
Apertura del visualizzatore del codice sorgente

È consigliabile includere un elemento di interfaccia utente, ad esempio un collegamento, un pulsante o un comando di menu, che, quando selezionato dall'utente, consente di richiamare il codice del visualizzatore del codice sorgente. Con l'applicazione di esempio seguente, ad esempio, viene aperto il visualizzatore del codice sorgente quando l'utente fa clic su un collegamento:

```
<html>
  <head>
    <title>Source Viewer Sample</title>
    <script type="text/javascript" src="AIRSourceViewer.js"></script>
    <script type="text/javascript">
      function showSources(){
        var viewer = air.SourceViewer.getDefault();
        viewer.viewSource()
      }
    </script>
  </head>
  <body>
    <p>Click to view the source files.</p>
    <input type="button"
      onclick="showSources()"
      value="View Source" />
  </body>
</html>
```

Interfaccia utente del visualizzatore del codice sorgente

Quando l'applicazione chiama il metodo `viewSource()` di un oggetto `SourceViewer`, viene aperta la finestra del visualizzatore del codice sorgente. La finestra include un elenco di file e directory di origine (a sinistra) e un'area di visualizzazione che mostra il codice sorgente del file selezionato (sulla destra):



Le directory sono elencate tra parentesi quadre. L'utente può fare clic su una parentesi graffa per espandere o comprimere l'elenco di una directory.

Nel visualizzatore del codice sorgente è possibile visualizzare il codice sorgente di file di testo con estensioni riconosciute (ad esempio, HTML, JS, TXT, XML e altre) o di file di immagine con estensioni riconosciute per le immagini (JPG, JPEG, PNG e GIF). Se un utente seleziona un file che non presenta un'estensione riconosciuta, viene visualizzato un messaggio di errore: `Cannot retrieve text content from this filetype (Impossibile recuperare contenuto di testo da questo tipo di file)`.

Tutti i file di origine esclusi tramite il metodo `setup()` non vengono elencati (vedete [“Caricamento, configurazione e apertura del visualizzatore del codice sorgente”](#) a pagina 288).

Capitolo 19: Debug con AIR HTML Introspector

Adobe® AIR® SDK include il file JavaScript AIRIntrospector.js che è possibile includere nell'applicazione per agevolare il debug di applicazioni basate su HTML.

Informazioni su AIR Introspector

Adobe AIR HTML Introspector/Introspector per applicazioni JavaScript (detto AIR HTML Introspector) offre utili funzionalità per lo sviluppo e il debug di applicazioni basate su HTML:

- Include uno strumento di introspezione che consente di puntare a un elemento dell'interfaccia utente nell'applicazione e visualizzarne i tag e le proprietà DOM e
- una console per l'invio dei riferimenti degli oggetti per l'introspezione. Potete modificare i valori delle proprietà ed eseguire codice JavaScript, nonché serializzare oggetti nella console, riducendo così le operazioni di modifica dei dati. Potete inoltre copiare e salvare testo dalla console.
- Include una visualizzazione struttura per le proprietà e le funzioni DOM.
- Consente di modificare gli attributi e i nodi di testo per gli elementi DOM.
- Elenca collegamenti, stili CSS e file JavaScript caricati nell'applicazione.
- Consente di visualizzare il codice sorgente HTML iniziale e i tag correnti per l'interfaccia utente.
- Consente di accedere ai file nella directory dell'applicazione. Questa funzionalità è disponibile solo per la console di AIR HTML Introspector aperta per la sandbox dell'applicazione. Non disponibile per le console aperte per contenuto non di sandbox dell'applicazione.
- Include un visualizzatore per gli oggetti XMLHttpRequest e le relative proprietà, tra cui le proprietà `responseText` e `responseXML` (se disponibili).
- Potete cercare testo corrispondente nel codice sorgente e nei file di origine.

Caricamento del codice di AIR Introspector

Il codice di AIR Introspector è incluso in un file JavaScript, AIRIntrospector.js, nella directory frameworks di AIR SDK. Per utilizzare AIR Introspector nell'applicazione, copiate AIRIntrospector.js nella directory di progetto dell'applicazione e caricate il file tramite un tag script nel file HTML principale dell'applicazione:

```
<script type="text/javascript" src="AIRIntrospector.js"></script>
```

Includete il file anche in ogni file HTML che corrisponde alle diverse finestre native dell'applicazione.

Importante: includete il file AIRIntrospector.js solo per lo sviluppo e il debug dell'applicazione. Rimuovetelo dal pacchetto dell'applicazione AIR che distribuite.

Nel file AIRIntrospector.js viene definita una classe, Console, a cui potete accedere da codice JavaScript chiamando `air.Introspector.Console`.

Nota: il codice per cui viene utilizzato AIR Introspector deve trovarsi nella sandbox di sicurezza dell'applicazione (in un file nella directory dell'applicazione).

Analisi di un oggetto nella scheda Console

La classe Console definisce cinque metodi: `log()`, `warn()`, `info()`, `error()` e `dump()`.

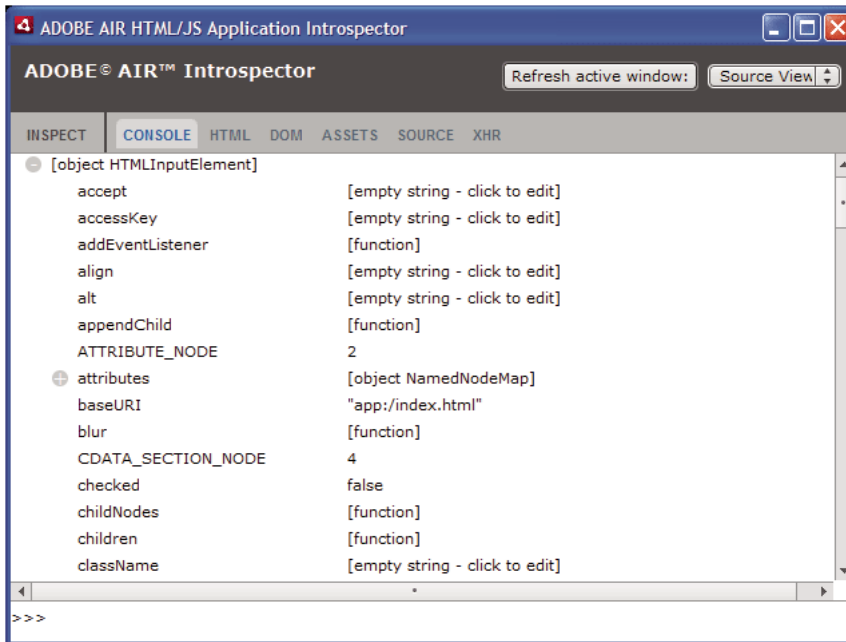
I metodi `log()`, `warn()`, `info()` e `error()` permettono di inviare un oggetto alla scheda Console. Il metodo più semplice tra questi è `log()`. Il codice seguente consente di inviare un oggetto semplice, rappresentato dalla variabile `test`, alla scheda Console:

```
var test = "hello";
air.Introspector.Console.log(test);
```

È tuttavia più utile inviare un oggetto complesso alla scheda Console. La pagina HTML seguente ad esempio include un pulsante (`btn1`) che chiama una funzione per inviare l'oggetto pulsante stesso alla scheda Console:

```
<html>
  <head>
    <title>Source Viewer Sample</title>
    <script type="text/javascript" src="scripts/AIRIntrospector.js"></script>
    <script type="text/javascript">
      function logBtn()
      {
        var button1 = document.getElementById("btn1");
        air.Introspector.Console.log(button1);
      }
    </script>
  </head>
  <body>
    <p>Click to view the button object in the Console.</p>
    <input type="button" id="btn1"
      onclick="logBtn()"
      value="Log" />
  </body>
</html>
```

Quando fate clic sul pulsante, nella scheda Console viene visualizzato l'oggetto btn1 e potete espandere la visualizzazione struttura dell'oggetto per ispezionarne le proprietà:



Potete modificare una proprietà dell'oggetto facendo clic sul listato a destra del nome della proprietà e modificando il testo del listato.

I metodi `info()`, `error()` e `warn()` sono analoghi al metodo `log()`. Quando chiamate questi metodi, tuttavia, all'inizio della riga nella Console viene visualizzata un'icona:

Metodo	Icona
<code>info()</code>	
<code>error()</code>	
<code>warn()</code>	

I metodi `log()`, `warn()`, `info()` ed `error()` consentono di inviare un riferimento solo a un oggetto effettivo, pertanto le proprietà disponibili sono quelle presenti al momento della visualizzazione. Se desiderate serializzare l'oggetto effettivo, utilizzate il metodo `dump()`. Il metodo presenta due parametri:

Parametro	Descrizione
<code>dumpObject</code>	L'oggetto da serializzare.
<code>levels</code>	Il numero massimo di livelli da esaminare nella struttura dell'oggetto (in aggiunta al livello principale). Il valore predefinito è 1 (che indica che viene mostrato un livello sotto quello principale). Questo parametro è opzionale.

La chiamata del metodo `dump()` consente di serializzare un oggetto prima di inviarlo alla scheda Console affinché non sia possibile modificarne le proprietà. Consideriamo l'esempio del codice seguente:

```
var testObject = new Object();  
testObject.foo = "foo";  
testObject.bar = 234;  
air.Introspector.Console.dump(testObject);
```

Quando eseguite questo codice, nella Console viene visualizzato l'oggetto `testObject` con le relative proprietà, ma non potete modificare i valori delle proprietà nella Console.

Configurazione di AIR Introspector

Potete configurare la console impostando le proprietà della variabile `AIRIntrospectorConfig` globale. Con il codice JavaScript seguente ad esempio viene configurato AIR Introspector per mandare a capo le colonne al centesimo carattere:

```
var AIRIntrospectorConfig = new Object();  
AIRIntrospectorConfig.wrapColumns = 100;
```

Ricordate di impostare le proprietà della variabile `AIRIntrospectorConfig` prima di caricare il file `AIRIntrospector.js` (tramite un tag `script`).

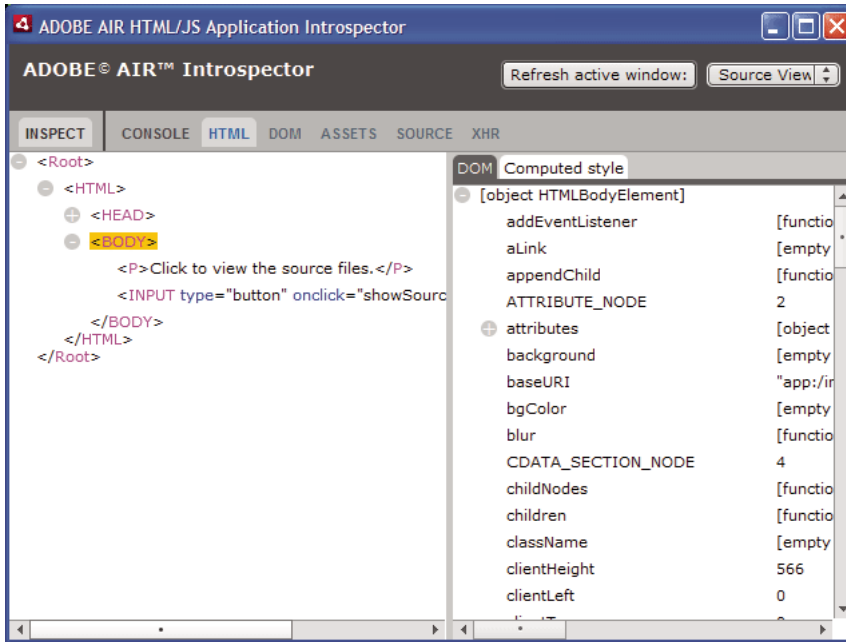
La variabile `AIRIntrospectorConfig` dispone di otto proprietà:

Proprietà	Valore predefinito	Descrizione
<code>closeIntrospectorOnExit</code>	<code>true</code>	Imposta la finestra di Inspector per la chiusura quando tutte le altre finestre dell'applicazione sono chiuse.
<code>debuggerKey</code>	123 (il tasto F12)	Il codice del tasto della scelta rapida da tastiera per mostrare e nascondere la finestra di AIR Introspector.
<code>debugRuntimeObjects</code>	<code>true</code>	Imposta Introspector per espandere gli oggetti in runtime oltre agli oggetti definiti in JavaScript.
<code>flashTabLabels</code>	<code>true</code>	Imposta le schede Console e XMLHttpRequest affinché lampeggino per indicare eventuali modifiche al loro interno (ad esempio quando nelle schede viene registrato testo).
<code>introspectorKey</code>	122 (il tasto F11)	Il codice del tasto della scelta rapida da tastiera per aprire il pannello di analisi.
<code>showTimestamp</code>	<code>true</code>	Imposta la scheda Console affinché visualizzi indicatori di data e ora all'inizio di ogni riga.
<code>showSender</code>	<code>true</code>	Imposta la scheda Console affinché visualizzi informazioni sull'oggetto che invia il messaggio all'inizio di ogni riga.
<code>wrapColumns</code>	2000	Numero di colonne in corrispondenza del quale il testo dei file di origine viene mandato a capo.

Interfaccia di AIR Introspector

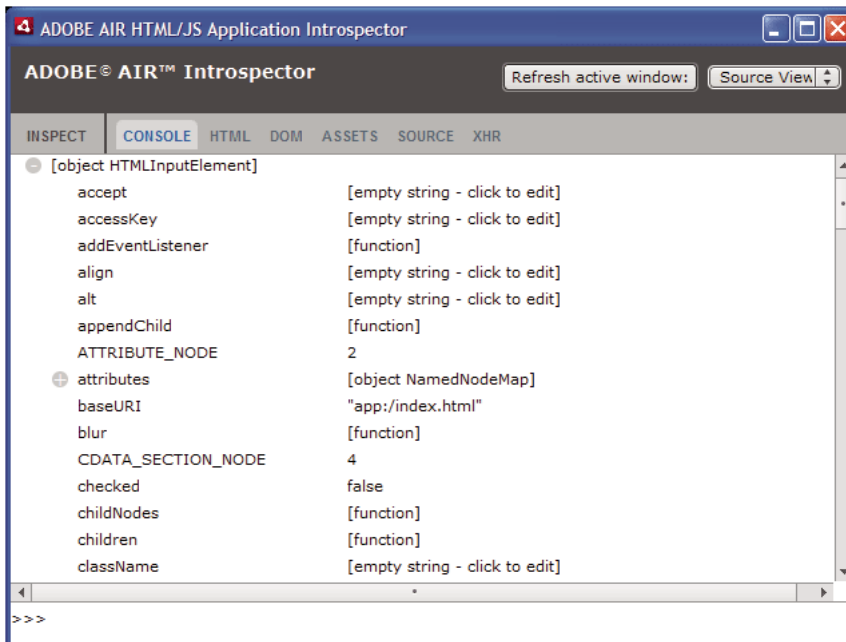
Per aprire la finestra di AIR Introspector durante il debug dell'applicazione, premete il tasto F12 o chiamate uno dei metodi della classe `Console` (vedete [“Analisi di un oggetto nella scheda Console”](#) a pagina 293). Potete configurare un tasto di scelta rapida diverso dal tasto F12. Vedete [“Configurazione di AIR Introspector”](#) a pagina 295.

La finestra di AIR Introspector comprende sei schede: Console, HTML, DOM, Assets (Risorse), Source (Codice sorgente) e XHR come illustrato nella figura seguente:



Scheda Console

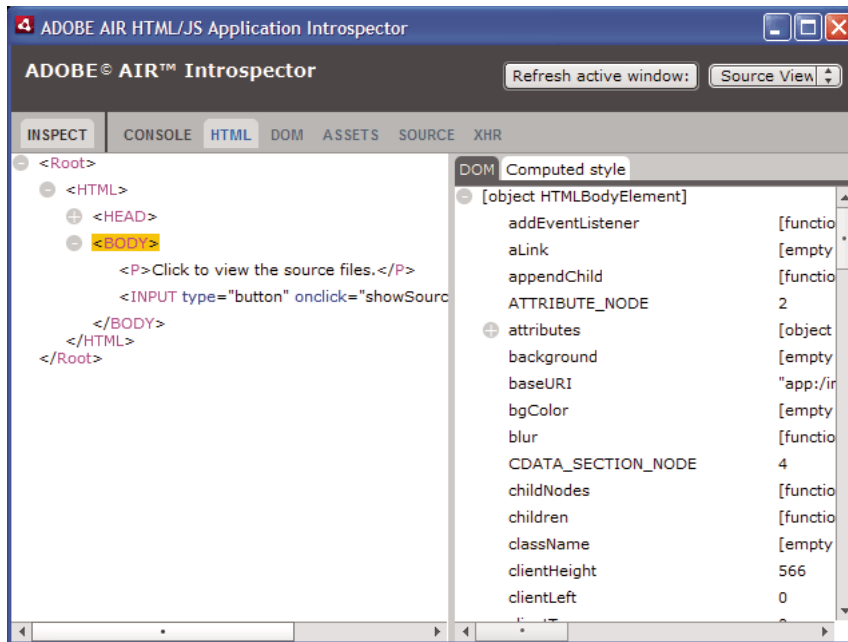
Nella scheda Console sono visualizzati i valori delle proprietà passate come parametri a uno dei metodi della classe `air.Introspector.Console`. Per informazioni dettagliate, vedete “[Analisi di un oggetto nella scheda Console](#)” a pagina 293.



- Per cancellare la console, fate clic con il pulsante destro del mouse sul testo e selezionate Clear Console (Cancella console).
- Per salvare testo della scheda Console in un file, fate clic con il pulsante destro del mouse sulla scheda Console e scegliete Save Console To File (Salva console in file).
- Per salvare testo della scheda Console negli Appunti, fate clic con il pulsante destro del mouse sulla scheda Console e scegliete Save Console To Clipboard (Salva console negli Appunti). Per copiare solo il testo selezionato negli Appunti, fate clic con il pulsante destro del mouse sul testo e scegliete Copia.
- Per salvare testo della classe Console in un file, fate clic con il pulsante destro del mouse sulla scheda Console e selezionate Save Console To File (Salva console in file).
- Per cercare nella scheda testo corrispondente, premete CTRL+F in Windows o Comando+F in Mac OS. La ricerca non viene eseguita nei nodi della struttura non visibili.

Scheda HTML

La scheda HTML consente di visualizzare l'intero HTML DOM in una struttura. Fate clic su un elemento per visualizzarne le proprietà nella parte destra della scheda. Fate clic sulle icone + e - per espandere e comprimere un nodo della struttura.



Potete modificare qualsiasi attributo o elemento di testo nella scheda HTML. Il valore modificato verrà riflesso nell'applicazione.

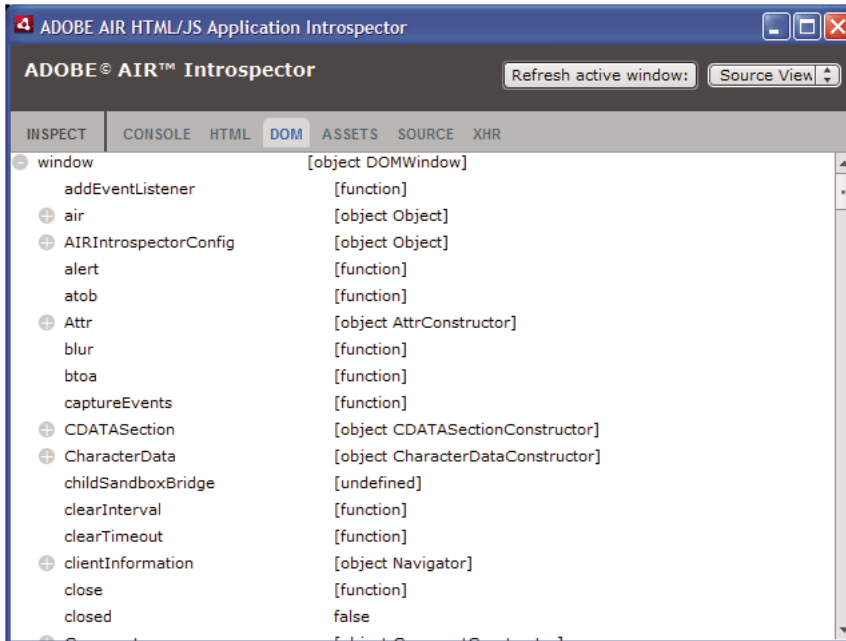
Fate clic sul pulsante Inspect (Analizza) a sinistra dell'elenco di schede nella finestra di AIR Introspector. Potete fare clic su qualsiasi elemento nella pagina HTML della finestra principale. L'oggetto DOM associato verrà visualizzato nella scheda HTML. Quando lo stato attivo si trova nella finestra principale potete inoltre premere la scelta rapida da tastiera per attivare e disattivare il pulsante Inspect (Analizza). La scelta rapida da tastiera per impostazione predefinita è F11. Potete configurare un tasto di scelta rapida diverso dal tasto F11. Vedete "[Configurazione di AIR Introspector](#)" a pagina 295.

Fate clic sul pulsante Refresh Active Window (Aggiorna finestra attiva) nella parte superiore della finestra di AIR Introspector per aggiornare i dati visualizzati nella scheda HTML.

Premete CTRL+F in Windows o Comando+F in Mac OS per cercare il testo corrispondente nella scheda. La ricerca non viene eseguita nei nodi della struttura non visibili.

Scheda DOM

Nella scheda DOM è mostrato l'oggetto window in una struttura. Potete modificare qualsiasi stringa e proprietà numerica e il valore modificato verrà riflesso nell'applicazione.

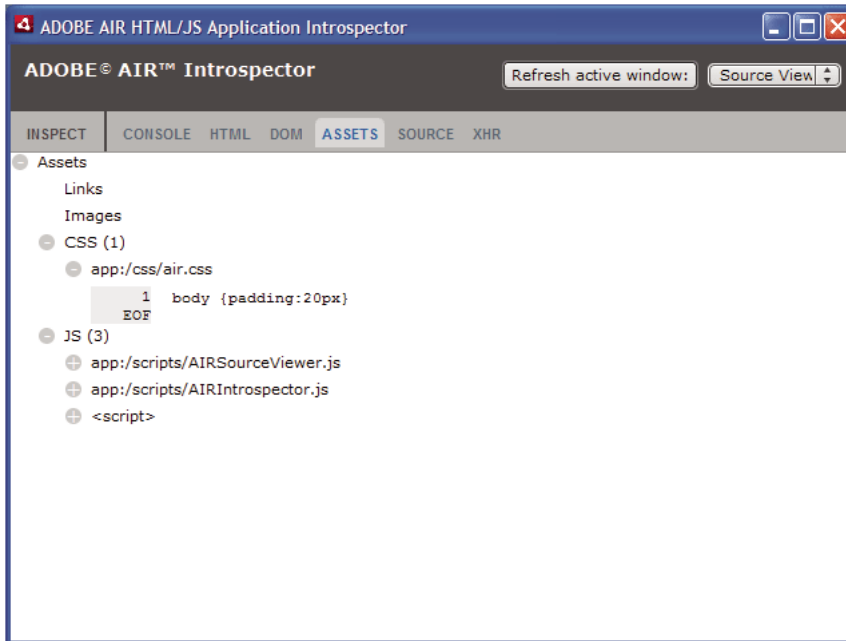


Fate clic sul pulsante Refresh Active Window (Aggiorna finestra attiva) nella parte superiore della finestra di AIR Introspector per aggiornare i dati visualizzati nella scheda DOM.

Premete CTRL+F in Windows o Comando+F in Mac OS per cercare il testo corrispondente nella scheda. La ricerca non viene eseguita nei nodi della struttura non visibili.

Scheda Assets (Risorse)

Nella scheda Assets (Risorse) potete fare clic sui collegamenti, le immagini, i file CSS e JavaScript caricati nella finestra nativa. Espandendo uno di questi nodi viene mostrato il contenuto del file o l'immagine effettiva in uso.



Fate clic sul pulsante Refresh Active Window (Aggiorna finestra attiva) nella parte superiore della finestra di AIR Introspector per aggiornare i dati visualizzati nella scheda Assets (Risorse).

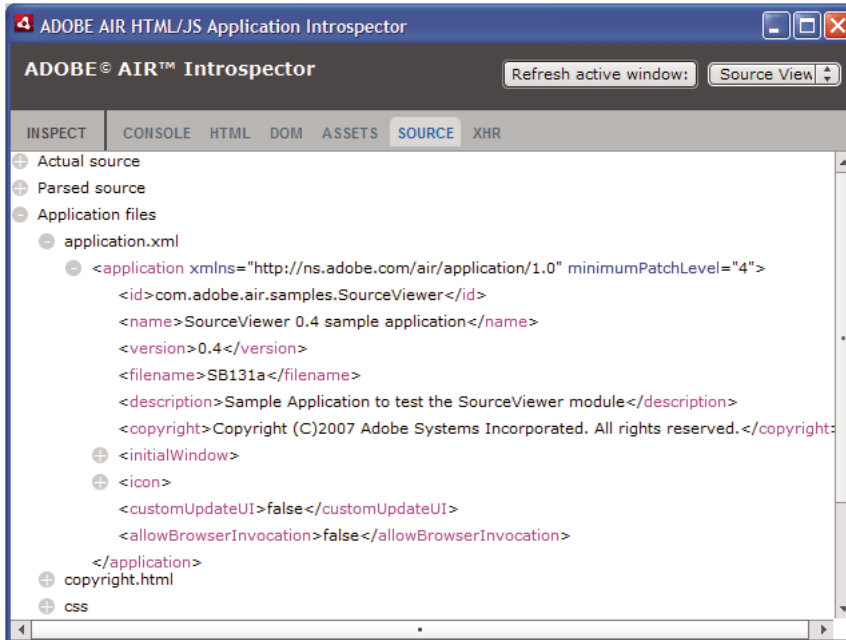
Premete CTRL+F in Windows o Comando+F in Mac OS per cercare il testo corrispondente nella scheda. La ricerca non viene eseguita nei nodi della struttura non visibili.

Scheda Source (Codice sorgente)

La scheda Source (Codice sorgente) comprende tre sezioni:

- Actual source (Codice sorgente effettivo): contiene il codice sorgente HTML della pagina caricata come contenuto principale all'avvio dell'applicazione.
- Parsed source (Codice sorgente analizzato): contiene il codice corrente che definisce l'interfaccia utente dell'applicazione che può essere diverso dal codice sorgente effettivo, perché l'applicazione genera codice in tempo reale mediante tecniche Ajax.

- Application files (File applicazione): contiene un elenco dei file presenti nella directory dell'applicazione. Questo elenco è disponibile per AIR Introspector solo se questo viene avviato dal contenuto nella sandbox di sicurezza dell'applicazione. In questa sezione potete visualizzare il contenuto dei file di testo o le immagini.

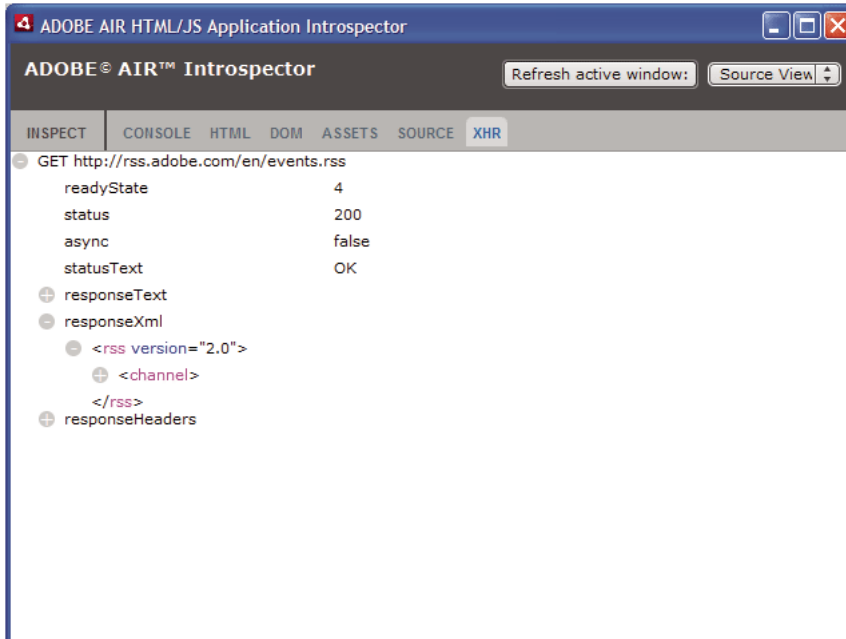


Fate clic sul pulsante Refresh Active Window (Aggiorna finestra attiva) nella parte superiore della finestra di AIR Introspector per aggiornare i dati visualizzati nella scheda Source (Codice sorgente).

Premete CTRL+F in Windows o Comando+F in Mac OS per cercare il testo corrispondente nella scheda. La ricerca non viene eseguita nei nodi della struttura non visibili.

Scheda XHR

Nella scheda XHR viene intercettata la comunicazione XMLHttpRequest dell'applicazione e vengono registrate le informazioni. Potete pertanto visualizzare le proprietà XMLHttpRequest tra cui `responseText` e `responseXML` (se disponibili) in una struttura.



Premete CTRL+F in Windows o Comando+F in Mac OS per cercare il testo corrispondente nella scheda. La ricerca non viene eseguita nei nodi della struttura non visibili.

Uso di AIR Introspector con contenuto in una sandbox non dell'applicazione

Potete caricare contenuto dalla directory dell'applicazione in un iframe o frame mappato a una sandbox non dell'applicazione. Vedete [Sicurezza HTML in Adobe AIR](#) (sviluppatori ActionScript) o [HTML security in Adobe AIR](#) (sviluppatori HTML). Potete utilizzare AIR Introspector con tale contenuto tenendo presenti le regole seguenti:

- Il file AIRIntrospector.js deve essere incluso nel contenuto sia nella sandbox dell'applicazione che nella sandbox non dell'applicazione (iframe).
- Non sovrascrivete la proprietà `parentSandboxBridge` in quanto è utilizzata dal codice di AIR Introspector. Aggiungete tutte le proprietà necessarie. Anziché scrivere quanto segue:

```
parentSandboxBridge = mytrace: function(str) {runtime.trace(str)} ;
```

Utilizzate una sintassi analoga alla seguente:

```
parentSandboxBridge.mytrace = function(str) {runtime.trace(str)};
```

- Dal contenuto della sandbox non dell'applicazione non potete aprire AIR Introspector premendo il tasto F12 o chiamando uno dei metodi della classe `air.Introspector.Console`. Potete aprire la finestra di Introspector solo facendo clic sul pulsante Open Introspector (Apri Introspector). Il pulsante viene aggiunto per impostazione predefinita nell'angolo superiore destro dell'iframe o frame. A causa di limitazioni di sicurezza imposte al contenuto di sandbox non di applicazione, una nuova finestra può essere aperta solo come conseguenza di un'azione dell'utente, ad esempio la scelta di un pulsante.
- Potete aprire finestre di AIR Introspector distinte per la sandbox dell'applicazione e per quella non dell'applicazione. Potete distinguere tra le due utilizzando il titolo visualizzato nelle finestre di AIR Introspector.
- Nella scheda Source (Codice sorgente) non sono visualizzati i file dell'applicazione se AIR Introspector viene eseguito da una sandbox non dell'applicazione.
- Con AIR Introspector è possibile analizzare solo codice nella sandbox da cui è il programma è stato aperto.

Capitolo 20: Localizzazione di applicazioni AIR

Adobe AIR 1.1 e versioni successive

Adobe® AIR® dispone di supporto multilingue.

Per una panoramica sulla localizzazione di contenuto in ActionScript 3.0 e nel framework di Flex, vedete la sezione relativa alla localizzazione di applicazioni nella Guida per gli sviluppatori di ActionScript 3.0.

Lingue supportate in AIR

Il supporto per la localizzazione per le applicazioni AIR nelle lingue seguenti è stato introdotto in AIR 1.1:

- Cinese semplificato
- Cinese tradizionale
- Francese
- Tedesco
- Italiano
- Giapponese
- Coreano
- Portoghese brasiliano
- Russo
- Spagnolo

In AIR 1.5 sono state aggiunte le lingue seguenti:

- Ceco
- Olandese
- Polacco
- Svedese
- Turco

Altri argomenti presenti nell' Aiuto

[Building multilingual Flex applications on Adobe AIR \(Creazione di applicazioni Flex multilingue in Adobe AIR\)](#)

[Building a multilingual HTML-based application \(Creazione di un'applicazione HTML multilingue\)](#)

Localizzazione del nome e della descrizione dell'applicazione nel programma di installazione dell'applicazione AIR

Adobe AIR 1.1 e versioni successive

È possibile specificare più lingue per gli elementi `name` e `description` nel file descrittore dell'applicazione. Ad esempio, l'elemento seguente specifica il nome dell'applicazione in tre lingue (inglese, francese e tedesco).

```
<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>
```

L'attributo `xml:lang` di ogni elemento di testo specifica un codice lingua, definito dallo standard RFC4646 (<http://www.ietf.org/rfc/rfc4646.txt>).

L'elemento `name` definisce il nome dell'applicazione visualizzato dal programma di installazione dell'applicazione AIR che utilizza il valore localizzato più prossimo alle lingue dell'interfaccia utente definite nelle impostazioni del sistema operativo.

Anche in questo caso, potete specificare più lingue per l'elemento `description` nel file descrittore dell'applicazione. Tale elemento definisce il testo descrittivo visualizzato nel programma di installazione dell'applicazione AIR.

Queste impostazioni si applicano solo alle lingue disponibili nel programma di installazione dell'applicazione AIR. Non definiscono le impostazioni locali disponibili per l'applicazione installata in esecuzione. Le applicazioni AIR possono disporre di interfacce utente in grado di supportare più lingue, comprese e in aggiunta a quelle disponibili per il programma di installazione dell'applicazione AIR.

Per ulteriori informazioni, vedete “[Elementi del descrittore dell'applicazione AIR](#)” a pagina 216.

Altri argomenti presenti nell’Aiuto

[Building multilingual Flex applications on Adobe AIR](#) (Creazione di applicazioni Flex multilingue in Adobe AIR)

[Building a multilingual HTML-based application](#) (Creazione di un'applicazione HTML multilingue)

Localizzazione di contenuto HTML con il framework di localizzazione AIR HTML

Adobe AIR 1.1 e versioni successive

L'AIR 1.1 SDK contiene un framework per la localizzazione HTML. Il file `AIRLocalizer.js` di JavaScript definisce il framework. La directory `frameworks` dell'AIR SDK contiene il file `AIRLocalizer.js`, il quale comprende una classe `air.Localizer` con le funzionalità che supportano l'utente nella creazione di applicazioni che supportano più versioni localizzate.

Caricamento del codice del framework per la localizzazione HTML in AIR

Per utilizzare il framework di localizzazione, copiate il file AIRLocalizer.js nel progetto. Inserirlo quindi nel file HTML principale dell'applicazione utilizzando un tag script:

```
<script src="AIRLocalizer.js" type="text/javascript" charset="utf-8"></script>
```

Il codice JavaScript successivo può chiamare l'oggetto `air.Localizer.localizer`:

```
<script>
    var localizer = air.Localizer.localizer;
</script>
```

L'oggetto `air.Localizer.localizer` è un oggetto singleton che definisce metodi e proprietà per l'utilizzo e la gestione di risorse localizzate. La classe `Localizer` comprende i seguenti metodi:

Metodo	Descrizione
<code>getFile()</code>	Ottiene il testo di un pacchetto di risorse specificato per impostazioni locali specificate. Vedete "Ottenere risorse per impostazioni locali specifiche" a pagina 312.
<code>getLocaleChain()</code>	Restituisce le lingue nella catena di impostazioni locali. Vedete "Definizione della catena di impostazioni locali" a pagina 311.
<code>getResourceBundle()</code>	Restituisce le chiavi del pacchetto e i valori corrispondenti come un oggetto. Vedete "Ottenere risorse per impostazioni locali specifiche" a pagina 312.
<code>getString()</code>	Ottiene la stringa definita per una risorsa. Vedete "Ottenere risorse per impostazioni locali specifiche" a pagina 312.
<code>setBundlesDirectory()</code>	Imposta il percorso della directory dei pacchetti. Vedete "Personalizzazione delle impostazioni di AIR HTML Localizer" a pagina 310.
<code>setLocalAttributePrefix()</code>	Imposta il prefisso utilizzato dagli attributi del localizer impiegati negli elementi HTML DOM. Vedete "Personalizzazione delle impostazioni di AIR HTML Localizer" a pagina 310.
<code>setLocaleChain()</code>	Imposta l'ordine delle lingue nella catena delle impostazioni locali. Vedete "Definizione della catena di impostazioni locali" a pagina 311.
<code>sortLanguagesByPreference()</code>	Ordina le impostazioni locali nella catena in base al relativo ordine nelle impostazioni del sistema operativo. Vedete "Definizione della catena di impostazioni locali" a pagina 311.
<code>update()</code>	Aggiorna l'HTML DOM (o un elemento DOM) con le stringhe localizzate dalla catena di impostazioni locali corrente. Per informazioni sulle catene di impostazioni locali, consultate "Gestione di catene di impostazioni locali" a pagina 308. Per ulteriori informazioni sul metodo <code>update()</code> , vedete "Aggiornamento degli elementi DOM per l'uso dell'impostazione locale corrente" a pagina 309.

La classe `Localizer` comprende le seguenti proprietà statiche:

Proprietà	Descrizione
<code>localizer</code>	Restituisce un riferimento all'oggetto <code>Localizer</code> singleton per l'applicazione.
<code>ultimateFallbackLocale</code>	Le impostazioni locali utilizzate se l'applicazione non supporta le preferenze dell'utente. Vedete "Definizione della catena di impostazioni locali" a pagina 311.

Indicazione delle lingue supportate

Usate l'elemento `<supportedLanguages>` nel file descrittore dell'applicazione per identificare le lingue supportate dall'applicazione. Questo elemento è utilizzato solo da iOS, dal runtime autonomo Mac e dalle applicazioni Android, e viene ignorato da tutti gli altri tipi di applicazioni.

Se non specificate l'elemento `<supportedLanguages>`, l'impostazione predefinita prevede che il compilatore esegua le operazioni seguenti in base al tipo di applicazione:

- iOS - Tutte le lingue supportate dal runtime AIR sono elencate in iOS App Store come lingue supportate dell'applicazione.
- Runtime autonomo Mac - L'applicazione impacchettata con il runtime autonomo non contiene informazioni sulla localizzazione.
- Android - Il pacchetto dell'applicazione contiene le risorse per tutte le lingue supportate dal runtime AIR.

Per ulteriori informazioni, vedete “[supportedLanguages](#)” a pagina 247.

Definizione di pacchetti di risorse

Il framework di localizzazione HTML legge le versioni localizzate delle stringhe dai file di *localizzazione*. Un file di localizzazione è una raccolta di valori basati su chiave, serializzati in un file di testo. Un file di localizzazione viene anche detto *pacchetto*.

Create una sottodirectory della directory di progetto dell'applicazione, denominata "locale". Potete anche utilizzare un nome diverso: vedete “[Personalizzazione delle impostazioni di AIR HTML Localizer](#)” a pagina 310. Questa directory comprenderà i file di localizzazione ed è detta anche *directory dei pacchetti*.

Create una sottodirectory della directory dei pacchetti per ogni impostazione locale supportata dall'applicazione. Denominate ogni sottodirectory in base al codice delle impostazioni locali, ad esempio assegnate alla directory del francese il codice "fr" e a quella dell'inglese il codice "en". Potete utilizzare un carattere di sottolineatura (_) per definire un'impostazione locale che comprenda una lingua e un codice di paese. Assegnate ad esempio alla directory dell'inglese americano il codice "en_us". In alternativa, potete utilizzare il trattino al posto del carattere di sottolineatura, come in "en-us". Il framework di localizzazione HTML supporta entrambi.

Potete aggiungere alla sottodirectory di un'impostazione locale il numero di file di risorse che desiderate. In genere si crea un file di localizzazione per ogni lingua (e lo si inserisce nella directory relativa alla lingua). Il framework di localizzazione HTML comprende un metodo `getFile()` che consente di leggere il contenuto di un file (vedete “[Ottenere risorse per impostazioni locali specifiche](#)” a pagina 312).

I file che hanno l'estensione `.properties` sono detti file di proprietà della localizzazione. Potete utilizzarli per specificare coppie chiave-valore per un'impostazione locale. Un file `properties` definisce un valore stringa per ogni riga. L'esempio seguente definisce un valore stringa "Hello in English." per una chiave denominata `greeting`:

```
greeting=Hello in English.
```

Un file `properties` contenente il testo seguente definisce sei coppie chiave-valore:

```
title=Sample Application
greeting=Hello in English.
exitMessage=Thank you for using the application.
color1=Red
color2=Green
color3=Blue
```

L'esempio mostra una versione inglese del file `properties` da inserire nella directory `en`.

La versione francese di questo file `properties` si trova nella directory `fr`:


```
title=Application Example
greeting=Bonjour en français.
exitMessage=Merci d'avoir utilisé cette application.
color1=Rouge
color2=Vert
color3=Bleu
```

Potete definire più file di risorse per diversi tipi di informazioni. Un file `legal.properties` può ad esempio contenere testo legale standard (come le informazioni di copyright). Potete riutilizzare la stessa risorsa in più applicazioni. Analogamente, potete definire file distinti che specifichino contenuto localizzato per parti differenti dell'interfaccia utente.

Per supportare più lingue, utilizzate la codifica UTF-8 per questi file.

Gestione di catene di impostazioni locali

Quando l'applicazione carica il file `AIRLocalizer.js`, esegue l'esame delle impostazioni locali specificate nell'applicazione. Queste impostazioni locali corrispondono alle sottodirectory della directory dei pacchetti (vedete ["Definizione di pacchetti di risorse"](#) a pagina 307). Questo elenco di impostazioni locali disponibili è detto anche *catena di impostazioni locali*. Il file `AIRLocalizer.js` ordina automaticamente la catena di impostazioni locali in base all'ordine preferito definito nelle impostazioni del sistema operativo. La proprietà `Capabilities.languages` elenca le lingue dell'interfaccia utente del sistema operativo nell'ordine preferito.

In tal modo, se un'applicazione definisce risorse per le impostazioni locali "en", "en_US" ed "en_UK", il framework di AIR HTML Localizer ordina la catena di impostazioni locali in modo corretto. Quando viene avviata un'applicazione in un sistema in cui "en" rappresenta l'impostazione locale principale, la catena delle impostazioni locali viene così ordinata: ["en", "en_US", "en_UK"]. In tal caso, l'applicazione cerca le risorse prima nel pacchetto "en", poi in quello "en_US".

Se invece in un sistema "en_US" rappresenta l'impostazione locale principale, l'ordinamento corrisponde a ["en_US", "en", "en_UK"]. In tal caso, l'applicazione cerca le risorse prima nel pacchetto "en_US" e poi in quello "en".

Per impostazione predefinita, l'applicazione definisce le prime impostazioni locali della catena come impostazioni locali predefinite da utilizzare. Potete chiedere all'utente di selezionare le impostazioni locali alla prima esecuzione dell'applicazione e quindi memorizzare la selezione in un file di preferenze e utilizzare tali impostazioni locali al successivo avvio dell'applicazione.

L'applicazione può utilizzare stringhe delle risorse di qualunque impostazione locale della catena. Se per un'impostazione locale non è definita alcuna stringa di risorse, l'applicazione utilizzerà la successiva stringa di risorse corrispondente per altre impostazioni locali definite nella catena.

Potete personalizzare la catena di impostazioni locali chiamando il metodo `setLocaleChain()` dell'oggetto `Localizer`. Vedete ["Definizione della catena di impostazioni locali"](#) a pagina 311.

Aggiornamento di elementi DOM con contenuto localizzato

Un elemento dell'applicazione può fare riferimento al valore di una chiave di un file di proprietà di localizzazione. Ad esempio, l'elemento `title` dell'esempio seguente specifica un attributo `local_innerHTML`. Il framework di localizzazione utilizza questo attributo per cercare un valore localizzato. Per impostazione predefinita, il framework cerca i nomi degli attributi che iniziano con "local_". Il framework aggiorna gli attributi con nomi che corrispondono al testo successivo a "local_". In tal caso, il framework imposta l'attributo `innerHTML` dell'elemento `title`. L'attributo `innerHTML` utilizza il valore definito per la chiave `mainWindowTitle` nel file delle proprietà predefinite (`default.properties`):

```
<title local_innerHTML="default.mainWindowTitle"/>
```

Se per l'impostazione locale corrente non è definito alcun valore corrispondente, il framework di localizer cerca nel resto della catena di impostazioni locali. Utilizza l'impostazione locale successiva nella catena di impostazioni locali per cui è definito un valore.

Nell'esempio seguente, il testo (attributo `innerHTML`) dell'elemento `p` utilizza il valore della chiave `greeting` definito nel file delle proprietà predefinite:

```
<p local_innerHTML="default.greeting" />
```

Nell'esempio seguente, l'attributo `value` (e il testo visualizzato) dell'elemento `input` utilizza il valore della chiave `btnBlue` definita nel file delle proprietà predefinite:

```
<input type="button" local_value="default.btnBlue" />
```

Per aggiornare l'HTML DOM affinché vengano utilizzate le stringhe definite nella catena di impostazioni locali corrente, chiamate il metodo `update()` dell'oggetto `Localizer`. Se chiamate il metodo `update()`, l'oggetto `Localizer` analizza il DOM e applica modifiche se trova attributi di localizzazione ("`local_...`");

```
air.Localizer.localizer.update();
```

Potete definire valori sia per un attributo (come "`innerHTML`") che per il corrispondente attributo di localizzazione (come "`local_innerHTML`"). In tal caso, il framework di localizzazione sovrascrive solo il valore dell'attributo se trova un valore corrispondente nella catena di localizzazione. Ad esempio, l'elemento seguente definisce gli attributi `value` e `local_value`:

```
<input type="text" value="Blue" local_value="default.btnBlue"/>
```

Potete anche aggiornare solamente un elemento DOM specifico. Vedete la sezione successiva, "[Aggiornamento degli elementi DOM per l'uso dell'impostazione locale corrente](#)" a pagina 309.

Per impostazione predefinita, AIR HTML Localizer utilizza "`local_`" come prefisso per gli attributi che definiscono le impostazioni di localizzazione di un elemento. Ad esempio, per impostazione predefinita un attributo `local_innerHTML` definisce il pacchetto e il nome della risorsa utilizzati per il valore `innerHTML` di un elemento. Inoltre, per impostazione predefinita un attributo `local_value` definisce il pacchetto e il nome della risorsa utilizzati per l'attributo `value` di un elemento. È possibile configurare Localizer per utilizzare un prefisso di attributo diverso da "`local_`". Vedete "[Personalizzazione delle impostazioni di AIR HTML Localizer](#)" a pagina 310.

Aggiornamento degli elementi DOM per l'uso dell'impostazione locale corrente

Quando l'oggetto `Localizer` aggiorna l'HTML DOM, gli elementi contrassegnati utilizzano i valori degli attributi in base a stringhe definite nella catena corrente delle impostazioni locali. Affinché il localizer HTML aggiorni l'HTML DOM, chiamate il metodo `update()` dell'oggetto `Localizer`:

```
air.Localizer.localizer.update();
```

Per aggiornare solamente un elemento DOM specificato, passatelo come parametro al metodo `update()`. Il metodo `update()` dispone di un solo parametro, `parentNode`, che è opzionale. Se specificato, il parametro `parentNode` definisce l'elemento DOM da localizzare. Quando chiamate il metodo `update()` e specificate un parametro `parentNode`, vengono impostati i valori localizzati per tutti gli elementi secondari che specificano attributi di localizzazione.

Ad esempio, considerate il seguente 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>
```

Per aggiornare questo elemento affinché vengano utilizzate le stringhe localizzate specificate nella catena corrente di impostazioni locali, utilizzate il codice JavaScript seguente:

```
var divElement = window.document.getElementById("colorsDiv");
air.Localizer.localizer.update(divElement);
```

Se nella catena delle impostazioni locali non viene trovato alcun valore chiave, il framework di localizzazione imposta il valore dell'attributo sul valore dell'attributo "local_". Ad esempio, nell'esempio precedente, supponete che il framework di localizzazione non riesca a trovare un valore per la chiave `lblColors` (in qualsiasi file `default.properties` della catena di impostazioni locali). In tal caso, utilizza `default.lblColors` come valore `innerHTML`. L'uso di questo valore consente di indicare (allo sviluppatore) risorse mancanti.

Il metodo `update()` invia un evento `resourceNotFound` quando non riesce a trovare una risorsa nella catena di impostazioni locali. La costante `air.Localizer.RESOURCE_NOT_FOUND` definisce la stringa `"resourceNotFound"`. L'evento dispone di tre proprietà: `bundleName`, `resourceName` e `locale`. La proprietà `bundleName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `resourceName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa.

Il metodo `update()` invia un evento `bundleNotFound` quando non riesce a trovare un pacchetto specificato. La costante `air.Localizer.BUNDLE_NOT_FOUND` definisce la stringa `"bundleNotFound"`. L'evento dispone di due proprietà: `bundleName` e `locale`. La proprietà `bundleName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa.

Il metodo `update()` opera in modo asincrono (e invia gli eventi `resourceNotFound` e `bundleNotFound` in modo asincrono). Il codice seguente imposta listener di eventi per gli eventi `resourceNotFound` e `bundleNotFound`:

```
air.Localizer.localizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.update();
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":@" + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":@" + event.locale);
}
```

Personalizzazione delle impostazioni di AIR HTML Localizer

Il metodo `setBundlesDirectory()` dell'oggetto `Localizer` consente di personalizzare il percorso della directory dei pacchetti. Il metodo `setLocalAttributePrefix()` dell'oggetto `Localizer` consente di personalizzare il percorso della directory dei pacchetti e il valore dell'attributo utilizzato da `Localizer`.

La directory dei pacchetti predefinita corrisponde alla sottodirectory delle impostazioni locali della directory dell'applicazione. Potete specificare un'altra directory chiamando il metodo `setBundlesDirectory()` dell'oggetto `Localizer`. Questo metodo accetta un parametro, `path`, che corrisponde al percorso della directory dei pacchetti predefinita in formato stringa. Il valore del parametro `path` può corrispondere a uno dei seguenti:

- Una stringa che definisce un percorso relativo alla directory dell'applicazione, ad esempio `"locales"`

- Una stringa che definisce un URL valido che utilizza gli schemi URL `app`, `app-storage` o `file`, ad esempio `"app://languages"` (non utilizzate lo schema URL `http`)
- Un oggetto `File`

Per informazioni sugli URL e sui percorsi di directory, vedete:

- [Percorsi degli oggetti File](#) (sviluppatori ActionScript)
- [Paths of File objects](#) (sviluppatori HTML)

Ad esempio, il codice seguente imposta la directory dei pacchetti su una sottodirectory denominata "languages" nella directory di memorizzazione dell'applicazione (non della directory dell'applicazione):

```
air.Localizer.localizer.setBundlesDirectory("languages");
```

Se per il parametro `path` non viene passato un percorso valido, il metodo genera un'eccezione `BundlePathNotFoundError`. Questo errore ha `"BundlePathNotFoundError"` come proprietà `name` e la relativa proprietà `message` specifica il percorso non valido.

Per impostazione predefinita, AIR HTML Localizer utilizza `"local_"` come prefisso per gli attributi che definiscono le impostazioni di localizzazione di un elemento. Ad esempio, l'attributo `local_innerHTML` definisce il pacchetto e il nome della risorsa utilizzati per il valore `innerHTML` del seguente elemento `input`:

```
<p local_innerHTML="default.greeting" />
```

Il metodo `setLocalAttributePrefix()` dell'oggetto `Localizer` consente di utilizzare un prefisso dell'attributo diverso da `"local_"`. Questo metodo statico accetta un parametro, ovvero la stringa che desiderate utilizzare come prefisso dell'attributo. Ad esempio, il codice seguente imposta il framework di localizzazione affinché venga utilizzato `"loc_"` come prefisso dell'attributo:

```
air.Localizer.localizer.setLocalAttributePrefix("loc_");
```

Potete personalizzare il prefisso dell'attributo utilizzato dal framework di localizzazione. Questa operazione può essere utile se il valore predefinito (`"local_"`) entra in conflitto con il nome di un altro attributo che avete utilizzato nel codice. Assicuratevi di utilizzare caratteri validi per gli attributi HTML quando chiamate questo metodo (ad esempio, il valore non può contenere un carattere di spazio vuoto).

Per ulteriori informazioni sull'uso degli attributi di localizzazione negli elementi HTML, vedete ["Aggiornamento di elementi DOM con contenuto localizzato"](#) a pagina 308.

Le impostazioni relative alla directory dei pacchetti e al prefisso degli attributi non persistono tra le diverse sessioni dell'applicazione. Se utilizzate impostazioni personalizzate per la directory dei pacchetti o il prefisso degli attributi, assicuratevi di configurarle a ogni inizializzazione dell'applicazione.

Definizione della catena di impostazioni locali

Per impostazione predefinita, quando caricate il codice di `AIRLocalizer.js`, viene impostata la catena di impostazioni locali predefinita. Le impostazioni locali disponibili nella directory dei pacchetti e nelle impostazioni della lingua del sistema operativo definiscono questa catena di impostazioni locali. Per informazioni dettagliate, vedete ["Gestione di catene di impostazioni locali"](#) a pagina 308.

È possibile modificare la catena delle impostazioni locali chiamando il metodo statico `setLocaleChain()` dell'oggetto `Localizer`. Ad esempio, potete chiamare questo metodo se l'utente indica una preferenza per una lingua specifica. Il metodo `setLocaleChain()` accetta un parametro, `chain`, che è un array di impostazioni locali, ad esempio `["fr_FR", "fr", "fr_CA"]`. L'ordine delle impostazioni locali nell'array imposta l'ordine in cui il framework cerca le risorse (nelle operazioni successive). Se non si trova una risorsa per la prima impostazione locale della catena, si continua a cercare nelle risorse di un'altra impostazione locale. Se l'argomento `chain` non è presente, non è un array oppure è un array vuoto, la funzione non viene eseguita e viene generata un'eccezione `IllegalArgumentsError`.

Il metodo statico `getLocaleChain()` dell'oggetto `Localizer` restituisce un array che elenca le impostazioni locali della catena di impostazioni locali corrente.

Il codice seguente legge la catena di impostazioni locali corrente e aggiunge due impostazioni locali per il francese all'inizio della catena:

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
```

Il metodo `setLocaleChain()` invia un evento "change" quando aggiorna la catena di impostazioni locali. La costante `air.Localizer.LOCALE_CHANGE` definisce la stringa "change". L'evento dispone di una proprietà, `localeChain`, un array di codici di impostazioni locali nella nuova catena di impostazioni locali. Il codice seguente imposta un listener di eventi per questo 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 proprietà statica `air.Localizer.ultimateFallbackLocale` rappresenta l'impostazione locale utilizzata quando l'applicazione non supporta alcuna preferenza dell'utente. Il valore predefinito è "en". Potete impostarlo su un'altra impostazione locale, come illustrato nel codice seguente:

```
air.Localizer.ultimateFallbackLocale = "fr";
```

Ottenere risorse per impostazioni locali specifiche

Il metodo `getString()` dell'oggetto `Localizer` restituisce la stringa definita per una risorsa in un'impostazione locale specifica. Non è necessario specificare un valore `locale` quando si chiama il metodo. In tal caso, il metodo analizza l'intera catena delle impostazioni locali e restituisce la stringa nella prima impostazione locale che include il nome della risorsa specificato. Il metodo dispone dei parametri seguenti:

Parametro	Descrizione
bundleName	Il pacchetto che contiene la risorsa. È il nome file del file delle proprietà senza l'estensione .properties. Ad esempio, se questo parametro è impostato su "alerts", il codice di Localizer cerca nei file di localizzazione denominati alerts.properties.
resourceName	Il nome della risorsa.
templateArgs	Opzionale. Un array di stringhe per sostituire i tag numerati nella stringa di sostituzione. Considerate ad esempio una chiamata a una funzione con parametro templateArgs impostato su ["Raúl", "4"] e dove la stringa di risorsa corrispondente è "Hello, {0}. You have {1} new messages.". In tal caso, la funzione restituisce "Hello, Raúl. You have 4 new messages.". Per ignorare questa impostazione, passate un valore null.
locale	Opzionale. Il codice dell'impostazione locale (ad esempio "en", "en_us" o "fr") da utilizzare. Se viene specificata un'impostazione locale ma non viene trovato un valore corrispondente, il metodo non continua a cercare i valori in altre impostazioni locali della catena. Se non è specificato alcun codice di impostazione locale, la funzione restituisce la stringa nella prima impostazione locale della catena che include un valore per il nome della risorsa specificata.

Il framework di localizzazione può aggiornare attributi HTML DOM contrassegnati. Tuttavia, è possibile utilizzare le stringhe localizzate in altri modi. Ad esempio, è possibile utilizzare una stringa in codice HTML generato dinamicamente o come valore di parametro in una chiamata di funzione. Ad esempio, il codice seguente chiama la funzione `alert()` con la stringa definita nella risorsa `error114` del file delle proprietà predefinito dell'impostazione locale `fr_FR`:

```
alert(air.Localizer.localizer.getString("default", "error114", null, "fr_FR"));
```

Il metodo `getString()` invia un evento `resourceNotFound` quando non riesce a trovare una risorsa nel pacchetto specificato. La costante `air.Localizer.RESOURCE_NOT_FOUND` definisce la stringa `"resourceNotFound"`. L'evento dispone di tre proprietà: `bundleName`, `resourceName` e `locale`. La proprietà `bundleName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `resourceName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa.

Il metodo `getString()` invia un evento `bundleNotFound` quando non riesce a trovare il pacchetto specificato. La costante `air.Localizer.BUNDLE_NOT_FOUND` definisce la stringa `"bundleNotFound"`. L'evento dispone di due proprietà: `bundleName` e `locale`. La proprietà `bundleName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa.

Il metodo `getString()` opera in modo asincrono (e invia gli eventi `resourceNotFound` e `bundleNotFound` in modo asincrono). Il codice seguente imposta listener di eventi per gli eventi `resourceNotFound` e `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);
}
```

Il metodo `getResourceBundle()` dell'oggetto `Localizer` restituisce un pacchetto specifico per una determinata impostazione locale. Il valore restituito del metodo è un oggetto le cui proprietà corrispondono alle chiavi nel pacchetto. Se l'applicazione non è in grado di trovare il pacchetto specificato, il metodo restituisce `null`.

Il metodo accetta due parametri: `locale` e `bundleName`.

Parametro	Descrizione
<code>locale</code>	L'impostazione locale (ad esempio "fr").
<code>bundleName</code>	Il nome del pacchetto.

Il codice riportato di seguito, ad esempio, chiama il metodo `document.write()` per caricare il pacchetto predefinito per l'impostazione locale `fr`. Chiama quindi il metodo `document.write()` per scrivere i valori delle chiavi `str1` e `str2` in tale pacchetto:

```
var aboutWin = window.open();
var bundle = localizer.getResourceBundle("fr", "default");
aboutWin.document.write(bundle.str1);
aboutWin.document.write("<br/>");
aboutWin.document.write(bundle.str2);
aboutWin.document.write("<br/>");
```

Il metodo `getResourceBundle()` invia un evento `bundleNotFound` quando non riesce a trovare il pacchetto specificato. La costante `air.Localizer.BUNDLE_NOT_FOUND` definisce la stringa "bundleNotFound". L'evento dispone di due proprietà: `bundleName` e `locale`. La proprietà `bundleName` corrisponde al nome del pacchetto in cui non si trova la risorsa. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa.

Il metodo `getFile()` dell'oggetto `Localizer` restituisce il contenuto di un pacchetto, come una stringa, per un'impostazione locale specifica. Il file di pacchetto viene letto come file UTF-8. Il metodo comprende i parametri seguenti:

Parametro	Descrizione
<code>resourceFileName</code>	Il nome del file della risorsa (come "about.html").
<code>templateArgs</code>	Opzionale. Un array di stringhe per sostituire i tag numerati nella stringa di sostituzione. Ad esempio, considerate di richiamare la funzione in cui il parametro <code>templateArgs</code> è ["Raúl", "4"] e la stringa di risorsa corrispondente contengano due righe: <code><html></code> <code><body>Hello, {0}. You have {1} new messages.</body></code> <code></html></code> In tal caso, la funzione restituisce una stringa con due righe: <code><html></code> <code><body>Hello, Raúl. You have 4 new messages. </body></code> <code></html></code>
<code>locale</code>	Il codice dell'impostazione locale, ad esempio "en_GB", da utilizzare. Se viene specificata un'impostazione locale ma non viene trovato un file corrispondente, il metodo non continua a cercare in altre impostazioni locali della catena. Se <i>non</i> viene specificato alcun codice di impostazione locale, la funzione restituisce il testo nella prima impostazione locale della catena che presenta un file corrispondente a <code>resourceFileName</code> .

Ad esempio, il codice seguente chiama il metodo `document.write()` utilizzando il contenuto del file `about.html` dell'impostazione locale `fr`:

```
var aboutWin = window.open();
var aboutHtml = localizer.getFile("about.html", null, "fr");
aboutWin.document.close();
aboutWin.document.write(aboutHtml);
```

Il metodo `getFile()` invia un evento `fileNotFound` se non riesce a trovare una risorsa nella catena di impostazioni locali. La costante `air.Localizer.FILE_NOT_FOUND` definisce la stringa `"resourceNotFound"`. Il metodo `getFile()` opera in modo asincrono (e invia l'evento `fileNotFound` in modo asincrono). L'evento ha due proprietà: `fileName` e `locale`. La proprietà `fileName` corrisponde al nome del file non trovato. La proprietà `locale` corrisponde al nome dell'impostazione locale in cui non si trova la risorsa. Il codice seguente imposta un listener di eventi per questo 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);
}
```

Altri argomenti presenti nell'Aiuto

[Building a multilingual HTML-based application \(Creazione di un'applicazione HTML multilingue\)](#)

Capitolo 21: Variabili d'ambiente dei percorsi

AIR SDK contiene alcuni programmi che possono essere avviati da una riga di comando o una finestra di terminale. L'esecuzione di questi programmi può essere spesso più pratica se la directory bin del kit SDK è inclusa nella variabile d'ambiente dei percorsi.

Le informazioni che seguono spiegano come impostare il percorso in Windows, Mac e Linux e forniscono una guida generale. Tuttavia, poiché le impostazioni di configurazione variano molto da un computer all'altro, la procedura non può funzionare con tutti i sistemi. Eventualmente, fate riferimento alla documentazione del vostro sistema operativo o ad altre fonti su Internet.

Impostazione del percorso su Linux e Mac OS con la shell Bash

Quando digitate un comando in una finestra di terminale, la shell (il programma che legge quello che avete digitato e tenta di rispondere in modo appropriato) deve prima individuare il programma di comando nel file system. La shell cerca i comandi in un elenco di directory memorizzate in una variabile d'ambiente denominata \$PATH. Per vedere il contenuto corrente del percorso, digitate:

```
echo $PATH
```

Viene restituito un elenco di directory separate da due punti, con un aspetto simile al seguente:

```
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin
```

L'obiettivo è quello di aggiungere il percorso della directory bin di AIR SDK all'elenco, in modo tale che la shell possa individuare i tool ADT e ADL. Presupponendo che abbiate inserito AIR SDK in /Users/fred/SDKs/AIR, il comando seguente aggiungerà le directory necessarie al percorso:

```
export PATH=$PATH:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

Nota: se il percorso contiene caratteri di spazio vuoti, anteporre una barra rovesciata a tali caratteri, come nell'esempio seguente:

```
/Users/fred\ jones/SDKs/AIR\ 2.5\ SDK/bin
```

Potete quindi usare di nuovo il comando echo per verificare che il risultato sia quello desiderato:

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

Fin qui tutto bene. A questo punto dovrete essere in grado di digitare i seguenti comandi e ottenere una risposta incoraggiante:

```
adt -version
```

Se avete modificato correttamente la variabile \$PATH, il comando dovrebbe restituire la versione di ADT.

C'è ancora un problema però: la prossima volta che aprite una nuova finestra di terminale, noterete che le nuove voci del percorso non ci sono più. Il comando di impostazione del percorso deve essere eseguito ogni volta che aprite una nuova finestra di terminale.

Una soluzione tipica per questo problema è quella di aggiungere il comando a uno degli script di avvio utilizzati dalla shell. In Mac OS, potete creare il file `.bash_profile` nella directory `~/username` per eseguirlo ogni volta che aprite una nuova finestra di terminale. In Ubuntu, lo script di avvio eseguito quando avviate una nuova finestra di terminale è `.bashrc`. Altre distribuzioni Linux e altre prevedono convenzioni analoghe.

Per aggiungere il comando allo script di avvio della shell:

- 1 Accedete alla directory iniziale:

```
cd
```

- 2 Create il profilo di configurazione della shell (se necessario) e reindirizzate il testo che digitate alla fine del file con `"cat >>".` Utilizzate il file appropriato per la vostra combinazione di sistema operativo e shell. Potete usare `.bash_profile` in Mac OS e `.bashrc` in Ubuntu, per esempio.

```
cat >> .bash_profile
```

- 3 Digitate il testo da aggiungere al file:

```
export PATH=$PATH:/Users/cward/SDKs/android/tools:/Users/cward/SDKs/AIR/bin
```

- 4 Interrompete il reindirizzamento del testo premendo `CTRL-MAIUSC-D` sulla tastiera.

- 5 Visualizzate il file per verificare che tutto sia a posto:

```
cat .bash_profile
```

- 6 Aprite una nuova finestra di terminale per controllare il percorso

```
echo $PATH
```

I dati aggiunti al percorso dovrebbero essere presenti.

Se in seguito create una nuova versione di uno dei kit SDK in una directory differente, ricordatevi di aggiornare il comando del percorso nel file di configurazione, altrimenti la shell continuerà a usare la versione vecchia.

Impostazione del percorso in Windows

Quando aprite una finestra di comando in Windows, questa eredita le variabili d'ambiente globali definite nelle proprietà di sistema. Una delle variabili importanti è il percorso, ovvero l'elenco delle directory in cui il programma di comando esegue la ricerca quando digitate il nome di un programma da eseguire. Per verificare cosa è incluso nel percorso quando usate una finestra di comando, potete digitare:

```
set path
```

Verrà restituito un elenco di directory separate da punto e virgola, che avrà un aspetto simile al seguente:

```
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

L'obiettivo è quello di aggiungere il percorso della directory bin di AIR SDK all'elenco, in modo tale che il programma di comando possa individuare i tool ADT e ADL. Presupponendo che abbiate inserito AIR SDK in `C:\SDKs\AIR`, potete aggiungere il percorso corretto con la procedura seguente:

- 1 Aprite la finestra Proprietà di sistema dal Pannello di controllo oppure facendo clic su Risorse del computer con il pulsante destro e scegliendo Proprietà dal menu.
- 2 Nella scheda Avanzate, fate clic sul pulsante Variabili d'ambiente.

- 3 Selezionate la voce Path nella sezione Variabili di sistema della finestra di dialogo Variabili d'ambiente.
- 4 Fate clic su Modifica.
- 5 Fate scorrere il contenuto fino alla fine del testo nel campo Valore variabile.
- 6 Immettete il testo seguente alla fine del valore corrente.

```
;C:\SDKs\AIR\bin
```

- 7 Fate clic su OK in tutte le finestre di dialogo per salvare il percorso.

Se sono aperte delle finestre di comando, tenete presente che i relativi ambienti non saranno aggiornati. Aprite una nuova finestra di comando e digitate il comando seguente per verificare che i percorsi siano stati impostati correttamente:

```
adt -version
```

Se in seguito cambiate la posizione di AIR SDK oppure ne aggiungete una nuova versione, ricordate di aggiornare la variabile dei percorsi.