ADOBE® AIR® アプリケーションの構築

法律上の注意

目次

第 20 章：AIR アプリケーションのローカライズ

第 21 章：PATH 環境変数

第 1 章：Adobe AIR について

Adobe® AIR® は複数のオペレーティングシステム対応のマルチスクリーンランタイムです。これを使用すると、既に持っている Web 開発スキルを利用して、リッチインターネットアプリケーション（RIA）を作成してデスクトップやモバイルデバイスにデプロイできます。ActionScript 3.0 で Adobe® Flex および Adobe® Flash® を使用して、デスクトップ、テレビおよびモバイル AIR アプリケーションを構築できます（SWF ベース）。デスクトップ AIR アプリケーションは、HTML、JavaScript® および Ajax でも作成できます（HTML ベース）。

Adobe AIR の基礎知識と使用方法について詳しくは、Adobe AIR Developer Connection（http://www.adobe.com/jp/devnet/air/）を参照してください。

AIR を使用すると、使い慣れた環境で作業でき、自分にとって最も使用しやすいツールや方法を利用できます。さらに、Flash、Flex、HTML、JavaScript および Ajax をサポートすることにより、ニーズに合った最高の操作性を実現します。

例えば、次のようなテクノロジを使用してアプリケーションを開発できます。これらのテクノロジを組み合わせて開発することもできます。

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

AIR アプリケーションはネイティブのアプリケーションと同じように操作できます。ランタイムをユーザーのコンピューターまたはデバイスに一度インストールしておくと、AIR アプリケーションが他のデスクトップアプリケーションと同じようにインストールされ、実行されます（iOS では、AIR ランタイムは個別にインストールされません。各 iOS AIR アプリケーションはスタンドアロンのアプリケーションとなります）。

ランタイムにより、アプリケーションをデプロイするための一貫性のあるクロスオペレーティングシステム対応のプラットフォームおよびフレームワークが提供されます。これにより、デスクトップ間の機能および操作の一貫性が確保され、クロスブラウザーテストが不要になります。開発のターゲットを特定のオペレーティングシステムではなくランタイムにすることで、次のようなメリットがあります。

- AIR 用に開発されたアプリケーションは、追加の作業なしで、複数のオペレーティングシステム間で実行されます。ランタイムにより、AIR でサポートされているすべてのオペレーティングシステム間で、一貫性のある想定したとおりのプレゼンテーションと操作が実現します。
- 既存の Web テクノロジやデザインパターンを利用できるので、アプリケーションをより早く作成できます。従来のデスクトップ開発テクノロジーや複雑なネイティブコードを習得することなく、Web ベースのアプリケーションをデスクトップに拡張できます。
- アプリケーションの開発は、C や C++ などの下位の言語を使用するよりも簡単です。各オペレーティングシステム固有の複雑な下位 API を管理する必要もありません。

AIR 用のアプリケーションの開発では、豊富なフレームワークと API のセットを利用できます。

- ランタイムで提供される AIR 固有の API と AIR フレームワーク
- SWF ファイルで使用される ActionScript API と Flex フレームワーク（およびその他の ActionScript ベースのライブラリとフレームワーク）
- HTML、CSS および JavaScript
- 大部分の Ajax フレームワーク
- Adobe AIR 用のネイティブ拡張には ActionScript API が備えられており、ネイティブコードでプログラムされているプラットフォーム固有の機能にアクセスできます。ネイティブ拡張を使用すると、古いネイティブコードや、よりパフォーマンスの高いネイティブコードにもアクセスできます。

AIR はアプリケーションの作成、デプロイ、使用の方法に変革をもたらします。クリエイティブコントロールが向上し、Flash、Flex、HTML および Ajax ベースのアプリケーションをデスクトップ、モバイルデバイス、およびテレビに拡張できます。

新しい各 AIR アップデートの内容については、Adobe AIR リリースノート（http://www.adobe.com/go/learn_air_relnotes_jp）を参照してください。

第 2 章：Adobe AIR のインストール

Adobe® AIR® ランタイムを使用すると、AIR アプリケーションを実行できます。ランタイムは次のような方法でインストールできます。

- ランタイムを単独でインストールします（AIR アプリケーションはインストールしません）。
- Web ページインストールの「badge」を使用して初めて AIR アプリケーションをインストールする際に、ランタイムのインストールを求めるメッセージが表示されます。
- アプリケーションとランタイムの両方をインストールするカスタムインストーラーを作成します。この方法で AIR ランタイムを配布する場合は、アドビの承認が必要です。「Adobe runtime licensing」ページで承認を要求できます。アドビでは、このようなインストーラーを構築するためのツールは提供していませんが、サードパーティのインストーラーツールキットが多数あります。
- AIR をキャプティブランタイムとしてバンドルする AIR アプリケーションをインストールします。キャプティブランタイムは、バンドルを行うアプリケーションでのみ使用され、他の AIR アプリケーションを実行するために使用されることはありません。ランタイムのバンドルは、Mac および Windows ではオプションです。iOS では、すべてのアプリケーションにバンドルされたランタイムが含まれています。AIR 3.7 より、すべての Android アプリケーションはデフォルトでバンドルされたランタイムを含みます（ただし、別のランタイムを使用するオプションもあります）。
- AIR SDK、Adobe® Flash® Builder™、Adobe Flex® SDK などの AIR 開発環境を設定します（AIR コマンドライン開発ツールが含まれます）。SDK に含まれるランタイムは、アプリケーションをデバッグする場合にのみ使用されます。インストールされた AIR アプリケーションの実行には使用されません。

AIR のインストールおよび AIR アプリケーションの実行に必要なシステム構成について詳しくは、「Adobe AIR：必要システム構成」（http://www.adobe.com/jp/products/air/systemreqs/）を参照してください。

AIR アプリケーションまたは AIR ランタイム自体をインストール、更新または削除すると、ランタイムインストーラーと AIR アプリケーションインストーラーの両方でログファイルが作成されます。これらのログの内容は、インストールの問題の原因を突き止めるのに役立ちます。インストールのログに関する説明を参照してください。

Adobe AIR のインストール

ランタイムをインストールまたはアップデートする作業は、当該コンピューターの管理権限を持つユーザーが実行する必要があります。

Windows コンピューターでのランタイムのインストール

1. http://get.adobe.com/air からランタイムインストールファイルをダウンロードします。
2. ランタイムインストールファイルをダブルクリックします。
3. インストールウィンドウで、表示されるメッセージに従ってインストールを完了します。

Mac OS コンピューターでのランタイムのインストール

1. http://get.adobe.com/air からランタイムインストールファイルをダウンロードします。
2. ランタイムインストールファイルをダブルクリックします。
3. インストールウィンドウで、表示されるメッセージに従ってインストールを完了します。
4. インストーラーで認証ウィンドウが表示された場合は、Mac OS のユーザー名とパスワードを入力します。

Linux コンピューターでのランタイムのインストール

注意：現時点で、AIR 2.7 以降は、Linux ではサポートされていません。Linux にデプロイされた AIR アプリケーションでは引き続き AIR 2.6 SDK を使用します。

バイナリインストーラーを使用する場合：

1. http://kb2.adobe.com/cps/853/cpsid_85304.html からインストールバイナリファイルをダウンロードします。
2. ファイル権限を設定し、インストーラーアプリケーションを実行できるようにします。コマンドラインからファイル権限を設定するには、次のコマンドを使用します。

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

   Linux のバージョンによっては、コンテキストメニューから開くことができるプロパティダイアログでファイル権限を設定できます。
3. コマンドラインからインストーラーを実行するか、ラインタイムインストールファイルをダブルクリックします。
4. インストールウィンドウで、表示されるメッセージに従ってインストールを完了します。

Adobe AIR はネイティブパッケージとしてインストールされます。つまり、rpm ベースのディストリビューションでは rpm として、Debian ディストリビューションでは deb としてインストールされます。現時点では、その他のパッケージ形式は AIR でサポートされていません。

パッケージインストーラーを使用する場合：

1. http://kb2.adobe.com/cps/853/cpsid_85304.html から AIR パッケージファイルを見つけます。システムでサポートされているパッケージ形式に応じて、rpm または Debian パッケージをダウンロードします。
2. 必要に応じて、AIR パッケージファイルをダブルクリックしてパッケージをインストールします。

   コマンドラインからインストールすることもできます。

   a Debian システムの場合：

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

   b rpm ベースのシステムの場合：

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

   既存のバージョンをアップデートする場合（AIR 1.5.3 以降）：

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

AIR 2 および AIR アプリケーションをインストールするには、コンピューターの管理者権限が必要です。

Adobe AIR は、/opt/Adobe AIR/Versions/1.0 にインストールされます。

AIR では、「application/vnd.adobe.air-application-installer-package+zip」という MIME タイプを登録します。つまり、これが .air ファイルの MIME タイプになるので、.air ファイルは AIR ランタイムに登録されます。

Android デバイスへのランタイムのインストール

Android Market から AIR ランタイムの最新リリースをインストールできます。

Web ページのリンク、または ADT -installRuntime コマンドを使用して、AIR ランタイムの開発バージョンをインストールできます。同時にインストールできる AIR ランタイムのバージョンは 1 つのみです。リリースバージョンと開発バージョンの両方をインストールすることはできません。

詳しくは、173 ページの「ADT installRuntime コマンド」を参照してください。

iOS デバイスへのランタイムのインストール

必要な AIR ランタイムコードは、iPhone、iTouch および iPad デバイス用に作成された各アプリケーションにバンドルされます。個々のランタイムコンポーネントはインストールしません。

関連項目

70 ページの「AIR for iOS」

Adobe AIR の削除

インストールしたランタイムを削除するには、以下の手順に従います。

Windows コンピューターでのランタイムの削除

1. Windows のスタートメニューで、設定／コントロールパネルを選択します。
2. コントロールパネルで、「プログラム」／「プログラムと機能」を開くか、「プログラムの追加と削除」を開きます（実行している Windows のバージョンによって異なります）。
3. 「Adobe AIR」を選択してランタイムを削除します。
4. 「変更」ボタンまたは「削除」ボタンをクリックします。

Mac OS コンピューターでのランタイムの削除

- / アプリケーション / ユーティリティフォルダーにある「Adobe AIR Uninstaller」をダブルクリックします。

Linux コンピューターでのランタイムの削除

次のいずれかの操作を行います。

- アプリケーションメニューから「Adobe AIR Uninstaller」コマンドを選択します。
- AIR インストーラーバイナリを -uninstall オプションを指定して実行します。
- パッケージマネージャーで AIR パッケージ（adobeair および adobecerts）を削除します。

Android デバイスからのランタイムの削除

1. デバイスで「設定」アプリケーションを開きます。
2. アプリケーション／アプリケーションの管理の Adobe AIR エントリをタップします。
3. 「アンインストール」ボタンをタップします。

ADT の -uninstallRuntime コマンドも使用できます。詳しくは、174 ページの「ADT uninstallRuntime コマンド」を参照してください。

バンドルされたランタイムの削除

キャプティブバンドルされたランタイムを削除するには、インストールに使用したアプリケーションを削除する必要があります。キャプティブランタイムは、インストールを行うアプリケーションを実行するためにのみ使用されます。

AIR サンプルアプリケーションのインストールと実行

AIR アプリケーションをインストールまたはアップデートする作業は、当該コンピューターの管理権限を持つユーザーが実行する必要があります。

AIR の機能を紹介する、いくつかのサンプルアプリケーションが利用できます。これらのサンプルアプリケーションにアクセスしてインストールするには、次の手順に従います。

1 AIR サンプルアプリケーションをダウンロードして実行します。コンパイルされたアプリケーションとソースコードを入手できます。

2 サンプルアプリケーションをダウンロードして実行するには、サンプルアプリケーションのダウンロードボタンをクリックします。アプリケーションのインストールと実行を確認するメッセージが表示されます。

3 サンプルアプリケーションをダウンロードして後で実行する場合は、ダウンロードリンクを選択します。AIR アプリケーションは、次の方法でいつでも実行できます。

- Windows の場合は、デスクトップ上のアプリケーションアイコンをダブルクリックするか、Windows のスタートメニューから AIR アプリケーションを選択します。
- Mac OS の場合は、アプリケーションアイコンをダブルクリックします。このアイコンは、デフォルトで各自のユーザーディレクトリのアプリケーションフォルダー（例：Macintosh HD/ ユーザ /JoeUser/ アプリケーション /）にインストールされています。

注意：最新の情報については、AIR のリリースノートを確認してください。リリースノートは、http://www.adobe.com/go/learn_air_relnotes_jp にあります。

Adobe AIR アップデート

アドビ システムズ社では、新機能や小さな問題に対する修正プログラムによって、Adobe AIR を定期的に更新しています。自動通知およびアップデート機能を使用すると、Adobe AIR のアップデートバージョンが利用可能になったときに、ユーザーに対して自動的に通知することができます。

Adobe AIR のアップデートを行い Adobe AIR が適切に機能するようにします。アップデートにはセキュリティに関する重要な変更が含まれる場合があります。新しいバージョンが利用可能になった場合、特に、セキュリティアップデートについて言及されている場合は、必ず Adobe AIR の最新バージョンにアップデートすることをお勧めします。

デフォルトでは、AIR アプリケーションを起動したときに、ランタイムによって、アップデートが利用可能であるかどうかがチェックされます。このチェックは、前回のアップデートチェックから 2 週間を越えると実行されます。アップデートが利用可能であれば、アップデートがバックグラウンドでダウンロードされます。

ユーザーは、AIR 設定マネージャーアプリケーションを使用して、自動アップデート機能を無効にできます。AIR 設定マネージャーアプリケーションは、次の場所でダウンロードできます。
http://airdownload.adobe.com/air/applications/SettingsManager/SettingsManager.air

Adobe AIR の通常のインストールプロセスでは、インストール環境の基本情報（オペレーティングシステムのバージョンや言語など）を送信するために、http://airinstall.adobe.com への接続が行われます。この情報は、1 回のインストールにつき 1 度だけ転送されます。この情報を使用することで、アドビ システムズ社では、インストールが成功したことを確認できます。個人を特定する情報を収集したり、転送したりしません。

キャプティブランタイムの更新

キャプティブランタイムバンドルと共にアプリケーションを配布しても、キャプティブランタイムは自動的には更新されません。ユーザーのセキュリティ確保のために、アドビが公開するアップデートを監視して、関連するセキュリティ変更が公開されたら、アプリケーションを新しいランタイムバージョンで更新する必要があります。

第 3 章：AIR API の操作

Adobe® AIR® には、Adobe® Flash® Player で実行される SWF コンテンツでは使用できない機能が含まれています。

ActionScript 3.0 開発者
Adobe AIR API については、以下の 2 つのドキュメントで説明されています。

- ActionScript 3.0 開発ガイド
- Adobe Flash Platform 用 ActionSc ipt 3.0 リファレンスガイド

HTML 開発者
HTML ベースの AIR アプリケーションを作成する場合は、AIRAliases.js ファイルを通じて JavaScript で利用できる API（「JavaScript からの AIR API クラスのアクセス」を参照）について、以下のドキュメントを参照してください。

- Adobe AIR 用 HTML 開発ガイド
- HTML 開発者用 Adobe AIR API リファレンスガイド

AIR に固有の ActionScript 3.0 クラス

次の表は、Adobe AIR に固有のランタイムクラスを示しています。ブラウザーの Adobe® Flash® Player で実行されている SWF コンテンツに対してこれらのクラスを使用することはできません。

HTML 開発者
AIRAliases.js ファイルを通じて JavaScript で利用できるクラスについては、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。

クラス	ActionScript 3.0 パッケージ	追加された AIR バージョン
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
CertificateStatus	flash.security	2.0

クラス	ActionScript 3.0 パッケージ	追加された AIR バージョン
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 以前。3.7 からは削除されます
GameInputDevice	flash.ui	3.0
GameInputEvent	flash.ui	3.0
GameInputFinger	flash.ui	3.6 以前。3.7 からは削除されます
GameInputHand	flash.ui	3.6 以前。3.7 からは削除されます
Geolocation	flash.sensors	2.0
GeolocationEvent	flash.events	2.0
HTMLHistoryItem	flash.html	1.0
HTMLHost	flash.html	1.0
HTMLLoader	flash.html	1.0
HTMLPDFCapability	flash.html	1.0
HTMLSWFCapabiltiy	flash.html	2.0
HTMLUncaughtScriptExceptionEvent	flash.events	1.0

クラス	ActionScript 3.0 パッケージ	追加された AIR バージョン
HTMLWindowCreateOptions	flash.html	1.0
Icon	flash.desktop	1.0
IFilePromise	flash.desktop	2.0
ImageDecodingPolicy	flash.system	2.6
InteractiveIcon	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
NativeWindowRenderMode	flash.display	3.0
NativeWindowResize	flash.display	1.0
NativeWindowSystemChrome	flash.display	1.0
NativeWindowType	flash.display	1.0

クラス	ActionScript 3.0 パッケージ	追加された AIR バージョン
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
SQLError	flash.errors	1.0
SQLErrorEvent	flash.events	1.0
SQLErrorOperation	flash.errors	1.0
SQLEvent	flash.events	1.0
SQLIndexSchema	flash.data	1.0
SQLMode	flash.data	1.0
SQLResult	flash.data	1.0
SQLSchema	flash.data	1.0

クラス	ActionScript 3.0 パッケージ	追加された AIR バージョン
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

AIR 固有の機能を持つ Flash Player クラス

次のクラスはブラウザーで実行される SWF コンテンツでも使用できますが、AIR では新たなプロパティやメソッドが追加されています。

パッケージ	クラス	プロパティ、メソッド、またはイベント	追加された AIR バージョン
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
		orientationChange イベント	2.0
		orientationChanging イベント	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

パッケージ	クラス	プロパティ、メソッド、またはイベント	追加された AIR バージョン
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	1.0
		responseHeaders	1.0
		responseURL	1.0
	KeyboardEvent	commandKey	1.0
		controlKey	1.0

パッケージ	クラス	プロパティ、メソッド、またはイベント	追加された AIR バージョン
flash.net	FileReference	extension	1.0
		httpResponseStatus イベント	1.0
		uploadUnencoded()	1.0
	NetStream	drmAuthenticate イベント	1.0
		onDRMContentData イベント	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	httpResponseStatus イベント	1.0

パッケージ	クラス	プロパティ、メソッド、またはイベント	追加された AIR バージョン
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	2.0
		printMethod	2.0
flash.system	Capabilities	languages	1.1
	LoaderContext	allowLoadBytesCodeExecution	1.0
	Security	APPLICATION	1.0
flash.ui	KeyLocation	D_PAD	2.5

これらの新しいプロパティとメソッドの大多数は、AIR アプリケーションセキュリティサンドボックス内のコンテンツでのみ使用できます。ただし、URLRequest クラスの新しいメンバーは、その他のサンドボックス内で実行されるコンテンツでも使用できます。

ByteArray.compress() と ByteArray.uncompress() メソッドにはそれぞれ新たに algorithm パラメーターが組み込まれ、deflate 圧縮または zlib 圧縮のいずれかを選択できるようになっています。このパラメーターは、Adobe AIR で実行されているコンテンツでのみ使用できます。

AIR に固有の Flex コンポーネント

次の Adobe® Flex™ MX コンポーネントは、Adobe AIR のコンテンツの開発時に使用できます。

- FileEvent
- FileSystemComboBox

- FileSystemDataGrid
- FileSystemEnumerationMode
- FileSystemHistoryButton
- FileSystemList
- FileSystemSizeDisplayMode
- FileSystemTree
- FlexNativeMenu
- HTML
- Window
- WindowedApplication
- WindowedSystemManager

また、Flex 4 には、次の Spark AIR コンポーネントが含まれています。

- Window
- WindowedApplication

AIR Flex コンポーネントについて詳しくは、Using the Flex AIR components を参照してください。

第 4 章：AIR 開発用の Adobe Flash Platform ツール

次の Adobe Flash Platform 開発ツールを使用して、AIR アプリケーションを開発できます。

ActionScript 3.0（Flash および Flex）開発者向け：

- Adobe Flash Professional（「Adobe AIR のパブリッシュ」を参照）
- Adobe Flex 3.x および 4.x SDK（19 ページの「Flex SDK の設定」および 162 ページの「AIR 開発ツール（ADT）」を参照）
- Adobe Flash Builder（「Flash Builder を使用した AIR アプリケーションの開発」を参照）

HTML および Ajax 開発者向け：

- Adobe AIR SDK（17 ページの「AIR SDK のインストール」および 162 ページの「AIR 開発ツール（ADT）」を参照）
- Adobe Dreamweaver CS3、CS4、CS5（「Dreamweaver 用 AIR 拡張機能」を参照）

AIR SDK のインストール

Adobe AIR SDK には、アプリケーションの起動およびパッケージ化に使用する次のコマンドラインツールが用意されています。

AIR Debug Launcher（ADL） このツールを使用すると、事前に AIR アプリケーションをインストールしていなくても、AIR アプリケーションを実行できます。157 ページの「AIR Debug Launcher（ADL）」を参照してください。

AIR Development Tool（ADT） このツールを使用すると、AIR アプリケーションをパッケージ化して、配布可能なインストールパッケージにすることができます。162 ページの「AIR 開発ツール（ADT）」を参照してください。

AIR コマンドラインツールを使用するには、コンピューターに Java をインストールしておく必要があります。JRE または JDK（バージョン 1.5 以降）から Java 仮想マシンを使用できます。Java JRE と JDK は、http://java.sun.com/ で入手できます。

ADT ツールを実行するには、少なくとも 2GB のメモリがコンピューター上に必要です。

注意： エンドユーザーによる AIR アプリケーションの実行には、Java は必要ありません。

AIR SDK を使用した AIR アプリケーションの構築の概要については、33 ページの「AIR SDK を使用した初めての HTML ベースの AIR アプリケーションの作成」を参照してください。

AIR SDK のダウンロードとインストール

次の説明に従って、AIR SDK をダウンロードしてインストールできます。

Windows での AIR SDK のインストール

- AIR SDK インストールファイルをダウンロードします。
- AIR SDK は、標準的なファイルアーカイブとして配布されています。AIR をインストールするには、SDK の中身をコンピューター上のフォルダー（C:¥Program Files¥Adobe¥AIRSDK や C:¥AIRSDK など）に取り込みます。
- ADL ツールと ADT ツールは、AIR SDK 内の bin フォルダーに格納されています。PATH 環境変数に、このフォルダーへのパスを追加します。

Mac OS X での AIR SDK のインストール

- AIR SDK インストールファイルをダウンロードします。
- AIR SDK は、標準的なファイルアーカイブとして配布されています。AIR をインストールするには、SDK の中身をコンピューター上のフォルダー（/ ユーザ /<userName>/ アプリケーション /AIRSDK など）に取り込みます。
- ADL ツールと ADT ツールは、AIR SDK 内の bin フォルダーに格納されています。PATH 環境変数に、このフォルダーへのパスを追加します。

Linux での AIR SDK のインストール

- tbz2 形式の SDK を入手できます。
- SDK をインストールするには、SDK の解凍先フォルダーを作成し、tar -jxvf [AIR-SDK.tbz2 へのパス] コマンドを使用します。

AIR SDK ツールを使用した処理の開始については、「コマンドラインツールを使用した AIR アプリケーションの作成」を参照してください。

AIR SDK に含まれているもの

次の表で、AIR SDK に含まれているファイルの用途について説明します。

SDK フォルダー	ファイル／ツールの説明
bin	AIR Debug Launcher（ADL）を使用すると、事前に AIR アプリケーションのパッケージ化およびインストールをしていなくても、AIR アプリケーションを実行できます。このツールの使用方法について詳しくは、157 ページの「AIR Debug Launcher（ADL）」を参照してください。 AIR 開発ツール（ADT）では、アプリケーションを AIR ファイルにパッケージ化して配布します。このツールの使用方法については、162 ページの「AIR 開発ツール（ADT）」を参照してください。
frameworks	libs ディレクトリには AIR アプリケーションに使用できるコードライブラリが含まれます。 プロジェクトディレクトリには、コンパイル済みの SWF および SWC ライブラリのコードが含まれます。
include	include ディレクトリには、ネイティブ拡張を記述するための C 言語ヘッダーファイルが含まれます。
install	install ディレクトリには Android デバイス用の Windows USB ドライバーが含まれます（これらは、Google によって Android SDK で提供されるドライバーです）。
lib	AIR SDK ツールのサポートコードが含まれます。

SDK フォルダー	ファイル／ツールの説明
runtimes	デスクトップおよびモバイルデバイス用の AIR ランタイム。 デスクトップランタイムは、AIR アプリケーションがパッケージ化またはインストールされていないときに、AIR アプリケーションを起動するために ADL によって使用されます。 Android 用 AIR ランタイム（APK パッケージ）は、開発およびテストのために Android デバイスまたはエミュレーターにインストールできます。個別の APK パッケージはデバイスおよびエミュレーターに使用されます（Android 用のパブリック AIR ランタイムは、Android Market から使用できます）。
samples	このフォルダーには、サンプルのアプリケーション記述ファイル、シームレスインストール機能のサンプル（badge.swf）、およびデフォルトの AIR アプリケーションアイコンが含まれます。
templates	descriptor-template.xml - 各 AIR アプリケーションに必要なアプリケーション記述ファイルのテンプレートです。アプリケーション記述ファイルについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。 各リリースバージョンの AIR に関するアプリケーション記述子の XML 構造のスキーマファイルは、このフォルダーにあります。

Flex SDK の設定

Adobe® Flex™ を使用して Adobe® AIR® アプリケーションを開発するには、次の方法があります。

- Adobe AIR プロジェクトの作成と AIR アプリケーションのテスト、デバッグおよびパッケージ化を行うための統合ツールを備えた Adobe® Flash® Builder™ をダウンロードしてインストールします。21 ページの「Flex Builder での初めてのデスクトップ Flex AIR アプリケーションの作成」を参照してください。
- Adobe® Flex™ SDK をダウンロードし、使い慣れたテキストエディターとコマンドラインツールを使用して Flex AIR アプリケーションを開発できます。

Flex SDK を使用した AIR アプリケーションの構築の概要については、37 ページの「Flex SDK を使用した初めてのデスクトップ AIR アプリケーションの作成」を参照してください。

Flex SDK のインストール

コマンドラインツールを使用して AIR アプリケーションを構築するには、コンピューターに Java をインストールしておく必要があります。JRE または JDK（バージョン 1.5 以降）から Java 仮想マシンを使用できます。Java JRE と JDK は、http://java.sun.com で入手できます。

注意：エンドユーザーによる AIR アプリケーションの実行には、Java は必要ありません。

Flex SDK には、AIR アプリケーションのパッケージ化、コンパイルおよびデバッグに使用する AIR API とコマンドラインツールが用意されています。

1. Flex SDK をまだダウンロードしていない場合は、http://opensource.adobe.com/wiki/display/flexsdk/Downloads からダウンロードします。
2. フォルダー（Flex SDK など）に SDK の内容を配置します。
3. Flex SDK のファイルに AIR SDK のコンテンツを上書きコピーします。

   注意：Mac コンピューターでは、SDK フォルダーの（ディレクトリ全体ではなく）個々のファイルをコピーまたは置換します。デフォルトでは、Mac のディレクトリを同じ名前のディレクトリにコピーすると、ターゲットディレクトリの既存のファイルは削除されます。2 つのディレクトリのコンテンツは結合されません。端末ウィンドウで ditto コマンドを使用すると、AIR SDK を Flex SDK：ditto air_sdk_folder flex_sdk_folder に結合できます。

4 コマンドライン AIR ユーティリティは bin フォルダーにあります。

外部 SDK の設定

Android および iOS 用のアプリケーションを開発するには、プラットフォームの製造元から、プロビジョニングファイル、SDK、またはその他の開発ツールをダウンロードする必要があります。

Android SDK のダウンロードおよびインストールについては、「Android Developers：Installing the SDK」を参照してください。AIR 2.6 に関しては、Android SDK をダウンロードする必要はありません。この AIR SDK には、APK パッケージのインストールと起動に必要となる基本コンポーネントが含まれています。ただし、Android SDK はソフトウェアエミュレーターの作成と実行、デバイススクリーンショットの撮影など、様々な開発タスクで役立ちます。

外部 SDK は iOS 開発には必要ありません。ただし、特別な証明書とプロビジョニングプロファイルが必要になります。詳しくは、「Apple から開発ファイルを入手」を参照してください。

第 5 章：初めての AIR アプリケーションの作成

Flex Builder での初めてのデスクトップ Flex AIR アプリケーションの作成

Adobe® AIR® の機能を簡潔かつ実践的に説明するため、以下の手順に従って Adobe® Flash® Builder を使用し、SWF ファイルベースの単純な AIR アプリケーション「Hello World」を作成してパッケージ化します。

まだ実行していない場合は、Flash Builder をダウンロードしてインストールします。また、Adobe AIR の最新バージョンをダウンロードおよびインストールします。AIR は www.adobe.com/go/air_jp にあります。

AIR プロジェクトの作成

Flash Builder には、AIR アプリケーションの開発とパッケージ化用のツールが用意されています。

Flash Builder または Flex Builder で AIR アプリケーションの作成を開始するには、Flex ベースの他のアプリケーションプロジェクトの場合と同様、新しいプロジェクトを定義します。

1 Flash Builder を開きます。

2 ファイル／新規／ Flex プロジェクトを選択します。

3 プロジェクト名を AIRHelloWorld と入力します。

4 Flex では、AIR アプリケーションがアプリケーションの種類の 1 つと見なされます。種類のオプションは 2 つあります。

- Adobe® Flash® Player で実行する Web アプリケーション
- Adobe AIR で実行するデスクトップアプリケーション

アプリケーションの種類として「デスクトップ」を選択します。

5 「終了」をクリックしてプロジェクトを作成します。

作成直後の AIR プロジェクトは 2 つのファイルから成ります。メインの MXML ファイルと、アプリケーション XML ファイル（アプリケーション記述ファイル）です。後者のファイルでは、アプリケーションプロパティを指定します。

詳しくは、「Flash Builder を使用した AIR アプリケーションの開発 」を参照してください。

AIR アプリケーションコードの記述

「Hello World」アプリケーションのコードを記述するには、エディターで開かれているアプリケーションの MXML ファイル（AIRHelloWorld.mxml）を編集します（ファイルが開かれていない場合は、プロジェクトナビゲーターを使用してファイルを開きます）。

デスクトップの Flex AIR アプリケーションは、MXML の WindowedApplication タグ内に含まれています。MXML の WindowedApplication タグでは、タイトルバーや閉じるボタンなどの基本的なウィンドウコントロールを含む単純なウィンドウが作成されます。

1 WindowedApplication コンポーネントに title 属性を追加し、値 "Hello World" を割り当てます。

```
<?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 アプリケーションに Label コンポーネントを追加（WindowedApplication タグ内に記述）します。追加した Label コンポーネントの text プロパティを "Hello AIR" に設定し、中央に配置されるようにレイアウト制約を設定します。次の例を参照してください。

```
<?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 WindowedApplication の開始タグと、前の手順で入力したラベルコンポーネントタグとの間に、次のスタイルブロックを追加します。

```
<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>
```

これらのスタイル設定はアプリケーション全体に適用され、ウィンドウの背景がやや透明な灰色にレンダリングされます。

アプリケーションのコードは次のとおりです。

```
<?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>
```

次に、アプリケーションを透明にするために、アプリケーション記述子で以下のように設定を変更します。

1 Flex ナビゲーターペインで、プロジェクトのソースディレクトリにあるアプリケーション記述ファイルを参照します。例えばプロジェクト名を「AIRHelloWorld」と指定した場合、このファイル名は「AIRHelloWorld-app.xml」になります。

2 アプリケーション記述ファイルをダブルクリックして、Flash Builder で編集します。

3 XML コードで、initialWindow プロパティの systemChrome プロパティおよび transparent プロパティのコメント化した行を参照します。コメントを削除します（コメント区切り文字 "<!--" および "-->" を削除します）。

4 次に示すように、systemChrome プロパティのテキスト値を none に設定します。

```
<systemChrome>none</systemChrome>
```

5 次に示すように、transparent プロパティのテキスト値を true に設定します。

```
<transparent>true</transparent>
```

6 ファイルを保存します。

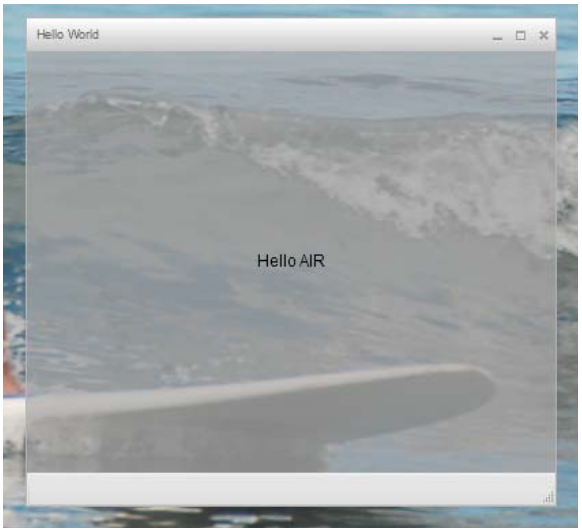
AIR アプリケーションのテスト

記述したアプリケーションコードをテストするには、デバッグモードで実行します。

1 メインツールバーのデバッグボタン をクリックします。

実行／デバッグ／ AIRHelloWorld コマンドを選択することもできます。

この結果、以下の例のような AIR アプリケーションが表示されます。



2 Label コントロールの horizontalCenter プロパティと verticalCenter プロパティが使用され、テキストがウィンドウ中央に配置されます。一般のデスクトップアプリケーションと同様に、ウィンドウを移動したりサイズを変更したりします。

注意：アプリケーションがコンパイルされない場合、コード入力時の不注意によるシンタックスエラーやスペルミスを修正します。エラーや警告は、Flash Builder の問題ビューに表示されます。

AIR アプリケーションのパッケージ化、署名および実行

これで、「Hello World」アプリケーションを配布用 AIR ファイルにパッケージ化する用意ができました。AIR ファイルは、アプリケーションファイルのアーカイブファイルです。アプリケーションファイルとは、プロジェクトの bin フォルダーに含まれているすべてのファイルです。この単純な例の場合、SWF ファイルとアプリケーション XML ファイルです。AIR パッケージをユーザーに配布すると、ユーザーはこれを使用してアプリケーションのインストールを行います。ここで必要となる手順が電子署名です。

1 アプリケーションにコンパイルエラーがなく、正常に実行できることを確認します。

2 プロジェクト／リリースビルドの書き出しを選択します。

3 AIRHelloWorld プロジェクトと AIRHelloWorld.mxml アプリケーションがプロジェクトおよびアプリケーションとしてリストに表示されていることを確認します。

4 署名済み AIR パッケージとして書き出しオプションを選択します。「次へ」をクリックします。

5 既存の電子証明書がある場合、「参照」をクリックして証明書を選択します。

6 自己署名入り電子証明書を新しく作成する必要がある場合、「作成」を選択します。

7 必要な情報を入力し、「OK」をクリックします。

8 「終了」をクリックして、AIRHelloWorld.air という AIR パッケージを生成します。

Flash Builder のプロジェクトナビゲーターから、またはファイルシステムから AIR ファイルをダブルクリックすると、アプリケーションをインストールして実行できます。

Flash Professional を使用した初めてのデスクトップ AIR アプリケーションの作成

Adobe® AIR® の仕組みを簡単に説明した実践形式のデモを体験するには、このトピックの説明に従ってください。Adobe® Flash® Professional を使用して、「Hello World」という名前の簡単な AIR アプリケーションを作成し、パッケージ化できます。

まだ実行していない場合は、www.adobe.com/go/air_jp から Adobe AIR をダウンロードしてインストールします。

Flash における Hello World アプリケーションの作成

Flash における Adobe AIR アプリケーションの作成方法は、他の FLA ファイルを作成する場合とほとんど同じです。以下に、Flash Professional を使用して簡単な Hello World アプリケーションを作成する手順を示します。

Hello World アプリケーションを作成するには

1 Flash を起動します。

2 スタートアップスクリーンで「AIR」をクリックして、Adobe AIR パブリッシュ設定が含まれている空の FLA ファイルを作成します。

3 「ツール」パネルのテキストツールを選択し、ステージの中央に静的テキストフィールド（デフォルト）を作成します。15 ～ 20 文字を入力するのに十分な幅を指定します。

4 テキストフィールドに「Hello World」と入力します。

5 ファイルに名前（例：HelloAIR）を付けて保存します。

アプリケーションのテスト

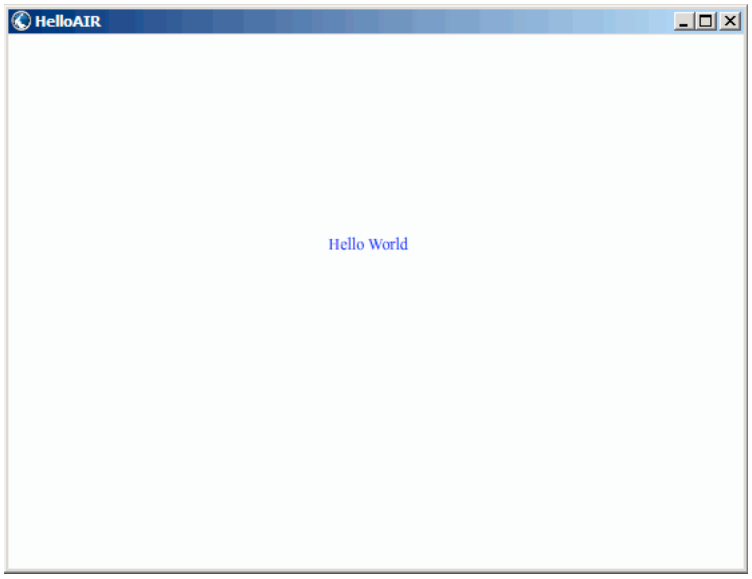
1 Ctrl + Enter キーを押すか、制御／ムービープレビュー／テストを選択して、Adobe AIR でアプリケーションをテストします。

2 ムービーのデバッグ機能を使用するには、最初に ActionScript コードをアプリケーションに追加します。これをすばやく行うには、次のように trace ステートメントを追加します。

```
trace("Running AIR application using Debug Movie");
```

3 Ctrl + Shift + Enter キーを押すか、デバッグ／ムービーのデバッグ／デバッグを選択して、ムービーのデバッグモードでアプリケーションを実行します。

Hello World アプリケーションは次の図のように表示されます。



アプリケーションのパッケージ化

1 ファイル／パブリッシュを選択します。

2 既存の電子証明書を使用して Adobe AIR パッケージに署名するか、次の手順に従って自己署名証明書を作成します。

   a 「証明書」フィールドの隣にある「新規」ボタンをクリックします。

   b 「発行者名」、「組織単位」、「組織名」、「電子メール」、「国」、「パスワード」および「パスワードの確認」に必要事項を入力します。

   c 証明書の種類を指定します。証明書の「種類」オプションは、セキュリティのレベルを表します。「1024-RSA」は安全性の低い 1024 bit キーを使用し、「2048-RSA」は安全性の高い 2048 bit キーを使用します。

   d 情報を証明書ファイルに保存します。これを行うには、「名前を付けて保存」コマンドを使用するか、「参照」ボタンをクリックしてフォルダーの場所（例：**C:/Temp/mycert.pfx**）を参照します。完了したら、「OK」をクリックします。

   e Flash に電子署名ダイアログが再び表示されます。作成した自己署名証明書のパスとファイル名が「証明書」テキストボックスに表示されます。表示されない場合は、パスとファイル名を入力するか、「参照」ボタンをクリックしてファイルを見つけて、選択します。

   f 電子署名ダイアログボックスの「パスワード」テキストフィールドに、手順 b で割り当てたパスワードと同じパスワードを入力します。Adobe AIR アプリケーションの署名について詳しくは、186 ページの「AIR ファイルへの電子署名」を参照してください。

3 アプリケーションファイルとインストーラーファイルを作成するには、「パブリッシュ」ボタンをクリックします（Flash CS4 および CS5 では、「OK」をクリックします）。AIR ファイルを作成する前に、ムービーのプレビューまたはデバッグを実行して、SWF ファイルと application.xml ファイルを作成しておく必要があります。

4 アプリケーションをインストールするには、アプリケーションを保存したフォルダーにある AIR ファイル（**application**.air）をダブルクリックします。

5 アプリケーションインストールダイアログの「インストール」ボタンをクリックします。

6 インストールの環境設定と場所設定をレビューして、「インストール後にアプリケーションを起動」チェックボックスがオンになっていることを確認します。「続行」をクリックします。

7 インストール完了のメッセージが表示されたら、「終了」ボタンをクリックします。

Flash Professional を使用した初めての Android 用 AIR アプリケーションの作成

Android 用 AIR アプリケーションを開発する場合、Android 用 Flash Professional CS5 拡張機能を Adobe Labs からダウンロードする必要があります。

また、「Android Developers：Installing the SDK」の手順に従って、Android Web サイトから Android SDK をダウンロードしてインストールする必要もあります。

プロジェクトの作成

1 Flash Professional CS5 を開きます。

2 新規 AIR for Android プロジェクトを作成します。

Flash Professional のホーム画面には、AIR for Android アプリケーションを作成するためのリンクがあります。ファイル／新規を選択して、AIR for Android テンプレートを作成することもできます。

3 ドキュメントを HelloWorld.fla として保存します。

コードの作成

このチュートリアルはコードの作成に関するものではないので、ステージ上に「Hello, World!」と書き込むテキストツールを使用します。

アプリケーションプロパティの設定

1 ファイル／ AIR Android 設定を選択します。

2 「一般」タブで、次の設定を行います。

- 出力ファイル：HelloWorld.apk
- アプリケーション名：HelloWorld
- アプリケーション ID：HelloWorld
- バージョン番号：0.0.1
- 縦横比：縦長

3 「デプロイ」タブで、次の設定を行います。

- 証明書：有効な AIR コードコードサイニング証明書を参照します。「作成」ボタンをクリックして、証明書を新規作成します（Android Marketplace を介してデプロイされる Android アプリケーションには、少なくとも 2033 年まで有効な証明書が必要です）。「パスワード」フィールドに証明書のパスワードを入力します。
- Android デプロイタイプ：デバッグ
- パブリッシュ後：両方のオプションを選択
- 必要に応じて、Android SDK の tools サブディレクトリにある ADB ツールへのパスを入力します。

4 「OK」をクリックして、Android 設定ダイアログを閉じます。

デプロイのこのステージでは、アプリケーションのアイコンまたは権限は必要ありません。多くの Android 用 AIR アプリケーションでは、保護された機能にアクセスするためにいくつかの権限が必要になります。これらの権限は、アプリケーションに不可欠な場合にのみ設定します。アプリケーションが要求する権限が多すぎると、ユーザーが利用を取り止める可能性があるからです。

5 ファイルを保存します。

Android デバイス上でのアプリケーションのパッケージ化およびインストール

1 USB デバッグがデバイス上で有効であることを確認します。USB デバッグを、Applications ／ Development の「Settings app」で有効にします。

2 USB ケーブルでデバイスとコンピューターを接続します。

3 インストールしていない場合は、Android Market に移動し、Adobe AIR をダウンロードして、AIR ランタイムをインストールします（173 ページの「ADT installRuntime コマンド」を使用してローカルで AIR をインストールすることもできます。Android デバイスおよびエミュレーターで使用するための Android パッケージは、AIR SDK に含まれています）。

4 ファイル／パブリッシュを選択します。

Flash Professional によって、APK ファイルが作成され、接続している Android デバイスにアプリケーションがインストールされて起動します。

初めての AIR for iOS アプリケーションの作成

AIR 2.6 以降、iOS 4.2 以降

iOS アプリケーションの基本機能のコード作成、ビルド、テストは、Adobe ツールのみを使用して実行できます。ただし、デバイスに iOS アプリケーションをインストールし、そのアプリケーションを配布するには、有料サービスである Apple iOS Developer Program に登録する必要があります。iOS Developer Program への登録後は、iOS Provisioning Portal にアクセスし、このポータルで Apple から次のアイテムやファイルを取得できます。これらは、デバイス上にアプリケーションをインストールして、テストやその後の配布を実行するために必要となるものです。これらのアイテムおよびファイルには次のものがあります。

- 開発用証明書および配布用証明書
- アプリケーション ID
- 開発用プロビジョニングファイルおよび配布用プロビジョニングファイル

アプリケーションコンテンツの作成

「Hello world!」というテキストを表示する SWF ファイルを作成します。このタスクは、Flash Professional、Flash Builder、または他の IDE を使用して実行できます。この例では、テキストエディターと、Flex SDK に含まれるコマンドラインの SWF コンパイラーだけを使用します。

1 アプリケーションファイルを保存するディレクトリを任意の場所に作成します。「**HelloWorld.as**」という名前のファイルを作成し、お好みのコードエディターでファイルを編集します。

2 次のコードを追加します。

```
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 amxmlc コンパイラーを使用してこのクラスをコンパイルします。

```
amxmlc HelloWorld.as
```

「**HelloWorld.swf**」という SWF ファイルが同じフォルダー内に作成されます。

注意：この例では、amxmlc が存在するディレクトリを含むように環境パス変数を設定していることを前提としています。パスの設定について詳しくは、300 ページの「PATH 環境変数」を参照してください。別の方法として、amxmlc やこの例で使用されるその他のコマンドラインツールへの完全なパスを入力することもできます。

アプリケーションのアイコンアートと起動画面アートの作成

すべての iOS アプリケーションには、iTunes アプリケーションとデバイス画面のユーザーインターフェイスに表示されるアイコンがあります。

1 プロジェクトディレクトリ内にディレクトリを作成して、icons という名前を付けます。

2 icons ディレクトリに PNG ファイルを 3 つ作成します。それぞれに Icon_29.png、Icon_57.png、Icon_512.png という名前を付けます。

3 PNG ファイルを編集して、アプリケーションに適したアートを作成します。ファイルは 29 x 29 ピクセル、57 x 57 ピクセル、512 x 512 ピクセルである必要があります。このテストでは、アートとしてべた塗りの正方形を使用します

注意：Apple App Store にアプリケーションを送信するときは、PNG 形式ではなく JPG 形式の 512 ピクセルファイルを使用します。PNG 形式のファイルは、開発版のアプリケーションをテストするときに使用します。

すべての iPhone アプリケーションで、アプリケーションが iPhone にロードされる間、起動画面イメージが表示されます。起動画面イメージは PNG ファイルで定義します。

1 メインの開発ディレクトリに、Default.png という名前の PNG ファイルを作成します（このファイルは icons サブディレクトリに入れないでください。必ず、大文字の D を使用した Default.png という名前を付けてください）。

2 幅 320 ピクセル、高さ 480 ピクセルになるようにファイルを編集します。ここでは、コンテンツは単純な白の長方形にしておきます（これは後から変更します）。

これらのグラフィックについて詳しくは、86 ページの「アプリケーションアイコン」を参照してください。

アプリケーション記述ファイルの作成

アプリケーションの基本プロパティを指定するアプリケーション記述ファイルを作成します。このタスクは、Flash Builder などの IDE やテキストエディターを使用して実行できます。

1 HelloWorld.as が保存されているプロジェクトフォルダーで、「**HelloWorld-app.xml**」という名前の XML ファイルを作成します。お好みの XML エディターでこのファイルを編集します。

2 次の XML を追加します。

```
<?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>
```

この例では単純化のため、設定可能なプロパティのうち、一部のみが設定されています。

注意：AIR 2 以前を使用している場合は、<versionNumber> エレメントではなく <version> エレメントを使用する必要があります。

3 iOS Provisioning Portal で指定したアプリケーション ID と一致するよう、アプリケーション ID を変更します（ID の先頭のランダムなバンドルシード部分は含めないでください）。

4 ADL を使用してアプリケーションをテストします。

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

デスクトップにウィンドウが開き、そこに「Hello World!」というテキストが表示されます。表示されない場合は、ソースコードおよびアプリケーション記述子に誤りがないかどうかを確認してください。

IPA ファイルのコンパイル

この時点で、ADT を使用して IPA インストーラーファイルをコンパイルできます。P12 ファイル形式による Apple 開発用証明書と秘密キー、および Apple 開発用プロビジョニングプロファイルが必要になります。

ADT ユーティリティを次のオプションを指定して実行します。keystore、storepass、provisioning-profile の各値については、独自の値に置き換えてください。

```
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
```

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。

ADT によって、プロジェクトディレクトリ内に「**HelloWorld.ipa**」という iOS アプリケーションインストーラーファイルが生成されます。IPA ファイルのコンパイルには数分かかる場合があります。

デバイスへのアプリケーションのインストール

テスト用 iOS アプリケーションをインストールするには：

1. iTunes アプリケーションを開きます。
2. このアプリケーションのプロビジョニングプロファイルを iTunes にまだ追加していない場合は、追加します。iTunes で、ファイル／ライブラリに追加を選択します。次に、ファイルタイプ mobileprovision のプロビジョニングプロファイルを選択します。

   この時点では、開発デバイスでアプリケーションをテストするために、開発用プロビジョニングプロファイルを使用します。

   後で、アプリケーションを iTunes Store に配布するときに、配布プロファイルを使用します。アプリケーションをアドホックで（iTunes Store を経由せずに複数のデバイスに）配布するには、アドホックプロビジョニングプロファイルを使用します。

   プロビジョニングプロファイルについて詳しくは、66 ページの「iOS 用の設定」を参照してください。
3. 一部のバージョンの iTunes では、同じバージョンのアプリケーションが既にインストールされている場合、アプリケーションは置き換えられません。この場合は、デバイスおよび iTunes のアプリケーションリストから、該当するアプリケーションを削除します。
4. アプリケーションの IPA ファイルをダブルクリックします。このファイルが iTunes のアプリケーションのリストに表示されます。
5. コンピューターの USB ポートにデバイスを接続します。
6. iTunes で、このデバイスの「アプリケーション」タブを確認し、インストールするアプリケーションのリストで該当するアプリケーションが選択されていることを確認します。
7. iTunes アプリケーションの左側のリストでデバイスを選択します。「同期」ボタンをクリックします。同期が完了すると、Hello World アプリケーションが iPhone に表示されます。

新しいバージョンがインストールされない場合は、デバイスおよび iTunes のアプリケーションリストからアプリケーションを削除し、この手順を再度実行します。現在インストールされているアプリケーションで、インストールしようとしているアプリケーションと同じアプリケーション ID とバージョンが使用されている場合に、この問題が発生することがあります。

起動画面イメージの編集

アプリケーションをコンパイルする前に、Default.png ファイルを作成しました（28 ページの「アプリケーションのアイコンアートと起動画面アートの作成」を参照）。この PNG ファイルは、アプリケーションのロード中に、起動画面イメージの役割を果たします。iPhone でアプリケーションをテストしたときに、起動時にこの空の画面が表示されることに気づいた方もいるでしょう。

このイメージを、アプリケーション（「Hello World!」）のスタートアップスクリーンと一致するように変更する必要があります。

1. iPhone デバイスでアプリケーションを開きます。最初の「Hello World」というテキストが表示されたら、ホームボタン（画面の下）を長押しします。ホームボタンを押したまま、電源 / スリープボタン（iPhone の上部）を押します。これでスクリーンショットが撮影され、カメラロールに送信されます。
2. iPhoto またはその他の写真転送アプリケーションから、このスクリーンショットイメージを開発コンピューターに転送します（Mac OS では、イメージキャプチャアプリケーションも使用できます）。

   スクリーンショットイメージは、次の手順で開発コンピューターにメールで送信することもできます。

   - 写真アプリケーションを開きます。
   - カメラロールを開きます。

- キャプチャしたスクリーンショットイメージを開きます。
- そのイメージをタップして、左下隅の転送（矢印）ボタンをタップします。「写真をメール」ボタンをクリックしてイメージを自分宛に送信します。

3 Default.png ファイル（開発ディレクトリ内）を、PNG 形式のスクリーンショットイメージに置き換えます。

4 アプリケーションを再コンパイルして（29 ページの「IPA ファイルのコンパイル」を参照）、デバイスに再インストールします。

これで、アプリケーションのロード時に新しいスタートアップスクリーンが使用されます。

注意：サイズが正しければ（320 x 480 ピクセル）、どのようなアートでも、Default.png ファイルとして作成できます。ただし、多くの場合には、Default.png イメージには、アプリケーションの初期状態と同じイメージを使用することをお勧めします。

Dreamweaver での初めての HTML ベースの AIR アプリケーションの作成

Adobe® AIR® 機能を簡潔かつ実践的に説明するため、以下の手順に従って、Dreamweaver® 用 Adobe® AIR® 拡張機能を使用し、HTML ベースの単純な AIR アプリケーション「Hello World」を作成してパッケージ化します。

まだ実行していない場合は、www.adobe.com/go/air_jp から Adobe AIR をダウンロードしてインストールします。

Dreamweaver 用 Adobe AIR 拡張機能のインストール手順については、「Dreamweaver 用 Adobe AIR 拡張機能のインストール」を参照してください。

必要システム構成など、拡張機能の概要については、「Dreamweaver 用 AIR 拡張機能」を参照してください。

注意：デスクトップおよび extendedDesktop プロファイルの場合、HTML ベースの AIR アプリケーションのみを開発できます。モバイルプロファイルはサポートされません。

アプリケーションファイルの準備

Adobe AIR アプリケーションには、スタートページと、Dreamweaver サイトで定義されているすべての関連ページが必要です。

1 Dreamweaver を起動し、サイトが定義済みであることを確認します。

2 ファイル／新規を選択し、ページタイプ列で「HTML」、レイアウト列で「なし」を選択し、「作成」をクリックして、新しい HTML ページを作成します。

3 新しいページで、**Hello World!** と入力します。

この例は非常に単純なものですが、必要な場合は、テキストのスタイルをニーズに合わせて設定したり、ページにコンテンツを追加したり、このスタートページを他のページにリンクしたりすることができます。

4 ページを hello_world.html という名前で保存します（ファイル／保存）。ファイルが Dreamweaver サイトに保存されていることを確認します。

Dreamweaver サイトについて詳しくは、Dreamweaver のヘルプを参照してください。

Adobe AIR アプリケーションの作成

1 Dreamweaver のドキュメントウィンドウで hello_world.html ページが開いていることを確認します（hello_world.html ページの作成手順については前の節を参照してください）。

2 サイト／Air アプリケーション設定を選択します。

Air - アプリケーションとインストーラーの設定ダイアログボックスの必要な設定の多くは自動的に設定されます。ただし、アプリケーションの最初のコンテンツ（スタートページ）は選択する必要があります。

3 「イニシャルコンテント」オプションの横にある「参照」ボタンをクリックして hello_world.html ページに移動し、そのページを選択します。

4 「電子署名」オプションの横にある「設定」ボタンをクリックします。

電子署名は、ソフトウェアの作成者がアプリケーションを作成してからコードが変更されたり壊れたりしていないことを保証するものであり、すべての Adobe AIR アプリケーションに必要です。

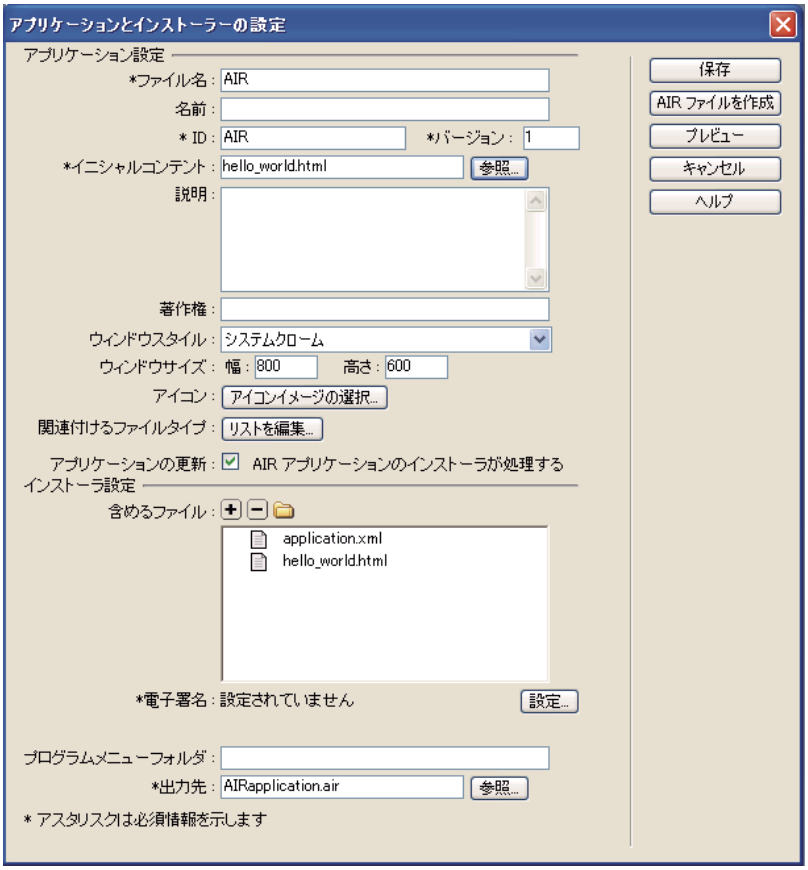
5 電子署名ダイアログボックスで、「デジタル証明書を使用して AIR パッケージに署名する」を選択して、「作成」ボタンをクリックします（電子証明書に既にアクセスしている場合は、「参照」ボタンをクリックしてその証明書を選択します）。

6 自己署名の電子証明書ダイアログボックスで必要なフィールドを入力します。名前とパスワードを入力して確認し、電子証明書のファイル名を入力する必要があります。Dreamweaver で、サイトのルートに電子証明書が保存されます。

7 「OK」をクリックして、電子署名ダイアログボックスに戻ります。

8 電子署名ダイアログボックスで、電子証明書に指定したパスワードを入力し、「OK」をクリックします。

完成した AIR - アプリケーションとインストーラーの設定ダイアログボックスは次のようになります。



ダイアログボックスのすべてのオプションおよびそれらの編集方法について詳しくは、「Dreamweaver での AIR アプリケーションの作成」を参照してください。

9 「AIR ファイルを作成」ボタンをクリックします。

Dreamweaver で Adobe AIR アプリケーションファイルが作成されて、サイトのルートフォルダーに保存されます。また、application.xml ファイルも作成されて同じ場所に保存されます。このファイルはマニフェストとして機能し、アプリケーションの様々なプロパティを定義します。

デスクトップへのアプリケーションのインストール

アプリケーションファイルを作成したので、そのファイルをデスクトップにインストールできます。

1 Adobe AIR アプリケーションファイルを Dreamweaver サイトから自分のデスクトップまたは別のデスクトップに移動します。

この手順は必須ではありません。実際には、新しいアプリケーションを Dreamweaver サイトディレクトリから直接コンピューターにインストールすることもできます。

2 アプリケーション実行可能ファイル（.air ファイル）をダブルクリックしてアプリケーションをインストールします。

Adobe AIR アプリケーションのプレビュー

AIR アプリケーションに含まれるページはいつでもプレビューできます。アプリケーションをインストールした場合の外観を確認する前にアプリケーションをパッケージ化する必要はありません。

1 Dreamweaver のドキュメントウィンドウで hello_world.html ページが開いていることを確認します。

2 ドキュメントツールバーの「ブラウザでのプレビュー / デバッグ」ボタンをクリックし、「Adobe AIR でプレビュー」を選択します。

Ctrl + Shift + F12 キー（Windows）または Cmd + Shift + F12 キー（Mac OS）を押すこともできます。

このページをプレビューすると、アプリケーションをデスクトップにインストールした後にアプリケーションのスタートページとしてユーザーに表示されるページを確認できます。

AIR SDK を使用した初めての HTML ベースの AIR アプリケーションの作成

Adobe® AIR® の機能を簡潔かつ実践的に説明するため、以下の手順に従って、HTML ベースの単純な AIR アプリケーション「Hello World」を作成してパッケージ化します。

作業を開始するには、ランタイムをインストールし、AIR SDK を設定しておく必要があります。このチュートリアルでは、**AIR Debug Launcher**（ADL）および **AIR** 開発ツール（ADT）を使用します。ADL および ADT は、コマンドラインユーティリティプログラムで、AIR SDK の bin ディレクトリにあります（17 ページの「AIR SDK のインストール」を参照）。このチュートリアルは、コマンドラインからのプログラムの実行に慣れていること、および使用しているオペレーティングシステムに必要なパス環境変数の設定方法を理解していることを前提にしています。

注意：Adobe® Dreamweaver® をご使用の場合は、31 ページの「Dreamweaver での初めての HTML ベースの AIR アプリケーションの作成」を参照してください。

注意：デスクトップおよび extendedDesktop プロファイルの場合、HTML ベースの AIR アプリケーションのみを開発できます。モバイルプロファイルはサポートされません。

プロジェクトファイルの作成

すべての HTML ベースの AIR プロジェクトには、2 つのファイルを含める必要があります。1 つはアプリケーションメタデータを指定するアプリケーション記述ファイルで、もう 1 つはトップレベルの HTML ページです。これらの必須ファイルに加えて、このプロジェクトには、AIR API クラスの便利なエイリアス変数を定義する JavaScript コードファイル AIRAliases.js が含まれます。

1. HelloWorld という名前のディレクトリを作成し、プロジェクトファイルを含めます。
2. HelloWorld-app.xml という名前の XML ファイルを作成します。
3. HelloWorld.html という名前の HTML ファイルを作成します。
4. AIR SDK のフレームワークフォルダーからプロジェクトディレクトリに AIRAliases.js をコピーします。

AIR アプリケーション記述ファイルの作成

AIR アプリケーションの構築を開始するには、次の構造の XML アプリケーション記述ファイルを作成します。

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

1. 編集用の HelloWorld-app.xml を開きます。
2. 次の AIR 名前空間属性を含め、ルートの <application> エレメントを追加します。

   <application xmlns="http://ns.adobe.com/air/application/2.7"> 名前空間の最後のセグメントである「2.7」は、アプリケーションに必要なランタイムのバージョンを指定します。

3. <id> エレメントを追加します。

   <id>examples.html.HelloWorld</id> アプリケーション ID は、（アプリケーションパッケージに署名するために使用された証明書から AIR が取得する）発行者 ID と共にアプリケーションを一意に識別します。アプリケーション ID は、インストール、プライベートなアプリケーションファイルシステムストレージディレクトリへのアクセス、プライベートな暗号化ストレージへのアクセスおよびアプリケーション間の通信に使用されます。

4. <versionNumber> エレメントを追加します。

   <versionNumber>0.1</versionNumber> インストールしているアプリケーションのバージョンをユーザーが確認できるようにします。

   注意：AIR 2 以前を使用している場合は、<versionNumber> エレメントではなく <version> エレメントを使用する必要があります。

5. <filename> エレメントを追加します。

   <filename>HelloWorld</filename> アプリケーション実行可能ファイル、インストールディレクトリおよびオペレーティングシステムでのアプリケーションに対するその他の参照に使用される名前です。

6. 次の子エレメントを含む <initialWindow> エレメントを追加して、初期アプリケーションウィンドウのプロパティを指定します。

   <content>HelloWorld.html</content> 読み込む対象となる AIR のルート HTML ファイルを識別します。

   <visible>true</visible> ウィンドウがすばやく表示されるようにします。

<width>400</width> ウィンドウの幅（ピクセル単位）を設定します。

<height>200</height> ウィンドウの高さを設定します。

7 ファイルを保存します。完全なアプリケーション記述ファイルは、次のとおりです。

```
<?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>
```

この例では、設定可能なアプリケーションプロパティのうち、一部のみが設定されます。ウィンドウクロム、ウィンドウサイズ、透明度、デフォルトのインストールディレクトリ、関連付けられているファイルの種類およびアプリケーションアイコンなどを指定できるようにする、アプリケーションプロパティの完全なセットについては、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

アプリケーションの HTML ページの作成

次に、AIR アプリケーションのメインファイルとして機能する簡単な HTML ページを作成する必要があります。

1 編集用の HelloWorld.html ファイルを開きます。次の HTML コードを追加します。

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

2 HTML の <head> セクションで、AIRAliases.js ファイルを読み込みます。

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

AIR は、HTML window オブジェクトで runtime という名前のプロパティを定義します。runtime プロパティでは、クラスの完全修飾パッケージ名を使用して、ビルトインの AIR クラスにアクセスできます。例えば、AIR File オブジェクトを作成するには、JavaScript で次のステートメントを追加します。

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

AIRAliases.js ファイルは、最も有用な AIR API の便利なエイリアスを定義します。AIRAliases.js を使用すると、File クラスへの参照を次のように短縮できます。

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

3 AIRAliases の script タグの下に、onLoad イベントを処理する JavaScript 関数を含む別の script タグを追加します。

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

appLoad() 関数は、単純に air.trace() 関数を呼び出します。ADL を使用してアプリケーションを実行すると、トレースメッセージがコマンドコンソールに出力されます。トレースステートメントは、デバッグ時に非常に役に立つ場合があります。

4 ファイルを保存します。

HelloWorld.html ファイルは次のようになります。

```
<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>
```

アプリケーションのテスト

コマンドラインからアプリケーションを実行およびテストするには、AIR Debug Launcher（ADL）ユーティリティを使用します。ADL 実行可能ファイルは、AIR SDK の bin ディレクトリにあります。まだ AIR SDK を設定していない場合は、17 ページの「AIR SDK のインストール」を参照してください。

1 コマンドコンソールまたはシェルを開きます。このプロジェクト用に作成したディレクトリに変更します。

2 次のコマンドを実行します。

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

AIR ウィンドウが開き、アプリケーションが表示されます。また、コンソールウィンドウには、air.trace() を呼び出した結果のメッセージが表示されます。

詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

AIR インストールファイルの作成

アプリケーションが正常に実行されたら、ADT ユーティリティを使用して、アプリケーションを AIR インストールファイルにパッケージ化できます。AIR インストールファイルは、ユーザーに配布できるアプリケーションファイルを格納するアーカイブファイルです。パッケージ化された AIR ファイルをインストールする前に Adobe AIR をインストールする必要があります。

アプリケーションのセキュリティを確保するには、すべての AIR インストールファイルに電子署名する必要があります。開発のために、ADT または別の証明書生成ツールを使用して基本的な自己署名入り証明書を生成できます。VeriSign や Thawte などの商用の証明機関から商用コード署名証明書を購入することもできます。ユーザーが自己署名入り AIR ファイルをインストールした場合、インストールプロセスの間、発行者は「不明」として表示されます。これは、自己署名入り証明書は AIR ファイルが作成されてから変更されていないことだけを保証するからです。なりすましの AIR ファイルの自己署名や、それをアプリケーションとして提示することを防ぐことはできません。AIR ファイルを一般にリリースする場合は、検証可能な商用証明書を強くお勧めします。AIR のセキュリティに関する問題の概要については、AIR のセキュリティ（ActionScript 開発者用）または AIR のセキュリティ（HTML 開発者用）を参照してください。

自己署名入り証明書とキーのペアの生成

❖ コマンドプロンプトから次のコマンドを入力します（ADT 実行可能ファイルは、AIR SDK の bin ディレクトリにあります）。

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

ADT は、証明書と関連の秘密キーを含む **sampleCert.pfx** という名前のキーストアファイルを生成します。

この例では、証明書に対して設定できる最小限の数の属性を使用しています。キーのタイプは、「**1024-RSA**」または「**2048-RSA**」にする必要があります（186 ページの「AIR アプリケーションへの署名」を参照してください）。

AIR インストールファイルの作成

❖ コマンドプロンプトから、次のコマンドを入力します（1 行に入力）。

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

キーストアファイルパスワードを求めるプロンプトが表示されます。

HelloWorld.air 引数は、ADT によって作成される AIR ファイルです。HelloWorld-app.xml はアプリケーション記述ファイルです。後続の引数は、アプリケーションによって使用されるファイルです。この例では 2 つのファイルだけが使用されますが、任意の数のファイルとディレクトリを含めることができます。ADT は、メインコンテンツファイル HelloWorld.html がパッケージに含まれることを確認しますが、AIRAliases.js を含めないと、アプリケーションは動作しなくなります。

AIR パッケージが作成された後に、パッケージファイルをダブルクリックすることによって、アプリケーションをインストールおよび実行できます。シェルウィンドウまたはコマンドウィンドウで、コマンドとして AIR ファイル名を入力することもできます。

次のステップ

AIR では、HTML と JavaScript コードは通常、一般的な Web ブラウザーと同じように動作します（実際、AIR は Safari Web ブラウザーで使用されているものと同じ WebKit レンダリングエンジンを使用します）。ただし、AIR で HTML アプリケーションを開発する際に理解しておく必要がある重要な違いがいくつかあります。これらの違い、およびその他の重要なトピックについて詳しくは、HTML および JavaScript のプログラミングを参照してください。

Flex SDK を使用した初めてのデスクトップ AIR アプリケーションの作成

Adobe® AIR® の機能を簡潔かつ実践的に説明するため、以下の手順に従って Flex SDK を使用し、SWF ベースの単純な AIR アプリケーション「Hello World」を作成します。このチュートリアルでは、Flex SDK に付属するコマンドラインツールを使用して AIR アプリケーションをコンパイル、テスト、およびパッケージ化する方法を説明します（Flex SDK には AIR SDK が含まれます）。

作業を開始するには、ランタイムをインストールし、Adobe® Flex™ を設定しておく必要があります。このチュートリアルでは、**AMXMLC** コンパイラー、**AIR Debug Launcher**（ADL）および **AIR** 開発ツール（ADT）を使用します。これらのプログラムは、Flex SDK の bin ディレクトリにあります（19 ページの「Flex SDK の設定」を参照してください）。

AIR アプリケーション記述ファイルの作成

この節では、次の構造を持つ XML ファイルであるアプリケーション記述子の作成方法について説明します。

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

1 HelloWorld-app.xml という名前の XML ファイルを作成し、プロジェクトディレクトリに保存します。

2 次の AIR 名前空間属性を含め、<application> エレメントを追加します。

<application xmlns="http://ns.adobe.com/air/application/2.7"> 名前空間の最後のセグメントである「2.7」は、アプリケーションに必要なランタイムのバージョンを指定します。

3 <id> エレメントを追加します。

<id>samples.flex.HelloWorld</id> アプリケーション ID は（アプリケーションパッケージに署名するために使用された証明書から AIR が派生させる）発行者 ID と共にアプリケーションを一意に識別します。推奨される形式は、「com.company.AppName」などのピリオド区切りの逆 DNS スタイルのストリングです。アプリケーション ID は、インストール、プライベートなアプリケーションファイルシステムストレージディレクトリへのアクセス、プライベートな暗号化ストレージへのアクセスおよびアプリケーション間の通信に使用されます。

4 <versionNumber> エレメントを追加します。

<versionNumber>1.0</versionNumber> インストールしているアプリケーションのバージョンをユーザーが確認できるようにします。

注意：AIR 2 以前を使用している場合は、<versionNumber> エレメントではなく <version> エレメントを使用する必要があります。

5 <filename> エレメントを追加します。

<filename>HelloWorld</filename> アプリケーション実行可能ファイル、インストールディレクトリに使用される名前であり、オペレーティングシステム内の参照に対しても同様です。

6 次の子エレメントを含む <initialWindow> エレメントを追加して、初期アプリケーションウィンドウのプロパティを指定します。

<content>HelloWorld.swf</content> 読み込む対象となる AIR のルート SWF ファイルを識別します。

<visible>true</visible> ウィンドウがすばやく表示されるようにします。

<width>400</width> ウィンドウの幅（ピクセル単位）を設定します。

<height>200</height> ウィンドウの高さを設定します。

7 ファイルを保存します。完全なアプリケーション記述ファイルは、次のとおりです。

```
<?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>
```

この例では、設定可能なアプリケーションプロパティのうち、一部のみが設定されます。ウィンドウクロム、ウィンドウサイズ、透明度、デフォルトのインストールディレクトリ、関連付けられているファイルの種類およびアプリケーションアイコンなどを指定できるようにする、アプリケーションプロパティの完全なセットについては、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

アプリケーションコードの記述

注意：SWF ベースの AIR アプリケーションは、MXML または Adobe® ActionScript® 3.0 で定義されたメインクラスを使用できます。この例では、MXML ファイルを使用してメインクラスを定義します。メインの ActionScript クラスを使用して AIR アプリケーションを作成するプロセスは、これに似ています。MXML ファイルを SWF ファイルにコンパイルするのではなく、ActionScript クラスファイルをコンパイルします。ActionScript を使用している場合は、メインクラスは、flash.display.Sprite を拡張する必要があります。

すべての Flex ベースのアプリケーションと同様に、Flex フレームワークを使用して構築された AIR アプリケーションには、メイン MXML ファイルが含まれます。デスクトップ AIR アプリケーションは、ルートエレメントとして、Application コンポーネントの代わりに **WindowedApplication** コンポーネントを使用します。WindowedApplication コンポーネントは、アプリケーションとその初期ウィンドウを制御するためのプロパティ、メソッドおよびイベントを提供します。AIR が複数ウィンドウをサポートしないプラットフォームおよびプロファイルの場合、継続して Application コンポーネントを使用できます。モバイル Flex アプリケーションでは、View コンポーネントまたは TabbedViewNavigatorApplication コンポーネントも使用できます。

次の手順に従って、Hello World アプリケーションを作成します。

1. テキストエディターを使用して、HelloWorld.mxml という名前のファイルを作成し、次の MXML コードを追加します。

```
<?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. 次に、アプリケーションに Label コンポーネントを追加（WindowedApplication タグ内に記述）します。
3. Label コンポーネントの text プロパティを "Hello AIR" に設定します。
4. 中央に配置されるようにレイアウト制約を設定します。

   次の例は、これまでに説明したコードを示します。

```
<?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>
```

アプリケーションのコンパイル

アプリケーションを実行およびデバッグする前に、amxmlc コンパイラーを使用して MXML コードを SWF ファイルにコンパイルします。amxmlc コンパイラーは、Flex SDK の bin ディレクトリにあります。必要に応じて、コンピューターの PATH 環境変数を設定して Flex SDK の bin ディレクトリを追加します。PATH 変数を設定すれば、コマンドラインでユーティリティを簡単に実行できます。

1. コマンドシェルまたは端末を開き、AIR アプリケーションのプロジェクトフォルダーに移動します。
2. 次のコマンドを入力します。

```
amxmlc HelloWorld.mxml
```

amxmlc を実行することにより、HelloWorld.swf が作成されます。これには、コンパイルされたアプリケーションのコードが格納されています。

注意：アプリケーションがコンパイルされない場合、シンタックスエラーまたはスペルミスを修正します。エラーと警告は、amxmlc コンパイラーの実行に使用するコンソールウィンドウに表示されます。

詳しくは、154 ページの「AIR 用 MXML および ActionScript ソースファイルのコンパイル」を参照してください。

アプリケーションのテスト

コマンドラインからアプリケーションの実行とテストを行うには、AIR Debug Launcher（ADL）を使用し、アプリケーション記述ファイルを使用してアプリケーションを起動します（ADL は、Flex SDK の bin ディレクトリにあります）。

❖ コマンドプロンプトから、次のコマンドを入力します。

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

次の図のような AIR アプリケーションが起動されます。

Label コントロールの horizontalCenter プロパティと verticalCenter プロパティが使用され、テキストがウィンドウ中央に配置されます。一般のデスクトップアプリケーションと同様に、ウィンドウを移動したりサイズを変更したりします。

詳しくは、157 ページの「AIR Debug Launcher（ADL）」を参照してください。

AIR インストールファイルの作成

アプリケーションが正常に実行されたら、ADT ユーティリティを使用して、アプリケーションを AIR インストールファイルにパッケージ化できます。AIR インストールファイルは、ユーザーに配布できるアプリケーションファイルを格納するアーカイブファイルです。パッケージ化された AIR ファイルをインストールする前に Adobe AIR をインストールする必要があります。

アプリケーションのセキュリティを確保するには、すべての AIR インストールファイルに電子署名する必要があります。開発のために、ADT または別の証明書生成ツールを使用して基本的な自己署名入り証明書を生成できます。民間の認証機関から市販のコードサイニング証明書を購入することもできます。ユーザーが自己署名入り AIR ファイルをインストールした場合、インストールプロセスの間、発行者は「不明」として表示されます。これは、自己署名入り証明書は AIR ファイルが作成されてから変更されていないことだけを保証するからです。なりすましの AIR ファイルの自己署名や、それをアプリケーションとして提示することを防ぐことはできません。AIR ファイルを一般にリリースする場合は、検証可能な商用証明書を強くお勧めします。AIR のセキュリティに関する問題の概要については、AIR のセキュリティ（ActionScript 開発者用）または AIR のセキュリティ（HTML 開発者用）を参照してください。

自己署名入り証明書とキーのペアの生成

❖ コマンドプロンプトから、次のコマンドを入力します（ADT 実行可能ファイルは、Flex SDK の bin ディレクトリにあります）。

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

この例では、証明書に対して設定できる最小限の数の属性を使用しています。キーのタイプは、「**1024-RSA**」または「**2048-RSA**」にする必要があります（186 ページの「AIR アプリケーションへの署名」を参照してください）。

AIR パッケージの作成

❖ コマンドプロンプトから、次のコマンドを入力します（1 行に入力）。

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

キーストアファイルパスワードを求めるプロンプトが表示されます。パスワードを入力し、Enter キーを押します。セキュリティを確保するために、パスワードの実際の文字は表示されません。

HelloWorld.air 引数は、ADT によって作成される AIR ファイルです。HelloWorld-app.xml はアプリケーション記述ファイルです。後続の引数は、アプリケーションによって使用されるファイルです。この例では、これらの 3 つのファイルだけが使用されますが、任意の数のファイルとディレクトリを含めることができます。

AIR パッケージが作成された後に、パッケージファイルをダブルクリックすることによって、アプリケーションをインストールおよび実行できます。シェルウィンドウまたはコマンドウィンドウで、コマンドとして AIR ファイル名を入力することもできます。

詳しくは、53 ページの「デスクトップ AIR インストールファイルのパッケージ化」を参照してください。

Flex SDK を使用した初めての Android 用 AIR アプリケーションの作成

まず最初に、AIR および Flex SDK をインストールして設定します。このチュートリアルでは、Flex SDK の **AMXMLC** コンパイラーおよび **AIR Debug Launcher**（ADL）、AIR SDK の **AIR** 開発ツール（ADT）を使用します。19 ページの「Flex SDK の設定」を参照してください。

また、「Android Developers：Installing the SDK」の手順に従って、Android Web サイトから Android SDK をダウンロードしてインストールする必要もあります。

注意： iPhone の開発について詳しくは、「Flash Professional CS5 での Hello World iPhone アプリケーションの作成」を参照してください。

AIR アプリケーション記述ファイルの作成

この節では、次の構造を持つ XML ファイルであるアプリケーション記述子の作成方法について説明します。

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

1 HelloWorld-app.xml という名前の XML ファイルを作成し、プロジェクトディレクトリに保存します。

2 次の AIR 名前空間属性を含め、<application> エレメントを追加します。

<application xmlns="http://ns.adobe.com/air/application/2.7"> 名前空間の最後のセグメントである「2.7」は、アプリケーションに必要なランタイムのバージョンを指定します。

3 <id> エレメントを追加します。

<id>samples.android.HelloWorld</id> アプリケーション ID は（アプリケーションパッケージに署名するために使用された証明書から AIR が派生させる）発行者 ID と共にアプリケーションを一意に識別します。推奨される形式は、「com.company.AppName」などのピリオド区切りの逆 DNS スタイルのストリングです。

4 <versionNumber> エレメントを追加します。

<versionNumber>0.0.1</versionNumber> インストールしているアプリケーションのバージョンをユーザーが確認できるようにします。

5 <filename> エレメントを追加します。

<filename>HelloWorld</filename> アプリケーション実行可能ファイル、インストールディレクトリに使用される名前であり、オペレーティングシステム内の参照に対しても同様です。

6 次の子エレメントを含む <initialWindow> エレメントを追加して、初期アプリケーションウィンドウのプロパティを指定します。

<content>HelloWorld.swf</content> 読み込む対象となる AIR のルート HTML ファイルを識別します。

7 <supportedProfiles> エレメントを追加します。

<supportedProfiles>mobileDevice</supportedProfiles> アプリケーションがモバイルプロファイル上でのみ動作することを指定します。

8 ファイルを保存します。完全なアプリケーション記述ファイルは、次のとおりです。

```
<?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>
```

この例では、設定可能なアプリケーションプロパティのうち、一部のみが設定されます。アプリケーション記述ファイル内で使用できる、ほかの設定があります。例えば、initialWindow エレメントに <fullScreen>true</fullScreen> を追加して、フルスクリーンアプリケーションを構築できます。Android 上でリモートデバッグおよびアクセス制御機能を有効にするには、Android の権限をアプリケーション記述子に追加する必要もあります。この単純なアプリケーションには権限は必要ないので、ここでは権限を追加する必要はありません。

詳しくは、70 ページの「モバイルアプリケーションプロパティの設定」を参照してください。

アプリケーションコードの記述

HelloWorld.as という名前のファイルを作成し、テキストエディターを使用して、次のコードを追加します。

```
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 );
        }
    }
}
```

アプリケーションのコンパイル

アプリケーションを実行およびデバッグする前に、amxmlc コンパイラーを使用して MXML コードを SWF ファイルにコンパイルします。amxmlc コンパイラーは、Flex SDK の bin ディレクトリにあります。必要に応じて、コンピューターの PATH 環境変数を設定して Flex SDK の bin ディレクトリを追加します。PATH 変数を設定すれば、コマンドラインでユーティリティを簡単に実行できます。

1 コマンドシェルまたは端末を開き、AIR アプリケーションのプロジェクトフォルダーに移動します。

2 次のコマンドを入力します。

```
amxmlc HelloWorld.as
```

amxmlc を実行することにより、HelloWorld.swf が作成されます。これには、コンパイルされたアプリケーションのコードが格納されています。

注意：アプリケーションがコンパイルされない場合、シンタックスエラーまたはスペルミスを修正します。エラーと警告は、amxmlc コンパイラーの実行に使用するコンソールウィンドウに表示されます。

詳しくは、154 ページの「AIR 用 MXML および ActionScript ソースファイルのコンパイル」を参照してください。

アプリケーションのテスト

コマンドラインからアプリケーションの実行とテストを行うには、AIR Debug Launcher（ADL）を使用し、アプリケーション記述ファイルを使用してアプリケーションを起動します（ADL は、AIR および Flex SDK の bin ディレクトリにあります）。

❖ コマンドプロンプトから、次のコマンドを入力します。

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

詳しくは、99 ページの「ADL を使用したデバイスシミュレーション」を参照してください。

APK パッケージファイルの作成

アプリケーションが正常に実行されたら、ADT ユーティリティを使用して、アプリケーションを APK パッケージファイルにパッケージ化できます。APK パッケージファイルはネイティブの Android アプリケーションファイル形式であり、ユーザーに配布できます。

すべての Android アプリケーションには署名が必要です。AIR ファイルとは異なり、自己署名入り証明書で通常どおり Android アプリケーションに署名する必要があります。Android オペレーティングシステムでは、アプリケーション開発者の識別の確立は試行されません。ADT で生成した証明書を使用して、Android アプリケーションに署名できます。Android Market に送信するアプリケーションに使用される証明書には、少なくとも 25 年間の有効期間が必要です。

自己署名入り証明書とキーのペアの生成

❖ コマンドプロンプトから、次のコマンドを入力します（ADT 実行可能ファイルは、Flex SDK の bin ディレクトリにあります）。

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

この例では、証明書に対して設定できる最小限の数の属性を使用しています。キーのタイプは、**1024-RSA** または **2048-RSA** にする必要があります（170 ページの「ADT certificate コマンド」を参照してください）。

AIR パッケージの作成

❖ コマンドプロンプトから、次のコマンドを入力します（1 行に入力）。

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

キーストアファイルパスワードを求めるプロンプトが表示されます。パスワードを入力し、Enter キーを押します。

詳しくは、92 ページの「モバイル AIR アプリケーションのパッケージ化」を参照してください。

AIR ランタイムのインストール

Android Market から AIR ランタイムの最新バージョンをデバイスにインストールできます。SDK に含まれているランタイムも、デバイスまたは Android エミュレーターにインストールできます。

❖ コマンドプロンプトから、次のコマンドを入力します（1 行に入力）。

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

Android SDK ディレクトリに -platformsdk フラグを設定します（tools フォルダーの親を指定します）。

ADT が、SDK に含まれている Runtime.apk をインストールします。

詳しくは、107 ページの「開発を目的とした AIR ランタイムとアプリケーションのインストール」を参照してください。

AIR アプリケーションのインストール

❖ コマンドプロンプトから、次のコマンドを入力します（1 行に入力）。

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

Android SDK ディレクトリに -platformsdk フラグを設定します（tools フォルダーの親を指定します）。

詳しくは、107 ページの「開発を目的とした AIR ランタイムとアプリケーションのインストール」を参照してください。

アプリケーションは、デバイスまたはエミュレーターのスクリーン上のアプリケーションアイコンをタップして起動できます。

第 6 章：デスクトップ用 AIR アプリケーションの開発

デスクトップ AIR アプリケーションを開発するためのワークフロー

AIR アプリケーションを開発するための基本的なワークフローは、従来のほとんどの開発モデルと同じで、コードの記述、コンパイル、テストを行い、サイクルの終了時にはインストーラーファイルへのパッケージ化を行います。

アプリケーションコードの記述には Flash、Flex および ActionScript を使用し、コンパイルには Flash Professional、Flash Builder または mxmlc および compc コマンドラインコンパイラーを使用できます。また、HTML と JavaScript を使用してアプリケーションコードを記述し、コンパイル手順をスキップすることもできます。

デスクトップ AIR アプリケーションは ADL ツールを使用してテストできます。ADL ツールでは、アプリケーションのパッケージ化およびインストールを行わずに、アプリケーションを実行できます。Flash Professional、Flash Builder、Dreamweaver 、Aptana IDE はすべて、Flash デバッガーと統合されています。また、コマンドラインから ADL を使用する場合、デバッガーツールの FDB を手動で起動することもできます。ADL 自体は、エラーの表示とステートメント出力のトレースに使用します。

すべての AIR アプリケーションはインストールファイルにパッケージ化する必要があります。以下の場合を除き、クロスプラットフォームの AIR ファイル形式をお勧めします。

- NativeProcess クラスなど、プラットフォームに依存する API にアクセスする必要がある場合。
- アプリケーションでネイティブ拡張を使用している場合。

このような場合は、AIR アプリケーションをプラットフォーム固有のネイティブインストーラーファイルとしてパッケージ化できます。

SWF ベースのアプリケーション

1. MXML または ActionScript コードを記述します。
2. アイコンビットマップファイルなど、必要なアセットを作成します。
3. アプリケーション記述子を作成します。
4. ActionScript コードをコンパイルします。
5. アプリケーションをテストします。
6. air ターゲットを使用して、AIR ファイルとしてパッケージ化し、署名します。

HTML ベースのアプリケーション

1. HTML および JavaScript コードを記述します。
2. アイコンビットマップファイルなど、必要なアセットを作成します。
3. アプリケーション記述子を作成します。
4. アプリケーションをテストします。
5. air ターゲットを使用して、AIR ファイルとしてパッケージ化し、署名します。

AIR アプリケーション用ネイティブインストーラーの作成

1 コードを記述します（ActionScript または HTML および JavaScript）。

2 アイコンビットマップファイルなど、必要なアセットを作成します。

3 extendedDesktop プロファイルを指定して、アプリケーション記述子を作成します。

4 ActionScript コードの場合はコンパイルします。

5 アプリケーションをテストします。

6 native ターゲットを使用して、各ターゲットプラットフォームでアプリケーションをパッケージ化します。

注意：ターゲットプラットフォーム用のネイティブインストーラーは、そのプラットフォーム上で作成する必要があります。例えば、Mac 上で Windows インストーラーを作成することはできません。VMWare のような仮想マシンを使用すると、同一のコンピューターハードウェア上で複数のプラットフォームを実行することができます。

キャプティブランタイムバンドルによる AIR アプリケーションの作成

1 コードを記述します（ActionScript または HTML および JavaScript）。

2 アイコンビットマップファイルなど、必要なアセットを作成します。

3 extendedDesktop プロファイルを指定して、アプリケーション記述子を作成します。

4 ActionScript コードの場合はコンパイルします。

5 アプリケーションをテストします。

6 bundle ターゲットを使用して、各ターゲットプラットフォームでアプリケーションをパッケージ化します。

7 バンドルファイルを使用してインストールプログラムを作成します（AIR SDK では、このようなインストーラーを作成するためのツールは提供されていませんが、多くのサードパーティのツールキットを使用できます）。

注意：ターゲットプラットフォーム用のバンドルは、そのプラットフォーム上で作成する必要があります。例えば、Mac 上で Windows バンドルを作成することはできません。VMWare のような仮想マシンを使用すると、同一のコンピューターハードウェア上で複数のプラットフォームを実行することができます。

デスクトップアプリケーションプロパティの設定

アプリケーション記述ファイルで基本的なアプリケーションプロパティを設定します。ここでは、デスクトップ AIR アプリケーションに関連するプロパティについて説明します。アプリケーション記述ファイルのエレメントについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

必要な AIR ランタイムのバージョン

アプリケーション記述ファイルの名前空間を使用して、アプリケーションに必要な AIR ランタイムのバージョンを指定します。

application エレメントに割り当てられている名前空間によって、アプリケーションが使用する機能の大部分が決まります。例えば、アプリケーションでは AIR 1.5 名前空間が使用されていて、ユーザーが AIR 3.0 をインストールしている場合、アプリケーションでは AIR 1.5 の動作が実現されます（その動作が AIR 3.0 で変更されている場合でも）。名前空間を変更し、アップデートを発行した場合にのみ、アプリケーションで新しい動作と機能を利用できます。セキュリティと WebKit の変更は、このポリシーの主な例外です。

ルートの application エレメントが持つ xmlns 属性を使用して、名前空間を指定します。

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

関連項目

205 ページの「application」

アプリケーション ID

いくつかの設定は、パブリッシュするアプリケーションごとに一意にする必要があります。このような一意の設定には、ID、名前、ファイル名などがあります。

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

関連項目

221 ページの「id」

216 ページの「filename」

229 ページの「name」

アプリケーションのバージョン

AIR 2.5 よりも前のバージョンの AIR では、version エレメントにアプリケーションのバージョンを指定します。任意の文字列を使用できます。AIR ランタイムはストリングを解釈しません。そのため、「2.0」は「1.0」よりも後のバージョンと見なされません。

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

AIR 2.5 以降のバージョンでは、versionNumber エレメントにアプリケーションのバージョンを指定します。version エレメントは使用されなくなりました。versionNumber の値を指定するとき、一連の数値をドットで 3 つまで区切って指定する必要があります。例えば、「0.1.2」のようにします。バージョン番号の各セグメントには、3 桁までの数字を指定できます（つまり、「999.999.999」が、許可される最大のバージョン番号になります）。番号に 3 つのセグメントをすべて含める必要はありません。「1」や「1.0」なども有効なバージョン番号です。

versionLabel エレメントを使用して、バージョンのラベルを指定することもできます。バージョンラベルを追加すると、AIR アプリケーションインストーラーのダイアログ画面などに、バージョン番号の代わりに表示されます。

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

関連項目

237 ページの「version」

237 ページの「versionLabel」

238 ページの「versionNumber」

メインウィンドウプロパティ

デスクトップで AIR によってアプリケーションが起動されると、ウィンドウが作成され、メイン SWF ファイルまたは HTML ページが読み込まれます。AIR では、このアプリケーションの初期ウィンドウの最初の外観と動作を制御するために、initialWindow エレメントの子エレメントが使用されます。

- **content** — initalWindow エレメントの子である content 内のメインアプリケーション SWF ファイル。デスクトッププロファイルで対象デバイスを指定するとき、SWF または HTML ファイルを使用できます。

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

ファイルは AIR パッケージに含める必要があります（ADT または IDE を使用します）。アプリケーション記述子内の名前を参照するだけでは、ファイルはパッケージに自動的には取り込まれません。

- **depthAndStencil** — 深度またはステンシルバッファーを使用する場合に指定します。通常、これらのバッファーは 3D コンテンツを操作する際に使用します。

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

- **height** — 初期ウィンドウの高さ。
- **maximizable** — ウィンドウを最大化するシステムクロムを表示するかどうかを指定します。
- **maxSize** — 使用できる最大サイズ。
- **minimizable** — ウィンドウを最小化するシステムクロムを表示するかどうかを指定します。
- **minSize** — 使用できる最小サイズ。
- **renderMode** — AIR 3 以降では、デスクトップアプリケーションに関して、レンダリングモードを auto、cpu、direct または gpu に設定できます。これより前のバージョンの AIR の場合、デスクトッププラットフォームでは、この設定は無視されます。実行時に renderMode の設定を変更することはできません。
  - auto — 基本的に cpu モードと同じです。
  - cpu — 表示オブジェクトがレンダリングされコピーされて、ソフトウェア内のメモリが表示されます。StageVideo は、ウィンドウがフルスクリーンモードの場合にのみ使用できます。Stage3D はソフトウェアレンダラーを使用します。
  - direct — 表示オブジェクトはランタイムソフトウェアによってレンダリングされますが、レンダリングされたフレームをコピーしてメモリを表示する処理（ブリッティング）はハードウェアによって高速化されます。StageVideo が使用できます。Stage3D は、可能であればハードウェアアクセラレーションを使用します。ウィンドウの透明化が true に設定されている場合、ウィンドウはソフトウェアによるレンダリングおよびブリッティングに「フォールバック」します。

    注意：Flash コンテンツの GPU アクセラレーションを、モバイルプラットフォーム向けの AIR で利用するには、renderMode="gpu" ではなく renderMode="direct"（Stage3D）を使用することをお勧めします。Adobe で公式にサポートおよび推奨している Stage3D ベースのフレームワークは、Starling（2D）および Away3D（3D）です。Stage3D および Starling/Away3D について詳しくは、http://gaming.adobe.com/getstarted/ を参照してください。
  - gpu — ハードウェアアクセラレーションが使用可能な場合は使用されます。
- **requestedDisplayResolution** — 高解像度画面を搭載した MacBook Pro コンピューターで標準解像度モードと高解像度モードのどちらを使用するのかを指定します。その他のすべてのプラットフォームでは、この値は無視されます。この値が **standard** の場合は、ステージの各ピクセルが、画面上の 4 ピクセルでレンダリングされます。この値が **high** の場合は、ステージの各ピクセルが、画面上の 1 つの物理的なピクセルに対応します。指定した値は、すべてのアプリケーションウィンドウで使用されます。デスクトップ AIR アプリケーションに対して（intialWindow エレメントの子としての）requestedDisplayResolution エレメントを使用することは、AIR 3.6 以降で可能です。
- **resizable** — ウィンドウサイズを変更するシステムクロムを表示するかどうかを指定します。
- **systemChrome** — 標準のオペレーティングシステムウィンドウの装飾を使用するかどうかを指定します。実行時にウィンドウの systemChrome の設定を変更することはできません。
- **title** — ウィンドウのタイトル。
- **transparent** — ウィンドウを背景とアルファブレンドするかどうかを指定します。透明化が有効な場合、ウィンドウではシステムクロムを使用できません。実行時にウィンドウの transparent の設定を変更することはできません。

- **visible** — ウィンドウの作成後すぐに表示可能にするかどうかを指定します。デフォルトでは、ウィンドウの初期状態は非表示であるので、ウィンドウが表示される前にアプリケーションでコンテンツを描画できます。
- **width** — ウィンドウの幅。
- **x** — ウィンドウの水平位置。
- **y** — ウィンドウの垂直位置。

関連項目

210 ページの「content」
212 ページの「depthAndStencil」
220 ページの「height」
228 ページの「maximizable」
228 ページの「maxSize」
229 ページの「minimizable」
229 ページの「minimizable」
229 ページの「minSize」
231 ページの「renderMode」
232 ページの「requestedDisplayResolution」
233 ページの「resizable」
235 ページの「systemChrome」
236 ページの「title」
237 ページの「transparent」
238 ページの「visible」
239 ページの「width」
239 ページの「x」
239 ページの「y」

デスクトップ機能

次のエレメントを使用して、デスクトップのインストールとアップデート機能を制御します。

- customUpdateUI — アプリケーションを更新する際に、独自のダイアログを表示できます。デフォルトの false に設定すると、標準の AIR ダイアログが使用されます。
- fileTypes — ファイルタイプを指定します。アプリケーションは、このファイルタイプを開くデフォルトのアプリケーションとして登録されます。別のアプリケーションがこのファイルタイプを開くデフォルトのアプリケーションとして指定されている場合、AIR は既存の登録をオーバーライドしません。ただし、アプリケーションは NativeApplication オブジェクトの setAsDefaultApplication() メソッドを使用して、実行時に登録をオーバーライドできます。ユーザーの許可を確認してから、既存のファイルタイプの関連づけをオーバーライドすることをお勧めします。

  注意：アプリケーションをキャプティブランタイムバンドルとしてパッケージ化する場合は（-bundle ターゲットを使用）、ファイルタイプの登録は無視されます。特定のファイルタイプを登録するには、登録を行うインストーラープログラムを作成する必要があります。

- installFolder — アプリケーションのインストール先となる標準のアプリケーションインストールフォルダーへの相対パスを指定します。この設定を使用して、カスタムフォルダー名を指定できます。また、共通フォルダー内に複数のアプリケーションをグループ化することもできます。
- programMenuFolder — Windows のすべてのプログラムメニューのメニュー階層を指定します。この設定を使用して、共通メニュー内に複数のアプリケーションをグループ化できます。メニューフォルダーを指定しない場合、アプリケーションのショートカットはメインメニューに直接追加されます。

関連項目

211 ページの「customUpdateUI」

217 ページの「fileTypes」

225 ページの「installFolder」

231 ページの「programMenuFolder」

サポートされるプロファイル

デスクトップのみに対応するアプリケーションの場合、サポートされるプロファイルリストからそのプロファイルを除外することで、別のプロファイル内のデバイスへのインストールを防ぐことができます。アプリケーションが NativeProcess クラスまたはネイティブ拡張を使用する場合、extendedDesktop プロファイルをサポートする必要があります。

supportedProfile エレメントがアプリケーション記述子の外にある場合、アプリケーションは定義済みのプロファイルすべてをサポートすると見なされます。アプリケーションを特定のプロファイルのリストに制限するには、プロファイルを空白で区切ってリストします。

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

desktop および extendedDesktop プロファイルでサポートされている ActionScript クラスの一覧については、242 ページの「様々なプロファイルの機能」を参照してください。

関連項目

234 ページの「supportedProfiles」

必要なネイティブ拡張

extendedDesktop プロファイルをサポートするアプリケーションでは、ネイティブ拡張を使用できます。

アプリケーション記述子で、AIR アプリケーションが使用するすべてのネイティブ拡張を宣言します。次の例に、2 つの必須のネイティブ拡張を指定するシンタックスを示します。

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

extensionID エレメントは、拡張記述ファイルの id エレメントと同じ値です。拡張記述ファイルは、extension.xml という名前の XML ファイルです。これは、ネイティブ拡張開発者が提供する ANE ファイル内にパッケージ化されています。

アプリケーションアイコン

デスクトップでは、アプリケーション記述子に指定したアイコンは、アプリケーションファイル、ショートカットおよびプログラムメニューのアイコンとして使用されます。アプリケーションアイコンは、16 × 16、32 × 32、48 × 48 および 128 × 128 ピクセルの PNG 画像のセットとして用意する必要があります。アプリケーション記述ファイルの icon エレメントで、アイコンファイルへのパスを指定します。

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

特定のサイズのアイコンを用意していない場合は、その次に大きなサイズが使用され、画面に合うようにサイズが調整されます。アイコンを用意しない場合は、デフォルトのシステムアイコンが使用されます。

関連項目

220 ページの「icon」

222 ページの「imageNxN」

無視される設定

デスクトップのアプリケーションは、モバイルプロファイル機能に適用されるアプリケーション設定を無視します。次の設定が無視されます。

- android
- aspectRatio
- autoOrients
- fullScreen
- iPhone
- renderMode（AIR 3 より前）
- requestedDisplayResolution
- softKeyboardBehavior

デスクトップ AIR アプリケーションのデバッグ

Flash Builder、Flash Professional、Dreamweaver などの IDE を使用してアプリケーションを開発する場合、通常、デバッグツールが組み込まれています。アプリケーションをデバッグするには、デバッグモードで起動するだけです。デバッグを直接サポートする IDE を使用していない場合、AIR Debug Launcher（ADL）と Flash Debugger（FDB）を使用してアプリケーションのデバッグをサポートできます。

関連項目

De Monsters：Monster Debugger

278 ページの「AIR HTML Introspector によるデバッグ」

ADL によるアプリケーションの実行

ADL を使用すると、パッケージ化とインストールを行わずに、AIR アプリケーションを実行できます。次の例では、アプリケーション記述ファイルをパラメーターとして ADL に渡しています（最初にアプリケーションの ActionScript コードをコンパイルする必要があります）。

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

ADL によって、トレースステートメント、実行時の例外および HTML 解析エラーが端末ウィンドウに出力されます。FDB プロセスが着信接続を待機している場合、ADL はデバッガーに接続します。

ADL を使用して、ネイティブ拡張を使用する AIR アプリケーションをデバッグすることもできます。次に、例を示します。

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

関連項目

157 ページの「AIR Debug Launcher（ADL）」

トレースステートメントの出力

ADL の実行に使用するコンソールにトレースステートメントを出力するには、trace() 関数を使用して、トレースステートメントをコードに追加します。

注意：trace() ステートメントがコンソール上に表示されない場合は、mm.cfg ファイルで ErrorRep rtingEnable または TraceOutputFileEnable を指定していないことを確認してください。このファイルのプラットフォーム別の場所について詳しくは、Editing the mm.cfg file を参照してください。

ActionScript の例：

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

JavaScript の例：

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

JavaScript コードで alert() および confirm() 関数を使用すると、アプリケーションからのデバッグメッセージを表示できます。また、シンタックスエラーがある場所の行番号や、キャッチされない JavaScript 例外もコンソールに出力されます。

注意：JavaScript の例のように air 接頭辞を使用するには、ページに AIRAliases.js ファイルを読み込む必要があります。このファイルは、AIR SDK の frameworks ディレクトリにあります。

Flash デバッガー（FDB）への接続

Flash デバッガーを使用して AIR アプリケーションをデバッグするには、FDB セッションを開始してから ADL でアプリケーションを起動します。

注意：SWF ベースの AIR アプリケーションでは、ActionScript ソースファイルのコンパイル時に -debug フラグを指定する必要があります（Flash Professional の場合は、パブリッシュ設定ダイアログボックスの「デバッグを許可」オプションを確認してください）。

1. FDB を開始します。FDB プログラムは、Flex SDK の bin ディレクトリにあります。

   コンソールに、FDB プロンプトの <fdb> が表示されます。

2. run コマンドを <fdb>run [Enter] のようにして実行します。
3. 別のコマンドまたはシェルコンソールで、アプリケーションのデバッグバージョンを開始します。

   ```
   adl myApp.xml
   ```

4. FDB コマンドを使用して、必要に応じてブレークポイントを設定します。
5. continue [Enter] のように入力します。

SWF ベースの AIR アプリケーションの場合は、ActionScript コードの実行だけがデバッガーによって制御されます。HTML ベースの AIR アプリケーションの場合は、JavaScript コードの実行だけがデバッガーによって制御されます。

デバッガーに接続せずに ADL を実行するには、-nodebug オプションを含めます。

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

FDB コマンドの基本情報を表示するには、help コマンドを実行します。

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

FDB コマンドについて詳しくは、Flex ドキュメントの「コマンドラインデバッガーコマンドの使用」を参照してください。

デスクトップ AIR インストールファイルのパッケージ化

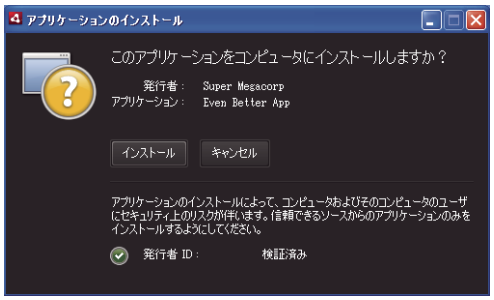
すべての AIR アプリケーションには、最低限、アプリケーション記述ファイルとメイン SWF ファイルまたはメイン HTML ファイルが必要です。アプリケーションと一緒にインストールする他のすべてのアセットも、AIR ファイルにパッケージ化する必要があります。

ここでは、SDK に付属するコマンドラインツールを使用して AIR アプリケーションをパッケージ化する方法について説明します。アドビオーサリングツールのいずれかを使用してアプリケーションをパッケージ化する方法については、次を参照してください。

- Adobe® Flex® Builder™ については、Flex Builder を使用した AIR アプリケーションのパッケージ化を参照してください。
- Adobe® Flash® Builder™ については、Flash Builder を使用した AIR アプリケーションのパッケージ化を参照してください。
- Adobe® Flash® Professional については、Adobe AIR のパブリッシュを参照してください。
- Adobe® Dreamweaver® については、Dreamweaver での AIR アプリケーションの作成を参照してください。

すべての AIR インストーラーファイルには、電子証明書を使用して署名する必要があります。AIR インストーラーは、署名を使用して、署名後にアプリケーションファイルが変更されていないかどうかを検証します。証明機関のコードサイニング証明書または自己署名入り証明書を使用できます。

信頼できる証明機関から発行された証明書を使用しているときは、該当の ID が発行者のものであることをアプリケーションのユーザーに保証します。インストールダイアログには、ID が証明機関によって確認済みであることが示されます。



信頼できる証明書によって署名されたアプリケーションのインストールの確認ダイアログ

自己署名証明書を使用している場合、ユーザーは該当の ID を署名者のものとして確認できません。また、自己署名証明書では、パッケージが変更されていないという保証も弱くなります（これは、正規のインストールファイルが、ユーザーの元に届く前に偽造ファイルと置き換えられる可能性があるからです）。インストールダイアログには、発行者の ID を確認できないことが示されます。アプリケーションをインストールするとき、ユーザーはより大きなセキュリティリスクを負うことになります。

自己署名証明書によって署名されたアプリケーションのインストールの確認ダイアログ

ADT -package コマンドを使用すると、1 つの手順で AIR ファイルをパッケージ化して署名することができます。-prepare コマンドを使用して未署名の中間パッケージを作成し、別の手順で -sign コマンドを使用してこの中間パッケージに署名することもできます。

注意：Java バージョン 1.5 以上では、PKCS12 証明書ファイルを保護するパスワードに high-ASCII（拡張 ASCII）文字を使用できません。コードサイニング証明書ファイルを作成するとき、または書き出すときは、パスワードに標準の ASCII 文字のみを使用します。

インストールパッケージへの署名時、ADT はタイムスタンプ局サーバーに自動的にアクセスして時刻を確認します。タイムスタンプ情報は、AIR ファイルに含められます。検証されたタイムスタンプを含む AIR ファイルは、後でいつでもインストールできます。ADT がタイムスタンプサーバーに接続できない場合、パッケージ化はキャンセルされます。このタイムスタンプオプションはオーバーライドできますが、タイムスタンプがない場合、インストールファイルへの署名に使用された証明書の期限が切れると、AIR アプリケーションをインストールできなくなります。

パッケージを作成して既存の AIR アプリケーションを更新する場合、そのパッケージには元のアプリケーションと同じ証明書で署名する必要がありあます。元の証明書が更新されたか過去 180 日以内に期限切れになった場合、または新しい証明書に変更する場合は、移行署名を適用できます。移行署名では、新しい証明書と古い証明書の両方でアプリケーション AIR ファイルに署名する必要があります。170 ページの「ADT migrate コマンド」の説明に従って、-migrate コマンドを使用して移行署名を適用します。

重要：元の証明書の期限が切れた後、移行署名を適用するための猶予期間は 180 日です。移行署名を適用しない場合、これまでアプリケーションを使用していたユーザーは、新しいバージョンをインストールする前に、既存のアプリケーションをアンインストールする必要があります。猶予期間が適用されるのは、アプリケーション記述子の名前空間に AIR のバージョン 1.5.3 以上が指定されているアプリケーションのみです。以前のバージョンの AIR ランタイムを対象としている場合、猶予期間はありません。

AIR 1.1 よりも前のバージョンでは、移行署名はサポートされていませんでした。移行署名を適用するには、バージョン 1.1 以降の SDK でアプリケーションをパッケージ化する必要があります。

AIR ファイルを使用してデプロイされているアプリケーションは、デスクトッププロファイルアプリケーションと呼ばれます。デスクトップファイルがアプリケーション記述ファイルによってサポートされていない場合は、AIR アプリケーションのネイティブインストーラーをパッケージ化するときに ADT を使用できません。このプロファイルを制限するには、アプリケーション記述ファイル内の supportedProfiles エレメントを使用します。241 ページの「デバイスプロファイル」および 234 ページの「supportedProfiles」を参照してください。

注意：アプリケーション記述ファイルの設定によって、AIR アプリケーションの ID とデフォルトのインストールパスが決まります。200 ページの「AIR アプリケーション記述ファイル」を参照してください。

発行者 ID

AIR 1.5.3 の時点で、発行者 ID は廃止されています。新しいアプリケーション（AIR 1.5.3 以降を使用して最初に発行されたもの）では、発行者 ID を指定する必要はありません。また、指定しないことをお勧めします。

以前のバージョンの AIR で発行されたアプリケーションを更新する場合は、アプリケーション記述ファイルで元の発行者 ID を指定する必要があります。元の発行者 ID を指定しないと、インストールされたバージョンのアプリケーションおよびアップデートバージョンが、それぞれ別のアプリケーションとして扱われます。別の ID を使用するか、publisherID タグを省略すると、ユーザーは新しいバージョンをインストールする前に以前のバージョンをアンインストールする必要があります。

元の発行者 ID を確認するには、META-INF/AIR サブディレクトリにある publisherid ファイルを探します。このディレクトリには、元のアプリケーションがインストールされています。このファイル内のストリングが 発行者 ID です。発行者 ID を手動で指定するには、アプリケーション記述ファイル内で、アプリケーション記述子の名前空間宣言に AIR 1.5.3（またはそれ以降の）ランタイムが指定されている必要があります。

AIR 1.5.3 よりも前に発行されたアプリケーション（アプリケーション記述子の名前空間で以前のバージョンの AIR が指定され、AIR 1.5.3 SDK で発行さたアプリケーション）については、発行者 ID は署名者の証明書に基づいて計算されます。この ID は、アプリケーションの ID を確認するために、アプリケーション ID と共に使用されます。発行者 ID が指定されている場合は、次の目的で使用します。

- AIR ファイルが、インストールする新しいアプリケーションではなくアップデートであることを確認する
- 暗号化されたローカルストアの暗号化キーの一部として
- アプリケーション記憶域ディレクトリのパスの一部として
- ローカル接続の接続ストリングの一部として
- AIR ブラウザー API でアプリケーションを呼び出すときに使用する識別ストリングの一部として
- OSID（カスタムインストール / アンインストールプログラムの作成時に使用）の一部として

AIR 1.5.3 よりも前の AIR については、新しい証明書または更新した証明書を使用し、移行署名に基づいてアプリケーションアップデートに署名すると、アプリケーションの発行者 ID が変更される場合があります。発行者 ID が変更されると、ID に依存している AIR の機能の動作も変わります。例えば、既存の暗号化されたローカルストアにあるデータにはアクセスできなくなります。また、Flash または AIR インスタンスでアプリケーションへのローカル接続を確立するには、接続ストリングに新しい ID を使用する必要があります。

AIR 1.5.3 以降の AIR の場合、発行者 ID は署名者の証明書に基づいていません。この ID は、publisherID タグがアプリケーション記述子に含まれている場合にのみ割り当てられます。アップデートの AIR パッケージに対して指定された発行者 ID が現在の発行者 ID と一致しない場合、アプリケーションは更新できません。

ADT によるパッケージ化

AIR ADT コマンドラインツールを使用して、AIR アプリケーションをパッケージ化できます。パッケージ化の前に、すべての ActionScript、MXML および拡張コードをコンパイルする必要があります。また、コード署名証明書が必要です。

ADT コマンドとオプションについて詳しくは、162 ページの「AIR 開発ツール（ADT）」を参照してください。

AIR パッケージの作成

AIR パッケージを作成するには、ADT パッケージコマンドを使用し、リリースビルドに対してターゲットの種類を **air** に設定します。

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

この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

アプリケーションファイルが含まれているディレクトリからコマンドを実行する必要があります。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf および icons ディレクトリです。

上に示したコマンドを実行するとき、ADT によって、キーストアパスワードの入力を求めるプロンプトが表示されます（入力するパスワードの文字が表示されないことがあります。入力が完了したら Enter キーを押します）。

AIRI ファイルからの AIR パッケージの作成

AIRI ファイルを作成して署名し、インストール可能な AIR パッケージを作成することができます。

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

デスクトップネイティブインストーラーのパッケージ化

AIR 2 以降では、ADT を使用して、AIR アプリケーションを配布するためのネイティブアプリケーションインストーラーを作成できます。例えば、Windows 用の AIR アプリケーションを配布するために、EXE インストーラーファイルを作成できます。また、Mac OS 用の AIR アプリケーションを配布するために、DMG インストーラーファイルを作成できます。AIR 2.5 および AIR 2.6 では、さらに、Linux 用の AIR アプリケーションを配布するために、DEB または RPM インストーラーファイルを作成できます。

ネイティブアプリケーションインストーラーでインストールされたアプリケーションは、拡張デスクトッププロファイルアプリケーションと呼ばれます。デスクトップ拡張プロファイルがアプリケーション記述ファイルによってサポートされていない場合は、AIR アプリケーションのネイティブインストーラーをパッケージ化するときに ADT を使用できません。このプロファイルを制限するには、アプリケーション記述ファイル内の supportedProfiles エレメントを使用します。241 ページの「デバイスプロファイル」および 234 ページの「supportedProfiles」を参照してください。

AIR アプリケーションのネイティブインストーラーバージョンは、基本的には 2 つの方法を使用して作成できます。

- アプリケーション記述ファイルおよびその他のソースファイルに基づいて、ネイティブインストーラーを作成できます（その他のソースファイルとして、SWF ファイル、HTML ファイルおよびその他のアセットが使用される場合があります）。
- AIR ファイルまたは AIRI ファイルに基づいて、ネイティブインストーラーを作成できます。

ADT は、生成するネイティブインストーラーのオペレーティングシステムと同じオペレーティングシステム上で使用する必要があります。つまり、Windows 用の EXE ファイルを作成するには、Windows 上で ADT を実行します。Mac OS 用の DMG ファイルを作成するには、Mac OS 上で ADT を実行します。Linux 用の DEB または RPG ファイルを作成するには、AIR 2.6 SDK から Linux 上で ADT を実行します。

ネイティブインストーラーを作成して AIR アプリケーションを配布すると、そのアプリケーションでは次の機能を使用できます。

- NativeProcess クラスを使用し、ネイティブプロセスを起動して操作できます。詳しくは、以下のいずれかを参照してください。
  - AIR のネイティブプロセスとの通信（ActionScript 開発者用）
  - AIR のネイティブプロセスとの通信（HTML 開発者用）
- ネイティブ拡張を使用できます。
- File.openWithDefaultApplication() メソッドを、デフォルトのシステムアプリケーションが定義されているすべてのファイルに対して呼び出して使用し、ファイルタイプに関係なく、そのファイルを開くことができます（ネイティブインス

トーラーでインストールされないアプリケーションには制限があります。詳しくは、リファレンスガイドの「File.openWithDefaultApplication()」の項目を参照してください)。

ただし、ネイティブインストーラーとしてパッケージ化するとき、アプリケーションでは AIR ファイル形式の利点のいくつかが失われます。単一ファイルでは、すべてのデスクトップコンピューターに配布できなくなります。組み込みのアップデート機能（およびアップデートフレームワーク）は動作しません。

ユーザーがネイティブインストーラーファイルをダブルクリックすると、AIR アプリケーションがインストールされます。必要なバージョンの Adobe AIR がコンピューターにインストールされていない場合は、まず、インストーラーによってその Adobe AIR がネットワークからダウンロードされ、インストールされます。適切なバージョンの Adobe AIR を取得する必要がある場合は、ネットワークに接続されていないと、インストールは失敗します。また、オペレーティングシステムが Adobe AIR 2 でサポートされていない場合も、インストールできません。

注意： インストールされたアプリケーションでファイルを実行可能にする場合には、アプリケーションをパッケージ化するときに、対象のファイルシステムでそのファイルが実行可能であることを確認してください（Mac と Linux では、必要に応じて、chmod を使用して実行可能フラグを設定できます)。

アプリケーションソースファイルからのネイティブインストーラーの作成

アプリケーションのソースファイルからネイティブインストーラーを作成するには、-package コマンドを次のシンタックスで使用します（1 行のコマンドラインで使用)。

```
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 ...] ...
```

このシンタックスは、AIR ファイルをパッケージ化するシンタックスと似ています（ネイティブインストーラーを使用しない場合)。ただし、異なる点もいくつかあります。

- -target native オプションをコマンドに追加します（-target air を指定すると、ネイティブインストーラーファイルの代わりに AIR ファイルが ADT によって生成されます)。
- ターゲット DMG または EXE ファイルを installer_file として指定します。
- オプションで、Windows では署名オプションの 2 番目のセットを追加できます。これは、シンタックスの例で [WINDOWS_INSTALLER_SIGNING_OPTIONS] として示されている部分です。Windows では、AIR ファイルの他に Windows インストーラーファイルにも署名できます。AIR ファイルに署名する場合と同じ種類の証明書および署名オプションシンタックスを使用してください（176 ページの「ADT コード署名のオプション」を参照)。同じ証明書を使用して AIR ファイルおよびインストーラーファイルに署名できます。また、異なる証明書を指定することもできます。署名された Windows インストーラーファイルを、ユーザーが Web からダウンロードすると、証明書に基づいてファイルのソースが確認されます。

-target オプション以外の ADT オプションについて詳しくは、162 ページの「AIR 開発ツール（ADT)」を参照してください。

次の例では、DMG ファイル（Mac OS 用のネイティブインストーラーファイル）を作成します。

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

次の例では、EXE ファイル（Windows 用のネイティブインストーラーファイル）を作成します。

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

次の例では、EXE ファイルを作成し、そのファイルに署名します。

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

ネイティブ拡張を使用するアプリケーション用のネイティブインストーラーの作成

アプリケーション用のソースファイルと、アプリケーションで必要なネイティブ拡張パッケージから、ネイティブインストーラーを構築できます。-package コマンドを次のシンタックスで使用します（1 行のコマンドラインで使用）。

```
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 ...] ...
```

このシンタックスは、ネイティブインストーラーをパッケージ化する際に使用するシンタックスと同じで、オプションが 2 つ追加されています。-extdir ***extension-directory*** オプションを使用して、アプリケーションで使用する ANE ファイル（ネイティブ拡張）を含むディレクトリを指定します。プライマリのコード署名証明書が、以前のバージョンで使用したものと異なる証明書である場合に、オプションの -migrate フラグと MIGRATION_SIGNING_OPTIONS パラメーターを使用して、移行署名を含むアプリケーションのアップデートに署名します。詳しくは、195 ページの「AIR アプリケーションのアップデートバージョンの署名」を参照してください。

ADT のオプションについて詳しくは、162 ページの「AIR 開発ツール（ADT）」を参照してください。

以下の例では、ネイティブ拡張を使用するアプリケーションの DMG ファイル（Mac OS 用のネイティブインストーラーファイル）が作成されます。

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

AIR ファイルまたは AIRI ファイルからのネイティブインストーラーの作成

AIR ファイルまたは AIRI ファイルに基づいて、ADT を使用してネイティブインストーラーファイルを生成できます。AIR ファイルに基づいてネイティブインストーラーを作成するには、ADT -package コマンドを次のシンタックスで使用します（1 行のコマンドラインで使用）。

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

このシンタックスは、AIR アプリケーションのソースファイルに基づいてネイティブインストーラーを作成するシンタックスと似ています。ただし、異なる点もいくつかあります。

- ソースとして、アプリケーション記述ファイルおよび AIR アプリケーション用のその他のソースファイルではなく、AIR ファイルを指定します。
- 既に署名されている場合は、AIR ファイルの署名オプションは指定しないでください。

AIRI ファイルに基づいてネイティブインストーラーを作成するには、ADT -package コマンドを次のシンタックスで使用します（1 行のコマンドラインで使用）。

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

このシンタックスは、AIR ファイルに基づいてネイティブインストーラーを作成するシンタックスと似ています。ただし、異なる点もいくつかあります。

- ソースとして、AIRI ファイルを指定します。
- ターゲット AIR アプリケーションの署名オプションを指定します。

次の例では、AIR ファイルに基づいて DMG ファイル（Mac OS 用のネイティブインストーラーファイル）を作成します。

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

次の例では、AIR ファイルに基づいて EXE ファイル（Windows 用のネイティブインストーラーファイル）を作成します。

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

次の例では、AIR ファイルに基づいて EXE ファイルを作成し、そのファイルに署名します。

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

次の例では、AIRI ファイルに基づいて DMG ファイル（Mac OS 用のネイティブインストーラーファイル）を作成します。

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

次の例では、AIRI ファイルに基づいて EXE ファイル（Windows 用のネイティブインストーラーファイル）を作成します。

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

次の例では、（AIRI ファイルに基づいて）EXE ファイルを作成し、AIR の署名およびネイティブの Windows の署名の両方を使用して署名します。

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

デスクトップコンピューター用のキャプティブランタイムバンドルのパッケージ化

キャプティブランタイムバンドルとは、アプリケーションコードおよびランタイムの専用のバージョンのを含むパッケージです。この方法でパッケージ化されたアプリケーションでは、ユーザーのコンピューター上にある他の場所にインストールされた共有ランタイムではなく、バンドルされたランタイムを使用します。

作成されるバンドルは、Windows 上のアプリケーションファイルと Mac OS 上の .app バンドルを含む自己完結型のフォルダーです。ターゲットのオペレーティングシステム用のバンドルは、そのオペレーティングシステムで実行しているときに作成する必要があります（VMWare のような仮想マシンを使用すると、複数のオペレーティングシステムを 1 台のコンピューター上で実行できます）。

このアプリケーションは、インストールせずにフォルダーまたはバンドルから実行できます。

メリット

- 自己完結型のアプリケーションが作成される
- インストールのためのインターネットアクセスが不要
- アプリケーションがランタイムアップデートから分離される
- 企業が特定のアプリケーションとランタイムの組み合わせを保証できる
- 従来のソフトウェアデプロイモデルをサポート
- 別個のランタイム再配布が不要
- NativeProcess API を使用できる
- ネイティブ拡張を使用できる
- File.openWithDefaultApplication() 関数を無制限で使用できる
- インストールせずに USB または光ディスクから実行できる

デメリット

- アドビからセキュリティパッチが発行されても、重要なセキュリティの修正が自動的に入手可能にならない
- .air ファイル形式を使用できない
- 必要に応じて独自のインストーラーの作成が必要
- AIR アップデート API およびフレームワークがサポートされていない
- Web ページから AIR アプリケーションをインストールして起動するための AIR ブラウザー API がサポートされていない
- Windows では、インストーラーによるファイル登録処理が必要
- アプリケーションディスクの容量が大きい

Windows でのキャプティブランタイムバンドルの作成

Windows 用のキャプティブランタイムバンドルを作成するには、Windows オペレーティングシステムで実行しているときにアプリケーションをパッケージ化する必要があります。アプリケーションをパッケージ化するには、次のように ADT bundle ターゲットを使用します。

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

このコマンドでは、myApp というディレクトリにバンドルが作成されます。このディレクトリには、アプリケーション用のファイルとランタイムファイルが含まれます。このフォルダーから直接プログラムを実行できます。ただし、プログラムメニューエントリ、登録ファイルのタイプ、URL スキームハンドラーを作成するには、必須のレジストリエントリを設定するインストーラープログラムを作成する必要があります。AIR SDK には、このようなインストーラーを作成するツールは含まれていませんが、サードパーティ製のツールを使用することができます。このようなツールには、有償または無償のオープンソースのインストーラーツールキットなどがあります。

Windows 上でネイティブ実行可能ファイルに署名するには、コマンドラインで、-target bundle エントリの後に第 2 の署名オプションセットを指定します。この署名オプションでは、Windows のネイティブシグネチャを適用する際に使用する秘密キーおよび関連する証明書を指定します（通常は、AIR コード署名証明書が使用されます）。プライマリ実行可能ファイルのみが署名されます。アプリケーションと共にパッケージ化されているその他の実行可能ファイルは、このプロセスでは署名されません。

ファイルタイプの関連付け

Windows でパブリックまたはカスタムのファイルタイプにアプリケーションを関連付けるには、インストーラープログラムで適切なレジストリエントリを設定する必要があります。ファイルタイプは、アプリケーション記述ファイルの fileTypes エレメントにも含める必要があります。

Windows のファイルタイプについて詳しくは、MSDN ライブラリの「File Types and File Associations」を参照してください。

URI ハンドラーの登録

アプリケーションで、特定の URI スキームを使用して URL の起動を処理するには、インストーラーで必要なレジストリエントリを設定する必要があります。

URI スキームを処理するようにアプリケーションを登録する方法について詳しくは、MSDN ライブラリの「Registering an Application to a URL Protocol」を参照してください。

Mac OS X でのキャプティブランタイムバンドルの作成

Mac OS X 用のキャプティブランタイムバンドルを作成するには、Macintosh オペレーティングシステムで実行しているときにアプリケーションをパッケージ化する必要があります。アプリケーションをパッケージ化するには、次のように ADT bundle ターゲットを使用します。

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

このコマンドでは、myApp.app というアプリケーションバンドルが作成されます。このバンドルには、アプリケーション用のファイルと、ランタイムファイルが含まれます。アプリケーションを実行するには、myApp.app アイコンをダブルクリックします。また、インストールするには、Applications フォルダーなど適切な場所に myApp.app をドラッグします。ただし、ファイルタイプや URI スキームハンドラーを登録するには、アプリケーションパッケージ内のプロパティリストファイルを編集する必要があります。

配布のために、ディスクイメージファイル（.dmg）を作成できます。Adobe AIR SDK では、キャプティブランタイムバンドル用の dmg ファイルを作成するためのツールは提供されていません。

ファイルタイプの関連付け
Mac OS X でパブリックまたはカスタムのファイルタイプにアプリケーションを関連付けるには、バンドル内の info.plist ファイルを編集して CFBundleDocumentTypes プロパティを設定する必要があります。Mac OS X Developer Library、「Information Property List Key Reference」の「CFBundleURLTypes」を参照してください。

URI ハンドラーの登録
アプリケーションで、特定の URI スキームを使用して URL の起動を処理するには、バンドル内の info.plist ファイルを編集して CFBundleURLTypes プロパティを設定する必要があります。Mac OS X Developer Library、「Information Property List Key Reference」の「CFBundleDocumentTypes」を参照してください。

デスクトップコンピューター向けの AIR パッケージの配布

AIR アプリケーションは AIR パッケージとして配布できます。このパッケージには、アプリケーションコードとすべてのアセットが含まれています。ダウンロード、電子メール、または CD-ROM のような物理メディアなど、一般的な方法を使用して、このパッケージを配布できます。アプリケーションをインストールするには、AIR ファイルをダブルクリックします。AIR ブラウザー API（Web ベースの ActionScript ライブラリ）を使用すると、Web ページのリンクを 1 度クリックして AIR アプリケーションをインストールできます（必要に応じて Adobe® AIR® もインストールできます）。

AIR アプリケーションは、ネイティブインストーラー（つまり、Windows では EXE ファイル、Mac では DMG ファイル、Linux では DEB または RPM ファイル）としてパッケージ化および配布できます。ネイティブインストールパッケージは、関連するプラットフォームの規定に従って、配布およびインストールできます。アプリケーションをネイティブパッケージとして配布する場合、AIR ファイルフォーマットの利点のいくつかが失われます。つまり、ほとんどのプラットフォームでは単一のインストールファイルを使用できなくなり、AIR アップデートフレームワークやブラウザー API も使用できなくなります。

デスクトップでの AIR アプリケーションのインストールと実行

AIR ファイルは、受信者に送信するだけです。例えば、AIR ファイルを電子メールの添付ファイルまたは Web ページのリンクとして送信できます。

AIR アプリケーションをダウンロードしたら、次の手順に従ってインストールします。

1 AIR ファイルをダブルクリックします。

Adobe AIR がコンピューターに既にインストールされている必要があります。

2 インストールウィンドウで、デフォルトの設定を選択した状態のまま、「続行」をクリックします。

Windows では、AIR が次の動作を自動的に行います。

- Program Files ディレクトリにアプリケーションをインストールします。
- アプリケーションのデスクトップショートカットを作成します。
- スタートメニューショートカットを作成します。
- 「プログラムの追加と削除」コントロールパネルにアプリケーションのエントリを追加します。

Mac OS では、デフォルトでアプリケーションが Applications ディレクトリに追加されます。

アプリケーションが既にインストールされている場合、インストーラーによって、アプリケーションの既存のバージョンを開くか、ダウンロードした AIR ファイルのバージョンにアップデートするかの選択肢が示されます。インストーラーでは、AIR ファイルのアプリケーション ID と発行者 ID を使用してアプリケーションを識別します。

3 インストールが完了したら、「完了」をクリックします。

Mac OS でアプリケーションのアップデートバージョンをインストールするには、アプリケーションディレクトリにインストールするための適切なシステム権限が必要です。Windows および Linux では、管理権限が必要です。

アプリケーションは、ActionScript または JavaScript を介して新しいバージョンをインストールすることもできます。詳しくは、254 ページの「AIR アプリケーションのアップデート」を参照してください。

AIR アプリケーションがインストールされたら、他のデスクトップアプリケーションと同様に、アプリケーションアイコンをダブルクリックするだけで実行できます。

- Windows では、アプリケーションのアイコン（デスクトップまたはフォルダーのいずれかにインストールされます）をダブルクリックするか、スタートメニューからアプリケーションを選択します。
- Linux では、アプリケーションのアイコン（デスクトップまたはフォルダーのいずれかにインストールされます）をダブルクリックするか、アプリケーションメニューからアプリケーションを選択します。
- Mac OS では、インストール先のフォルダーでアプリケーションをダブルクリックします。デフォルトのインストールディレクトリは、/ アプリケーションディレクトリです。

注意：AIR 2.6 以前用に開発された AIR アプリケーションのみ Linux 上にインストールできます。

AIR シームレスインストール機能を使用すると、ユーザーは Web ページのリンクをクリックすることで AIR アプリケーションをインストールできます。AIR ブラウザー呼び出し機能では、Web ページのリンクをクリックすることでインストール済みの AIR アプリケーションを実行できます。これらの機能については、次の節で説明します。

Web ページからのデスクトップ AIR アプリケーションのインストールと実行

AIR ブラウザー API を使用すると、Web ページから AIR アプリケーションをインストールおよび実行できます。AIR ブラウザー API は、アドビが管理する SWF ライブラリの **air.swf** で提供されます。AIR SDK にはサンプルの「バッジ」アプリケーションが含まれており、このライブラリを使用して AIR アプリケーション（必要に応じてランタイムも）をインストール、更新、および起動します。付属するサンプルバッジを変更するか、オンラインの air.swf ライブラリを直接使用する独自のバッジ Web アプリケーションを作成できます。

AIR アプリケーションは、Web ページのバッジを使用してインストールできます。ただし、Web のバッジで起動できるのは、アプリケーション記述ファイルに <allowBrowserInvocation>true</allowBrowserInvocation> エレメントを含むアプリケーションのみです。

関連項目

246 ページの「AIR.SWF ブラウザー API」

企業でのデスクトップコンピューターへのデプロイ

IT 管理者は、標準のデスクトップデプロイツールを使用して Adobe AIR ランタイムと AIR アプリケーションのサイレントインストールを行うことができます。IT 管理者は次のことができます。

- Microsoft SMS、IBM Tivoli またはブートストラッパーを使用したサイレントインストールが可能なデプロイツールによる Adobe AIR ランタイムのサイレントインストール
- ランタイムのデプロイに使用したツールと同じツールによる AIR アプリケーションのサイレントインストール

詳しくは、『**Adobe AIR 管理ガイド**』（**http://www.adobe.com/go/learn_air_admin_guide_jp**）を参照してください。

デスクトップコンピューターのインストールログ

インストールログは、AIR ランタイム自体または AIR アプリケーションのインストール時に記録されます。ログファイルの内容は、インストールまたは更新の際に発生した問題の原因を突き止めるのに役立ちます。

ログファイルは次の場所に作成されます。

- Mac：標準のシステムログ（/private/var/log/system.log）

  Mac システムログを表示するには、コンソールアプリケーションを開きます（通常、ユーティリティフォルダーにあります）。
- Windows XP：C:¥Documents and Settings¥<username>¥Local Settings¥Application Data¥Adobe¥AIR¥logs¥Install.log
- Windows Vista、Windows 7：C:¥Users¥<username>¥AppData¥Local¥Adobe¥AIR¥logs¥Install.log
- Linux：/home/<username>/.appdata/Adobe/AIR/Logs/Install.log

注意：これらのログファイルは、AIR 2 より前のバージョンの AIR では作成されませんでした。

第 7 章：モバイルデバイス向けの AIR アプリケーションの開発

AIR アプリケーションは、モバイルデバイス上にネイティブアプリケーションとしてデプロイされます。AIR ファイル形式ではなく、デバイスのアプリケーション形式が使用されます。現在、AIR は Android APK パッケージおよび iOS IPA パッケージをサポートしています。リリースバージョンのアプリケーションパッケージを作成したら、標準のプラットフォームメカニズムを介してアプリケーションを配布できます。通常、標準のプラットフォームメカニズムは Android の場合は Android Market で、iOS の場合は Apple App Store です。

AIR SDK、Flash Professional、Flash Builder または他の ActionScript 開発ツールを使用して、モバイルデバイス用 AIR アプリケーションを構築できます。HTML ベースのモバイル AIR アプリケーションは現在サポートされていません。

注意：RIM（Research In Motion）BlackBerry Playbook には、AIR 開発のために独自の SDK が用意されています。Playbook の開発について詳しくは、RIM: BlackBerry Tablet OS Development を参照してください。

注意：このドキュメントでは、AIR 2.6 SDK 以降を使用した iOS アプリケーションの開 発方法について説明します。AIR 2.6 以降で作成されたアプリケーションは、iOS 4 以降を実行する iPhone 3GS、iPhone 4、iPad の各デバイスにインストールできます。iOS の以前のバージョン用の AIR アプリケーションを開発するには、iPhone アプリケーションの構築の説明に従って、AIR 2 Packager for iPhone を使用する必要があります。

プライバシーのベストプラクティスについて詳細は、Adobe AIR SDK プライバシーガイドを参照してください。

AIR アプリケーションを実行するための完全な必要システム構成については、Adobe AIR 必要システム構成を参照してください。

開発環境の設定

モバイルプラットフォームの場合は、通常の AIR、Flex および Flash 開発環境の設定に加えて追加の設定が必要です（基本の AIR 開発環境の設定について詳しくは、17 ページの「AIR 開発用の Adobe Flash Platform ツール」を参照してください）。

Android 用の設定

通常、AIR 2.6 以降では、Android 用に特別な設定は必要ありません。AIR SDK には、Android ADB ツールが含まれています（lib/android/bin フォルダー）。AIR SDK では ADB ツールを使用して、デバイス上でのアプリケーションパッケージのインストール、アンインストール、および実行を行います。また、ADB を使用してシステムログを表示することもできます。Android エミュレーターを作成し実行するには、個別の Android SDK をダウンロードする必要があります。

アプリケーション記述子内の <manifestAdditions> エレメントにアプリケーションが追加するエレメントが、現在のバージョンの AIR で有効と認識されない場合は、新しいバージョンの Android SDK をインストールする必要があります。SDK のファイルパスに、AIR_ANDROID_SDK_HOME 環境変数または -platformsdk コマンドラインパラメーターを設定します。AIR パッケージングツールである ADT では、この SDK を使用して <manifestAdditions> エレメント内のエントリを検証します。

AIR 2.5 では、Google から個別の Android SDK のコピーをダウンロードする必要があります。Android SDK フォルダーを参照するよう、AIR_ANDROID_SDK_HOME 環境変数を設定できます。この環境変数を設定しない場合は、ADT コマンドラインの -platformsdk 引数で、Android SDK へのパスを指定する必要があります。

関連項目

184 ページの「ADT 環境変数」

300 ページの「PATH 環境変数」

iOS 用の設定

デバイス上で iOS アプリケーションをインストールしテストして、そのアプリケーションを配布するには、有料サービスである Apple iOS Developer Program に登録する必要があります。iOS Developer Program への登録後は、iOS Provisioning Portal にアクセスし、このポータルで Apple から次のアイテムやファイルを取得できます。これらは、デバイス上にアプリケーションをインストールして、テストやその後の配布を実行するために必要となるものです。これらのアイテムおよびファイルには次のものがあります。

- 開発用証明書および配布用証明書
- アプリケーション ID
- 開発用プロビジョニングファイルおよび配布用プロビジョニングファイル

モバイルアプリケーションのデザイン上の考慮事項

モバイルデバイスの動作状況や物理的特性は、慎重なコーディングとデザインを必要とします。例えば、可能な限り高速で実行されるようにコードを簡素化することが欠かせません。コードの最適化には、当然限界があります。デバイスの制限内で動作するインテリジェントなデザインを採用することも、ビジュアル表示がレンダリングシステムの処理能力を上回らないようにするのに役立ちます。

コード

コードの実行を高速化することは常に有用ですが、それと同時に、多くのモバイルデバイスのように低速なプロセッサーでは、無駄のないコードにすることで、書き込みにかかる時間が減ることによるメリットが大きくなります。さらに、モバイルデバイスは、ほとんど常に電池で動作します。同じ結果を少ない操作で実現すれば、消費電力を節約できます。

デザイン

アプリケーションのユーザーエクスペリエンスをデザインする場合、小さい画面サイズ、タッチスクリーン操作モード、また、モバイルユーザー環境が常に変化することを考慮する必要があります。

コードとデザイン

アプリケーションがアニメーションを使用する場合、レンダリングを最適化することは非常に重要です。ただし、多くの場合、コードを最適化するだけでは不十分です。コードが効率的にレンダリングできるように、アプリケーションのビジュアル表示をデザインする必要があります。

重要な最適化テクニックについては、『Flash Platform のパフォーマンスの最適化』ガイドで説明しています。このガイドで説明しているテクニックは、すべての Flash コンテンツおよび AIR コンテンツに適用されますが、モバイルデバイス上で適切に動作するアプリケーションの開発に不可欠です。

- 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

アプリケーションのライフサイクル

アプリケーションが別のアプリケーションへのフォーカスを失うと、AIR は、フレームレートを 4 フレーム / 秒に落とし、グラフィックのレンダリングを停止します。このフレームレートを下回ると、ネットワークのストリーミングおよびソケット接続は中断しやすくなります。アプリケーションでこのような接続を使用しない場合は、フレームレートをさらに低くすることができます。

適切な場合は、オーディオの再生を停止し、そのジオロケーションおよび加速度センサーのリスナーを削除します。AIR NativeApplication オブジェクトは、アクティブ化イベントおよび非アクティブ化イベントを送出します。これらのイベント使用して、アクティブ状態とバックグラウンド状態間の移行を管理します。

多くのモバイルオペレーティングシステムは、警告せずにバックグラウンドアプリケーションを停止します。アプリケーションの状態を頻繁に保存することで、アプリケーションがバックグラウンド状態からアクティブ状態に戻るとき、または新たに起動するときに、適切な状態に復元できます。

情報密度

モバイルデバイスの画面をデスクトップの画面と比較すると、物理的なサイズは小さくても、ピクセル密度（ppi）は高くなります。モバイルデバイス画面上では、同じフォントサイズの文字が、デスクトップコンピューター画面上よりも小さく表示されます。読みやすくするために、大きいフォントの使用が必要となる場合も少なくありません。一般的に、14 ポイントが、読み取りやすい最小のフォントサイズです。

モバイルデバイスは、多くの場合、移動中や、照明条件が悪い中で使用されます。画面上でどれだけ多くの情報を現実的に読みやすく表示できるかを検討します。モバイルデバイスに表示できる情報量は、デスクトップで同じピクセルサイズの画面に表示するよりも少なくなる場合があります。

ユーザーが画面にタッチするときに、指や手によって画面の一部が隠れてしまうことも考慮します。瞬間的なタッチではなく、ユーザーが一定時間触って操作するインタラクティブなエレメントは、画面の横および下部に配置します。

テキストの入力

多くのデバイスでは、テキスト入力に仮想キーボードを使用します。仮想キーボードは画面の一部を占め、しばしば邪魔になります。ソフトキー以外では、キーボードイベントの使用を避けてください。

入力テキストフィールドの代わりとなる機能を実装することを検討してください。例えば、ユーザー入力が数値の場合、テキストフィールドは必要ありません。値を増減する 2 つのボタンを代わりに使うことができます。

ソフトキー

モバイルデバイスには多数のソフトキーが用意されています。ソフトキーは、様々な機能をプログラム可能なボタンです。アプリケーションにおけるこれらのキーの設定については、プラットフォームの規則に従ってください。

画面の向きの変更

モバイルコンテンツは、縦長モードまたは横長モードで表示できます。アプリケーションが画面の向きの変更にどのように対応するかを検討します。詳しくは、「ステージの方向」を参照してください。

画面の暗転

AIR は、ビデオの再生中に画面が暗転するのを自動的には回避しません。AIR NativeApplication オブジェクトの systemIdleMode プロパティを使用して、デバイスが省電力モードに入るかどうかを制御できます（プラットフォームによっては、この機能を動作させるために、適切な権限を要求することが必要になります）。

着信電話

AIR ランタイムは、電話の発着時にオーディオを自動的にミュートします。Android では、バックグラウンドで実行中のアプリケーションがオーディオを再生する場合、アプリケーション記述子で Android の READ_PHONE_STATE 権限を設定してください。この設定を行わないと、Android のランタイムは電話の受信を検出してオーディオを自動的にミュートすることができなくなります。76 ページの「Android 権限」を参照してください。

ヒットターゲット

ユーザーがタップするボタンや他のユーザーインターフェイスをデザインする際には、ヒットターゲットのサイズを検討します。タッチスクリーン上で指で簡単に起動できるように、十分な大きさにする必要があります。また、ターゲット間に十分なスペースも確保します。ヒットターゲット領域は、一般的な高 dpi の電話画面の場合、両側に約 44 ～ 57 ピクセルで設定します。

アプリケーションパッケージのインストールサイズ

モバイルデバイスでは、通常、アプリケーションとデータのインストール用の記憶領域の容量は、デスクトップコンピューターよりも大幅に少なくなります。使用しないアセットおよびライブラリを削除して、パッケージサイズを最小化します。

Android では、アプリケーションのインストール時に、アプリケーションパッケージは個別のファイルに抽出されません。その代わり、アクセスされたときに、アセットは一時的な記憶領域に解凍されます。解凍されたアセットが使用する記憶領域を最小化するため、アセットが完全にロードされたら、ファイルと URL ストリームを完全に閉じます。

ファイルシステムでのアクセス

モバイルオペレーティングシステムに応じて異なるファイルシステムの制約が設けられており、これらの制約はデスクトップオペレーティングシステムによる制約とは異なる場合があります。このため、ファイルとデータの適切な保存場所はプラットフォームによって異なります。

AIR File クラスによって提供される共通ディレクトリへのショートカットが常に利用できるとは限りません。これは、様々なファイルシステムが存在するからです。次の表に、Android および iOS で使用できるショートカットを示します。

	Android	**iOS**
File.applicationDirectory	（ネイティブパスではなく）URL 経由での読み取り専用	読み取り専用
File.applicationStorageDirectory	使用可	使用可
File.cacheDirectory	使用可	使用可
File.desktopDirectory	SD カードのルート	使用不可
File.documentsDirectory	SD カードのルート	使用可
File.userDirectory	SD カードのルート	使用不可
File.createTempDirectory()	使用可	使用可
File.createTempFile()	使用可	使用可

iOS アプリケーションに対する Apple のガイドラインでは、様々な状況でファイルに使用すべき記憶域の場所について、具体的なルールが定められています。例えば、あるガイドラインには、ユーザーが入力したデータや、それ以外の再生成や再ダウンロードが不可能なデータを含むファイルのみを、リモートバックアップ用に指定したディレクトリに格納すべきであると記述されています。ファイルのバックアップとキャッシュに関する Apple のガイドラインに従う方法について詳しくは、「ファイルのバックアップとキャッシュの制御」を参照してください。

UI コンポーネント

Adobe は Flex フレームワークのモバイル最適化バージョンを開発しました。詳しくは、「Flex および Flash Builder を使用したモバイルアプリケーションの開発」を参照してください。

モバイルアプリケーション用のコミュニティコンポーネントプロジェクトも使用できます。これには、以下が含まれます。

- Josh Tynjala 氏の Starling 向けの Feathers UI コントロール
- Derrick Grigg 氏のスキン機能のある Minimal Comps
- Todd Anderson 氏の as3flobile コンポーネント

Stage 3D によって高速化されるグラフィックレンダリング

AIR 3.2 以降のモバイル用 AIR では、Stage 3D によって高速化されるグラフィックレンダリングがサポートされます。Stage3D の ActionScript API は、低レベルの GPU アクセラレーション API のセットであり、これによって 2D および 3D の高度な機能が実現されます。この低レベルの API により、開発者は GPU のハードウェアアクセラレーションを柔軟に利用でき、パフォーマンスを大きく向上させることができます。また、Stage3D ActionScript API をサポートするゲーム用エンジンを使用することもできます。

詳しくは、Gaming engines, 3D, and Stage 3D を参照してください。

ビデオのスムージング

パフォーマンス向上のために、AIR ではビデオのスムージングが無効になります。

ネイティブ機能

AIR 3.0 以降

多くのモバイルプラットフォームで、現時点での標準 AIR API ではアクセスできない機能が用意されています。AIR 3 以降では、独自のネイティブコードライブラリを使用して AIR を拡張できます。そのようなネイティブ拡張ライブラリでは、オペレーティングシステムから利用できる機能や、特定のデバイスに固有の機能にもアクセスできます。ネイティブ拡張は、iOS では C 言語で、Android では Java か C 言語で記述できます。ネイティブ拡張の開発については、「Adobe AIR 用ネイティブ拡張の概要」を参照してください。

モバイルデバイス向けの AIR アプリケーションを作成するためのワークフロー

モバイル（またはその他の）デバイス用に AIR アプリケーションを作成するためのワークフローは、一般的に、デスクトップアプリケーションを作成するためのワークフローと非常に類似しています。ワークフローの主要な相違点は、アプリケーションのパッケージ化、デバッグ、インストールを行う場合に明確になります。例えば、AIR for Android アプリケーションは、AIR パッケージ形式ではなく、ネイティブの Android APK パッケージ形式を使用します。したがって、これらのアプリケーションは、Android の標準のインストールメカニズムとアップデートメカニズムを使用します。

AIR for Android

次の手順は、Android 用 AIR アプリケーションを開発する場合の標準的な手順です。

- ActionScript または MXML コードを作成します。

- AIR アプリケーション記述ファイルを作成します（2.5 以降の名前空間を使用）。
- アプリケーションをコンパイルします。
- アプリケーションを Android パッケージ（.apk）としてパッケージ化します。
- デバイスまたは Android エミュレーターに AIR ランタイムをインストールします（外部ランタイムを使用している場合、キャプティブランタイムはデフォルトで AIR 3.7 以上です）。
- デバイス（または Android エミュレーター）にアプリケーションをインストールします。
- デバイス上でアプリケーションを起動します。

Adobe Flash Builder、Adobe Flash Professional CS5 またはコマンドラインツールを使用して、これらの手順を実行できます。

AIR アプリケーションを作成し、APK ファイルとしてパッケージ化したら、Android Market に送信したり、他の方法で配布できます。

AIR for iOS

次の手順は、AIR for iOS アプリケーションを開発する場合の標準的な手順です。

- iTunes をインストールします。
- Apple iOS Provisioning Portal で必要な開発用ファイルと ID を生成します。これらのアイテムは次のとおりです。
  - 開発用証明書
  - アプリケーション ID
  - プロビジョニングプロファイル

  プロビジョニングプロファイルを作成するときには、アプリケーションをインストールする予定のすべてのテストデバイスの ID をリストアップする必要があります。
- 開発用証明書と秘密キーを P12 キーストアファイルに変換します。
- アプリケーションの ActionScript コードまたは MXML コードを記述します。
- ActionScript または MXML のコンパイラーを使用してアプリケーションをコンパイルします。
- アプリケーションのアイコンアートと起動画面アートを作成します。
- アプリケーション記述子を作成します（2.6 以降の名前空間を使用）。
- ADT を使用して IPA ファイルをパッケージ化します。
- iTunes を使用して、テストデバイスにプロビジョニングプロファイルを配置します。
- iOS デバイスにアプリケーションをインストールしてテストします。iTunes または USB 経由の ADT（AIR 3.4 以降の USB サポート）のいずれかを使用して、IPA ファイルをインストールできます。

AIR アプリケーションの完成後は、配布用証明書とプロビジョニングプロファイルを使用して AIR アプリケーションを再パッケージ化できます。この時点で、Apple App Store に送信する準備が整います。

モバイルアプリケーションプロパティの設定

他の AIR アプリケーションと同様に、アプリケーション記述ファイルで基本的なアプリケーションプロパティを設定します。モバイルアプリケーションでは、ウィンドウサイズや透明度など、デスクトップ固有のプロパティのいくつかが無視されます。モバイルアプリケーションでは、プラットフォーム固有のプロパティも使用できます。例えば、Android アプリケーションに android エレメントを含めたり、iOS アプリケーションに iPhone エレメントを含めたりすることができます。

一般設定

アプリケーション記述子の設定のいくつかは、すべてのモバイルデバイスアプリケーションで重要なものです。

必要な AIR ランタイムのバージョン

アプリケーション記述ファイルの名前空間を使用して、アプリケーションに必要な AIR ランタイムのバージョンを指定します。

application エレメントに割り当てられている名前空間によって、アプリケーションが使用する機能の大部分が決まります。例えば、アプリケーションが AIR 2.7 名前空間を使用しているが、ユーザーがそれよりも後のバージョンをインストールしている場合、アプリケーションは AIR 2.7 の動作を参照します（その動作が後のバージョンで変更されている場合でも）。名前空間を変更し、アップデートを発行した場合にのみ、アプリケーションで新しい動作と機能を利用できます。セキュリティの修正は、この規則には該当しない重要な例外です。

AIR 3.6 以上の Android など、アプリケーションとは別のランタイムを使用するデバイスでは、AIR の必要なバージョンがない場合、ユーザーに対して AIR のインストールまたはアップグレードが要求されます。iPhone など、ランタイムを組み込むデバイスでは、このような状況は発生しません（本来、必要なバージョンの AIR がアプリケーションと共にパッケージ化されるため）。

注意：（AIR 3.7 以降）デフォルトで、ADT は Android アプリケーションと共にランタイムをパッケージ化します。

ルートの application エレメントの xmlns 属性を使用して、名前空間を指定します。対象とするモバイルプラットフォームに応じて、次の名前空間をモバイルアプリケーション用に使用します。

```
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">
```

注意：iOS 3 デバイスは、AIR 2.0 SDK に基づいた、Packager for iPhone SDK によってサポートされます。iOS 3 用 AIR アプリケーションの構築について詳しくは、「iPhone アプリケーションの構築」を参照してください。AIR 2.6 SDK（以降）は、iPhone 3GS、iPhone 4、iPad の各デバイス上の iOS 4 以降をサポートします。

関連項目

205 ページの「application」

アプリケーション ID

いくつかの設定は、パブリッシュするアプリケーションごとに一意にする必要があります。こうした設定には、ID、名前、ファイル名などがあります。

Android アプリケーション ID

Android では、ID は、AIR ID の先頭に「air.」が付加されて、Android パッケージ名に変換されます。したがって、AIR ID が **com.example.MyApp** である場合、Android パッケージ名は **air.com.example.MyApp** になります。

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

また、ID が Android オペレーティングシステムの有効なパッケージ名ではない場合、有効な名前に変換されます。ハイフン文字はアンダースコアに変更され、ID コンポーネントの先頭に数字がある場合は、その前に大文字「A」が付加されます。例えば、ID **3-goats.1-boat** は、パッケージ名 **air.A3_goats.A1_boat** に変換されます。

注意：アプリケーション ID に付加される接頭辞を使用して、Android Market 内で AIR アプリケーションを識別することができます。接頭辞によってアプリケーションが AIR アプリケーションとして識別されないようにする場合は、「Opt-out of AIR application analytics for Android」の説明に従って、APK ファイルのパッケージ化を解除し、アプリケーション ID を変更して、再パッケージ化する必要があります。

iOS アプリケーション ID
Apple iOS Provisioning Portal で作成したアプリケーション ID と一致するように AIR アプリケーション ID を設定します。

iOS アプリケーション ID では、バンドルシード ID の後にバンドル識別子が続きます。バンドルシード ID は、Apple がアプリケーション ID に割り当てる文字列（5RM86Z4DJM など）です。バンドル識別子は、選択したドメイン名の逆 DNS 形式の名前になります。バンドル識別子の末尾がアスタリスク（*）の場合は、ワイルドカードアプリケーション ID であることを示しています。バンドル識別子の末尾がワイルドカード文字の場合は、このワイルドカードを、任意の有効な文字列に置き換えることができます。

次に、例を示します。

- Apple アプリケーション ID が 5RM86Z4DJM.com.example.helloWorld の場合、アプリケーション記述子で com.example.helloWorld を使用する必要があります。
- Apple アプリケーション ID が 96LPVWEASL.com.example.*（ワイルドカードのアプリケーション ID）の場合、com.example.helloWorld、com.example.anotherApp などの、先頭が com.example の ID を使用できます。
- Apple アプリケーション ID がバンドルシード ID とワイルドカードのみの場合（例えば、38JE93KJL.*）、AIR のすべてのアプリケーション ID を使用できます。

アプリケーション ID を指定する場合、アプリケーション ID のバンドルシード ID の部分は含めないでください。

関連項目

221 ページの「id」

216 ページの「filename」

229 ページの「name」

アプリケーションのバージョン

AIR 2.5 以降のバージョンでは、versionNumber エレメントにアプリケーションのバージョンを指定します。version エレメントは使用されなくなりました。versionNumber の値を指定するとき、一連の数値をドットで 3 つまで区切って指定する必要があります。例えば、「0.1.2」のようにします。バージョン番号の各セグメントには、3 桁までの数字を指定できます（つまり、「999.999.999」が、許可される最大のバージョン番号になります）。番号に 3 つのセグメントをすべて含める必要はありません。「1」や「1.0」なども有効なバージョン番号です。

versionLabel エレメントを使用して、バージョンのラベルを指定することもできます。バージョンラベルを追加すると、そのラベルが、Android アプリケーションの情報画面などに、バージョン番号の代わりに表示されます。バージョンラベルは、Android Market を使用して配布されるアプリケーションに対して指定する必要があります。AIR アプリケーション記述子に versionLabel の値を指定しない場合、versionNumber の値が Android バージョンラベルフィールドに割り当てられます。

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

Android では、AIR の versionNumber は Android の整数 versionCode に変換されます。この際に、a*1000000 + b*1000 + c という数式が使用されます。ここで、a、b、c は AIR のバージョン番号である a.b.c の各コンポーネントを指します。

関連項目

237 ページの「version」

237 ページの「versionLabel」

238 ページの「versionNumber」

メインアプリケーションの SWF

メインアプリケーションの SWF ファイルを、initialWindow エレメントの子である content に指定します。モバイルプロファイルでデバイスをターゲットにするとき、SWF ファイルを使用する必要があります（HTML ベースのアプリケーションはサポートされていません）。

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

ファイルは AIR パッケージに含める必要があります（ADT または IDE を使用します）。アプリケーション記述子内の名前を参照するだけでは、ファイルはパッケージに自動的には取り込まれません。

メイン画面のプロパティ

initialWindow エレメントの子エレメントのいくつかは、メインアプリケーション画面の最初の外観と動作を制御します。

- **aspectRatio** - アプリケーションで、最初に **portrait**（高さが幅よりも長い）、**landscape**（高さが幅よりも短い）、**any**（ステージの方向が自動的にすべての方向に設定される）のうちのどのフォーマットで表示するかを指定します。

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

- **autoOrients** - デバイスを回転したとき、または方向に関連したジェスチャ（スライドキーボードを開いたり閉じたりするなど）を実行したときに、ステージの方向を自動的に変更するかどうかを指定します。デフォルトの **false** を使用すると、ステージの方向はデバイスによって変更されません。

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

- **depthAndStencil** — 深度またはステンシルバッファーを使用する場合に指定します。通常、これらのバッファーは 3D コンテンツを操作する際に使用します。

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

- **fullScreen** - アプリケーションが、デバイスの画面全体を使用するか、通常のオペレーティングシステムクロム（システムステータスバーなど）と画面を共有するかを指定します。

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

- **renderMode** - ランタイムがアプリケーションをレンダリングするときに、グラフィック処理装置（GPU）を使用するか、メインの中央演算処理装置（CPU）を使用するかを指定します。一般的には、GPU レンダリングではレンダリングの速度が速くなりますが、GPU モードでは、特定の描画モードおよび PixelBender フィルターなどの一部の機能が使用できません。また、各デバイスおよびデバイスドライバーに応じて、GPU 機能と制限事項が異なります。可能な限り様々なデバイス上で、アプリケーションを必ずテストしてください（特に GPU モードを使用する場合）。

  レンダリングモードは、**gpu**、**cpu**、**direct** または **auto** に設定できます。デフォルト値は **auto** ですが、現在では CPU モードに戻ります。

  注意：Flash コンテンツの GPU アクセラレーションを、モバイルプラットフォーム向けの AIR で利用するには、renderMode="gpu" ではなく renderMode="direct"（Stage3D）を使用することをお勧めします。Adobe で公式にサポートおよび推奨している Stage3D ベースのフレームワークは、Starling（2D）および Away3D（3D）です。Stage3D および Starling/Away3D について詳しくは、http://gaming.adobe.com/getstarted/ を参照してください。

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

  注意：バックグラウンドで実行するアプリケーションには renderMode=”direct” を使用できません。

GPU モードの制限事項は、次のとおりです。

- Flex フレームワークは GPU レンダリングモードをサポートしません。
- フィルターはサポートされていません。
- PixelBender のブレンド、および塗りはサポートされていません。
- ブレンドモードの内、レイヤー、アルファ、消去、オーバーレイ、ハードライト、比較（明）および比較（暗）はサポートされません。
- ビデオを再生するアプリケーションでの GPU レンダリングモードの使用はお勧めしません。
- GPU レンダリングモードでは、仮想キーボードが開いている場合、テキストフィールドは見える位置に正しく移動しません。ユーザーがテキストを入力しているときにテキストフィールドが表示されるようにするには、ステージの softKeyboardRect プロパティおよびソフトキーボードイベントを使用してテキストフィールドを表示可能領域 まで移動します。
- 表示オブジェクトが GPU で表示されない場合、まったく表示されません。例えば、フィルターが表示オブジェクトに適用されると、オブジェクトは表示されません。

注意： AIR 2.6 以降の iOS 用 GPU 実装は、以前の AIR 2.0 バージョンで使用されていた実装とは大きく異なります。異なる最適化の考慮事項が適用されます。

関連項目

208 ページの「aspectRatio」

209 ページの「autoOrients」

212 ページの「depthAndStencil」

220 ページの「fullScreen」

231 ページの「renderMode」

サポートされるプロファイル

supportedProfiles エレメントを追加して、アプリケーションがサポートするデバイスプロファイルを指定することができます。モバイルデバイスには mobileDevice プロファイルを使用します。Adobe Debug Launcher（ADL）でアプリケーションを実行すると、リストの最初のプロファイルがアクティブプロファイルとして使用されます。また、ADL の実行時に -profile を使用して、サポートされたリスト内の特定のプロファイルを選択することもできます。すべてのプロファイルを使用してアプリケーションを実行する場合、supportedProfiles エレメントを省略できます。この場合、ADL ではデスクトッププロファイルがデフォルトのアクティブプロファイルとして使用されます。

アプリケーションがモバイルデバイスとデスクトップの両方のプロファイルをサポートすること、および通常はモバイルプロファイルでアプリケーションをテストすることを指定するには、次のエレメントを追加します。

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

関連項目

234 ページの「supportedProfiles」

241 ページの「デバイスプロファイル」

157 ページの「AIR Debug Launcher（ADL）」

必要なネイティブ拡張

mobileDevice プロファイルをサポートするアプリケーションでは、ネイティブ拡張を使用できます。

アプリケーション記述子で、AIR アプリケーションが使用するすべてのネイティブ拡張を宣言します。次の例に、2 つの必須のネイティブ拡張を指定するシンタックスを示します。

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

extensionID エレメントは、拡張記述ファイルの id エレメントと同じ値です。拡張記述ファイルは、extension.xml という名前の XML ファイルです。これは、ネイティブ拡張開発者が提供する ANE ファイル内にパッケージ化されています。

仮想キーボードの動作

パンおよびサイズ変更の自動動作を無効にするには、softKeyboardBehavior エレメントを none に設定します。これらの動作はランタイムによって使用され、仮想キーボードが開いたあとに、フォーカスの当たっているテキスト入力フィールドが表示されるようにするものです。この自動動作を無効にした場合は、アプリケーション側で、キーボードが開いた後にテキスト入力領域や関連する他のコンテンツが表示されるようにする必要があります。SoftKeyboardEvent とともに、ステージの softKeyboardRect プロパティを使用すれば、キーボードが開いたタイミングを検出し、キーボードが隠している領域を判断することができます。

この自動動作を有効にするには、次のようにこのエレメントの値を pan に設定します。

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

pan はデフォルト値であるので、softKeyboardBehavior エレメントを省略すると、この自動キーボード動作は有効になります。

注意：GPU レンダリングも使用する場合は、パン動作はサポートされません。

関連項目

233 ページの「softKeyboardBehavior」

Stage.softKeyboardRect

SoftKeyboardEvent

Android の設定

Android プラットフォームでは、アプリケーション記述子の android エレメントを使用して、Android のアプリケーションマニフェストに情報を追加できます。このマニフェストは、Android オペレーティングシステムで使用されるアプリケーションのプロパティファイルです。ADT では、APK パッケージを作成すると、Android Manifest.xml ファイルが自動的に生成されます。AIR では、いくつかのプロパティが、特定の機能を動作させるために必要な値に設定されます。AIR アプリケーション記述子の Android セクションで設定されるその他のプロパティは、Manifest.xml ファイルの対応するセクションに追加されます。

注意：ほとんどの AIR アプリケーションでは、android エレメント内のアプリケーションで要求される Android 権限を設定する必要がありますが、通常は、その他のプロパティを設定する必要はありません。

ストリング、整数またはブール値を使用する属性だけを設定できます。参照をアプリケーションパッケージ内のリソースに設定することは、サポートされていません。

注意：ランタイムには、SDK バージョン 14 以上が必要です。以降のバージョンのみに対応するアプリケーションを作成する場合は、マニフェストに正しいバージョンの <uses-sdk android:minSdkVersion=""></uses-sdk> を含める必要があります。

予約された Android マニフェストの設定

AIR は、生成された Android マニフェストドキュメントに複数のマニフェストエントリを設定して、アプリケーションとランタイム機能が正しく動作するようにします。次の設定を定義することはできません。

manifest エレメント
manifest エレメントの次の属性を設定することはできません。

- package
- android:versionCode
- android:versionName
- xmlns:android

アクティビティエレメント
メインアクティビティエレメントの次の属性を設定することはできません。

- android:label
- android:icon

application エレメント
application エレメントの次の属性を設定することはできません。

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

Android 権限

Android セキュリティモデルでは、セキュリティまたはプライバシーと関連する機能を使用するために、各アプリケーションが権限を要求する必要があります。これらの権限は、アプリケーションをパッケージ化するときに指定する必要があり、実行時には変更できません。Android オペレーティングシステムは、ユーザーがアプリケーションをインストールするときに、アプリケーションがどの権限を要求しているかをユーザーに通知します。ある特定の機能に必要な権限が要求されないと、Android オペレーティングシステムは、アプリケーションがその機能にアクセスするときに、例外をスローする場合があります。ただし、例外が必ずスローされるとは保証されません。例外は、ランタイムによってアプリケーションに送信されます。サイレント障害の場合、権限に関するエラーメッセージが Android システムログに追加されます。

AIR では、アプリケーション記述子の android エレメントに Android 権限を指定します。次のフォーマットは、権限を追加する場合に使用されます（PERMISSION_NAME は Android 権限の名前です）。

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

manifest エレメント内の uses-permissions ステートメントは、Android マニフェストドキュメントに直接追加されます。

次の権限は、様々な AIR 機能を使用するために必要になります。

ACCESS_COARSE_LOCATION アプリケーションが、Geolocation クラスを介して Wi-Fi およびセルラーネットワーク位置データにアクセスすることを許可します。

ACCESS_FINE_LOCATION アプリケーションが Geolocation クラスを介して GPS データにアクセスすることを許可します。

ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE アプリケーションが、NetworkInfo クラスを介して ネットワーク情報にアクセスすることを許可します。

CAMERA アプリケーションがカメラにアクセスすることを許可します。

注意：ユーザーがカメラ機能を使用するための権限を要求するとき、Android では、アプリケーションもカメラを要求していると見なされます。カメラがアプリケーションのオプションの機能である場合、カメラのマニフェストに uses-feature エレメントを追加し、必要な属性を false に設定します。78 ページの「Android の互換性フィルター」を参照してください。

INTERNET アプリケーションがネットワーク要求を行うことを許可します。また、リモートデバッグも許可します。

READ_PHONE_STATE AIR ランタイムが、ユーザーの電話中にオーディオをミュートすることを許可します。この権限は、アプリケーションがバックグラウンド状態の間にオーディオを再生する場合に設定します。

RECORD_AUDIO アプリケーションがマイクにアクセスすることを許可します。

WAKE_LOCK および DISABLE_KEYGUARD アプリケーションが、SystemIdleMode クラス設定を使用してデバイスのスリープ状態を防ぐことを許可します。

WRITE_EXTERNAL_STORAGE アプリケーションがデバイスの外部メモリカードへ書き込むことを許可します。

例えば、強制的にすべての権限を要求するアプリケーションに対する権限を設定するために、アプリケーション記述子に以下を追加できます。

```
<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>
```

関連項目

Android のセキュリティと権限

Android の Manifest.permission クラス

Android カスタム URI スキーム

カスタム URI スキームを使用して、Web ページまたはネイティブ Android アプリケーションから、AIR アプリケーションを起動できます。カスタム URI のサポートは、Android マニフェストに指定されているインテントフィルターに依存します。そのため、カスタム URI を他のプラットフォームで使用することはできません。

カスタム URI を使用するには、<android> ブロック内のアプリケーション記述子に intent-filter を追加します。次の例の intent-filter エレメントを両方指定する必要があります。<data android:scheme="***my-customuri***"/> ステートメントを編集して、カスタムスキームの URI ストリングを反映する必要があります。

```
<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>
```

インテントフィルターは、Android オペレーティングシステムに対して、指定されたアクションをアプリケーションが実行できることを通知します。カスタム URI の場合、これは、ユーザーが該当する URI スキームを使用したリンクをクリックしたことを意味します（ブラウザーはその処理方法を認識していません）。

アプリケーションがカスタム URI を介して呼び出されると、NativeApplication オブジェクトは invoke イベントを送出します。クエリーパラメーターを含むリンクの URL は、InvokeEvent オブジェクトの arguments 配列に配置されます。任意の数の intent-filter を使用できます。

注意：StageWebView インスタンス内のリンクでは、カスタム URI スキームを使用する URL を開くことができません。

関連項目

Android インテントフィルター

Android のアクションとカテゴリ

Android の互換性フィルター

Android オペレーティングシステムは、アプリケーションマニフェストファイル内の多くのエレメントを使用して、アプリケーションが指定のデバイスと互換性があるかどうかを判別します。この互換性に関する情報は、必要に応じてマニフェストに追加できます。これらのエレメントを含めない場合は、アプリケーションをどの Android デバイスにもインストールできます。ただし、すべての Android デバイスで適切に動作するとは限りません。例えば、カメラアプリケーションは、カメラがない電話では役に立ちません。

フィルターに使用できる Android マニフェストタグは次のとおりです。

- supports-screens
- uses-configuration
- uses-feature
- uses-sdk（AIR 3 以降）

カメラアプリケーション

アプリケーション用にカメラの権限を要求する場合、Android では、そのアプリケーションが利用可能なすべてのカメラ機能（オートフォーカスやフラッシュなど）を必要としていると見なします。アプリケーションが一部のカメラ機能を必要としている場合、またはカメラがオプションの機能である場合、カメラ用に様々な uses-feature エレメントを設定して、カメラ機能がオプションであることを示します。このように設定しない場合、ある特定の機能を備えていないデバイスまたはカメラがないデバイスを持つユーザーは、Android Market でアプリケーションを検索することができません。

次の例は、カメラの権限を要求し、すべてのカメラ機能をオプションとする方法を説明しています。

```
<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>
```

オーディオ録音アプリケーション

オーディオを録音する権限を要求した場合、Android では、アプリケーションがマイクも要求していると見なします。オーディオ録音がアプリケーションのオプション機能である場合、uses-feature タグを追加して、マイクは必要ないことを指定できます。指定しないと、マイクがないデバイスのユーザーは、Android Market でこのアプリケーションを見つけることができません。

次の例は、マイクハードウェアをオプションとして設定しながら、マイクを使用する権限を要求する方法を説明しています。

```
<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>
```

関連項目

Android Developers：Android Compatibility

Android Developers：Android feature name constants

インストールの場所

manifest エレメントの installLocation 属性を auto または preferExternal に設定することで、アプリケーションを外部のメモリカードにインストールしたり移動することを許可できます。

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

Android オペレーティングシステムでは、外部メモリへのアプリケーションのインストールは保証されません。ユーザーは、システム設定アプリケーションを使用して、アプリケーションを内部メモリと外部メモリの間で移動することもできます。

外部メモリにインストールした場合でも、アプリケーション記憶域ディレクトリの内容、共有オブジェクト、一時ファイルなどのアプリケーションキャッシュおよびユーザーデータは、内部メモリに保存されます。内部メモリを使用し過ぎないように、アプリケーション記憶域ディレクトリに保存するデータは慎重に選択してください。大量のデータは File.userDirectory または File.documentsDirectory の場所を使用して、SD カードに保存してください（これらの場所は両方とも Android の SD カードのルートにマッピングされます）。

StageWebView オブジェクトでの Flash Player プラグインおよびその他のプラグインの有効化

Android 3.0 以降では、StageWebView オブジェクトにプラグインコンテンツを表示するために、Android の application 要素でハードウェアアクセラレーションを有効にする必要があります。プラグインのレンダリングを有効にするには、application エレメントの android:hardwareAccelerated 属性を true に設定します。

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

色深度

AIR 3+

AIR 3 以降では、表示はランタイムによって 32 bit カラーでのレンダリングに設定されます。それよりも前のバージョンの AIR の場合、ランタイムでは 16 bit カラーが使用されます。アプリケーション記述子の <colorDepth> エレメントを使用すると、16 bit カラーを使用するようにランタイムに対して指示することができます。

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

16 bit の色深度を使用すると、レンダリングのパフォーマンスを高めることができますが、色の再現性は低下します。

iOS の設定

iOS デバイスのみに適用される設定は、アプリケーション記述子の <iPhone> エレメント内に配置します。iPhone エレメントは InfoAdditions エレメント、requestedDisplayResolution エレメント、Entitlements エレメント、externalSwfs エレメントおよび forceCPURenderModeForDevices エレメントを子として持つことができます。

InfoAdditions エレメントを使用すると、アプリケーションの Info.plist 設定ファイルに追加されるキーと値のペアを指定できます。例えば、次の値は、アプリケーションのステータスバーのスタイルを設定しています。また、アプリケーションに永続的な Wi-Fi アクセスが不要であることを示しています。

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

InfoAdditions の設定は、CDATA タグで囲まれています。

Entitlements エレメントを使用すると、アプリケーションの Entitlements.plist 設定ファイルに追加するキーと値のペアを指定できます。Entitlements.plist 設定により、プッシュ通知など、iOS の特定の機能にアプリケーションからアクセスできます。

Info.plist および Entitlements.plist の設定について詳しくは、Apple デベロッパードキュメントを参照してください。

iOS でのバックグラウンドタスクのサポート

AIR 3.3

Adobe AIR 3.3 以降のバージョンでは、以下の特定のバックグラウンド動作を有効にすることで、iOS でのマルチタスクがサポートされます。

- オーディオ
- 位置情報の更新
- ネットワーク
- アプリケーションのバックグラウンドでの実行の終了

注意： swf バージョン 21 以前では、レンダリングモードがダイレクトに設定されている場合、AIR は iOS と Android でのバックグラウンドでの実行をサポートしません。この制約から、Stage3D ベースのアプリでは、オーディオ再生、位置の更新、ネットワークアップロードまたはダウンロードなどのバックグラウンドタスクを実行できません。iOS ではバックグラウンドで OpenGLES またはレンダリングを呼び出すことはできません。バックグラウンドで OpenGL を呼び出そうとしたアプリケーションは、iOS によって終了されます。Android では、バックグラウンドでの OpenGLES の呼び出し、またはその他のバックグラウンドタスク（オーディオ再生など）の実行について、アプリケーションへの制約はありません。swf バージョン 22 以降では、レンダリングモードがダイレクトに設定されている場合、AIR モバイルアプリケーションはバックグラウンドで実行できます。OpenGLES の呼び出しがバックグラウンドで実行されると、AIR iOS ランタイムでは ActionScript エラー（3768 - バックグラウンドでの実行中、Stage3D API は使用できません）が発生します。ただし、Android ではネイティブアプリケーションがバックグラウンドで OpenGLES を呼び出すことが可能なので、エラーは発生しません。モバイルリソースを最適に利用するため、アプリケーションがバックグラウンドで実行されている間はレンダリングの呼び出しを行わないでください。

バックグラウンドオーディオ

バックグラウンドオーディオの再生と録音を有効にするには、InfoAdditions エレメント内に次のキーと値のペアを含めます。

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

バックグラウンドでの位置情報の更新

バックグラウンドでの位置情報の更新を有効にするには、InfoAdditions エレメント内に次のキーと値のペアを含めます。

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

注意：位置情報 API は大量のバッテリーを消費するので、この機能は必要な場合にのみ使用してください。

バックグラウンドネットワーク

短いタスクをバックグラウンドで実行するには、アプリケーションの NativeApplication.nativeApplication.executeInBackground プロパティを true に設定します。

例えば、ユーザーが別のアプリケーションを前に移動した後にアプリケーションのファイルのアップロード処理を開始できます。アプリケーションはアップロード完了イベントを受け取ると、NativeApplication.nativeApplication.executeInBackground を false に設定します。

NativeApplication.nativeApplication.executeInBackground プロパティを true に設定すると、iOS は強制的にバックグラウンドタスクに対して制限時間を設けるので、アプリケーションは無制限には実行されない場合があります。iOS がバックグラウンド処理を停止すると、AIR は NativeApplication.suspend イベントを送出します。

バックグラウンドでの実行の終了

InfoAdditions エレメント内に次のキーと値のペアを含めることで、アプリケーションはバックグラウンドでの実行を明示的に終了できます。

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

予約された iOS InfoAdditions 設定

AIR は、生成された Info.plist ファイルに複数のエントリを設定して、アプリケーションとランタイム機能が正しく動作するようにします。次の設定を定義することはできません。

CFBundleDisplayName	CTInitialWindowTitle
CFBundleExecutable	CTInitialWindowVisible
CFBundleIconFiles	CTIosSdkVersion
CFBundleIdentifier	CTMaxSWFMajorVersion
CFBundleInfoDictionaryVersion	DTPlatformName
CFBundlePackageType	DTSDKName
CFBundleResourceSpecification	MinimumOSVersion（3.2 までは予約済み）
CFBundleShortVersionString	NSMainNibFile
CFBundleSupportedPlatforms	UIInterfaceOrientation
CFBundleVersion	UIStatusBarHidden
CTAutoOrients	UISupportedInterfaceOrientations

注意：MinimumOSVersion を定義できます。MinimumOSVersion の定義は Air 3.3 以降で有効です。

様々な iOS デバイスモデルのサポート

iPad をサポートするには、InfoAdditions エレメント内に UIDeviceFamily の適切なキーと値の設定を含めます。UIDeviceFamily 設定は string の配列です。各 string ではサポートするデバイスを定義します。<string>1</string> 設定は、iPhone および iPod Touch のサポートを定義します。<string>2</string> 設定は iPad のサポートを定義します。<string>3</string> 設定は tvOS のサポートを定義します。これらの設定の一方のみを指定した場合、そのデバイス群のみがサポートされます。例えば、次の設定では iPad のサポートのみに限定されます。

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

次の設定では、両方のデバイス群（iPhone/iPod Touch と iPad）がサポートされます。

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

さらに、AIR 3.7 以降では forceCPURenderModeForDevices タグを使用して、指定したデバイスのセットに対して強制的に CPU レンダリングモードを使用し、その他の iOS デバイスに対しては GPU レンダリングモードを有効にすることができます。

このタグは iPhone タグの子として追加し、デバイスのモデル名のリストをスペース区切りで指定します。有効なデバイスのモデル名のリストについては、219 ページの「forceCPURenderModeForDevices」を参照してください。

例えば、旧バージョンの iPod、iPhone、iPad で CPU モードを使用し、その他のすべてのデバイスで GPU モードを有効にするには、アプリケーション記述子で以下を指定します。

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

高解像度ディスプレイ

requestedDisplayResolution エレメントでは、アプリケーションが、高解像度画面を備えた iOS デバイスで標準解像度モードと高解像度モードのどちらを使用するのかを指定します。

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

高解像度モードでは、高解像度ディスプレイの各ピクセルを個別に処理できます。標準解像度モードでは、デバイスの画面は、アプリケーションに対して標準解像度画面として示されます。標準モードで単一のピクセルを描画すると、高解像度画面で 4 つのピクセルの色が設定されます。

デフォルト設定は、standard（標準）です。iOS デバイスを対象とする場合は、requestedDisplayResolution エレメントを、（InfoAdditions エレメントや initialWindow エレメントではなく）iPhone エレメントの子として使用します。

デバイスごとに異なる設定を使用する場合は、requestedDisplayResolution エレメントの値としてデフォルト値を指定します。また、excludeDevices 属性を使用して、その逆の値を使用するデバイスを指定します。例えば、以下のコードでは、第 3 世代 iPad 以外の高解像度モードをサポートするすべてのデバイスで高解像度モードを使用し、第 3 世代 iPad では標準モードを使用します。

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

excludeDevices 属性は、AIR 3.6 以降で使用できます。

関連項目

232 ページの「requestedDisplayResolution」

Renaun Erickson：Developing for both retina and non-retina iOS screens using AIR 2.6

iOS カスタム URI スキーム

カスタム URI スキームを登録することにより、デバイス上にある Web ページ内のリンクやその他のネイティブアプリケーションからアプリケーションを呼び出せるようになります。URI スキームを登録するには、InfoAdditions エレメントに CFBundleURLTypes キーを追加します。次の例では、**com.example.app** という名前の URI スキームを登録し、**example://foo** という形式の URL からアプリケーションを呼び出せるようにします。

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

アプリケーションがカスタム URI を介して呼び出されると、NativeApplication オブジェクトは invoke イベントを送出します。クエリーパラメーターを含むリンクの URL は、InvokeEvent オブジェクトの arguments 配列に配置されます。任意の数のカスタム URI スキームを使用できます。

注意：StageWebView インスタンス内のリンクでは、カスタム URI スキームを使用する URL を開くことができません。

注意：他のアプリケーションによって既にスキームが登録されている場合、このアプリケーションを置き換えて、新たなアプリケーションを URI スキーム用に登録することはできません。

iOS 互換性フィルタリング

特定のハードウェアまたはソフトウェア機能を持つデバイスでのみアプリケーションを使用できるようにする場合は、InfoAdditions エレメント内の UIRequiredDeviceCapabilities 配列にエントリを追加します。例えば、次のエントリは、アプリケーションはスチールカメラとマイクを必要としていることを示しています。

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

対応する機能がデバイスにない場合は、アプリケーションはインストールできません。AIR アプリケーションに関連する機能の設定には次のものが含まれます。

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

AIR 2.6 以降では、必須機能のリストに **armv7** と **opengles-2** が自動的に追加されます。

注意：アプリケーションでこれらの機能を使用するために、アプリケーション記述子にこれらの機能を含める必要はありません。UIRequiredDeviceCapabilities 設定は、適切に機能しないデバイス上へのアプリケーションのインストールを防ぐ目的でのみ使用してください。

一時停止せずに終了

ユーザーが AIR アプリケーションから切り替えると、AIR アプリケーションはバックグラウンドモードに入り一時停止します。アプリケーションを一時停止せずに完全に終了する場合は、UIApplicationExitsOnSuspend プロパティを YES に設定します。

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

外部の、アセットのみの SWF をロードすることでダウンロードサイズを最小限にする

AIR 3.7

アプリケーションによって使用される SWF のサブセットをパッケージ化し、Loader.load() メソッドを使用してランタイムでその他の（アセットのみの）外部 SWF をロードすることで、最初のアプリケーションのダウンロードサイズを最小限に抑えることができます。この機能を使用するには、ADT がアセットのみを含む SWF ファイルを残し、外部でロードされた SWF ファイルからメインアプリケーションの SWF にすべての ActionScript バイトコード（ABC）を移動するアプリケーションをパッケージ化する必要があります。これは、アプリケーションがインストールされた後にコードをダウンロードすることを禁じる Apple Store のルールに従うためです。

ADT は、外部でロードされた SWF（除外された SWF とも呼ばれます）をサポートするために以下を行います。

- <iPhone> エレメントの <externalSwfs> サブエレメント内で指定されたテキストファイルを読み取って、実行時にロードされる SWF の行区切りリストにアクセスします。

  ```
  <iPhone>
                                              . . .

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

- 外部でロードされた各 SWF からメインの実行可能ファイルに ABC コードを転送します。
- 外部でロードされた SWF を .ipa ファイルから除外します。
- 除外された SWF を .remoteStrippedSWFs ディレクトリにコピーします。これらの SWF を Web サーバーでホストすると、アプリケーションはこれらを必要に応じてランタイムにロードします。

以下の例のように、テキストファイルで 1 行に 1 つずつ名前を指定して、ランタイムでロードする SWF ファイルを指定します。

```
assets/Level1/Level1.swf
                            assets/Level2/Level2.swf
                            assets/Level3/Level3.swf
                            assets/Level4/Level4.swf
```

指定するファイルパスは、アプリケーション記述ファイルからの相対パスです。また、これらの swf は adt コマンドでアセットとして指定する必要があります。

注意：この機能は、標準のパッケージ化にのみ適用されます。高速のパッケージ化（インタープリター、シミュレーター、デバッグを使用する場合など）では、ADT は除外された SWF を作成しません。

サンプルコードを含め、この機能について詳細は、アドビのエンジニア Abhinav Dhandh のブログ「External hosting of secondary SWFs for AIR apps on iOS」を参照してください。

Geolocation のサポート

Geolocation をサポートするために、以下のいずれかのキーと値のペアを InfoAdditions エレメントに追加します。

```
<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>
```

アプリケーションアイコン

次の表は、各モバイルプラットフォームで使用されるアイコンのサイズを示しています。

アイコンのサイズ	プラットフォーム
29x29	iOS
36x36	Android
40 x 40	iOS
48x48	Android、iOS
50 x 50	iOS
57x57	iOS
58 x 58	iOS
60 x 60	iOS
72x72	Android、iOS
75 x 75	iOS
76 x 76	iOS
80 x 80	iOS
87 x 87	iOS
96x96	Android
100 x 100	iOS

アイコンのサイズ	プラットフォーム
114x114	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

アプリケーション記述ファイルの icon エレメントで、アイコンファイルへのパスを指定します。

```
<icon>
                        <image36x36>assets/icon36.png</image36x36>
                        <image48x48>assets/icon48.png</image48x48>
                        <image72x72>assets/icon72.png</image72x72>
                        </icon>
```

特定のサイズのアイコンを用意していない場合は、その次に大きなサイズが使用され、画面に合うようにサイズが調整されます。

Android におけるアイコン

Android では、アプリケーション記述子に指定されたアイコンは、アプリケーションランチャーアイコンとして使用されます。アプリケーションランチャーアイコンは、36 x 36 ピクセル、48 x 48 ピクセル、72 x 72 ピクセル、96 x 96 ピクセル、144 x 144 ピクセルおよび 192 x 192 ピクセルの PNG 画像のセットとして用意する必要があります。これらのアイコンのサイズは、それぞれ低密度、中密度および高密度の画面に使用されます。

開発者は、Google Play ストアへのアプリ提出時に、512 x 512 ピクセルのアイコンを提出する必要があります。

iOS のアイコン

iOS アプリケーションでは、次の場所でアプリケーション記述子に定義されたアイコンが使用されます。

- 29 x 29 ピクセルのアイコン - このアイコンは、低解像度の iPhone および iPod でスポットライト検索アイコンとして、また低解像度の iPad で設定アイコンとして使用されます。
- 40 x 40 ピクセルのアイコン - このアイコンは、低解像度の iPad でスポットライト検索アイコンとして使用されます。
- 48 x 48 ピクセルのアイコン - AIR によってこの画像に境界線が追加され、低解像度の iPad でスポットライト検索用の 50 x 50 アイコンとして使用されます。
- 50 x 50 ピクセルのアイコン - このアイコンは、低解像度の iPad でスポットライト検索アイコンとして使用されます。
- 57 x 57 ピクセルのアイコン - このアイコンは、低解像度の iPhone および iPod でアプリケーションアイコンとして使用されます。
- 58 x 58 ピクセルのアイコン - このアイコンは、Retina ディスプレイの iPhone および iPod でスポットライト検索アイコンとして、また Retina ディスプレイの iPad で設定アイコンとして使用されます。
- 60 x 60 ピクセルのアイコン - このアイコンは、低解像度の iPhone および iPod でアプリケーションアイコンとして使用されます。

- 72 x 72 ピクセルのアイコン（オプション）- このアイコンは、低解像度の iPad でアプリケーションアイコンとして使用されます。
- 76 x 76 ピクセルのアイコン（オプション）- このアイコンは、低解像度の iPad でアプリケーションアイコンとして使用されます。
- 80 x 80 ピクセルのアイコン - このアイコンは、高解像度の iPhone、iPod および iPad でスポットライト検索アイコンとして使用されます。
- 100 x 100 ピクセルのアイコン - このアイコンは、Retina ディスプレイの iPad でスポットライト検索アイコンとして使用されます。
- 114 x 114 ピクセルのアイコン - このアイコンは、Retina ディスプレイの iPhone および iPod でアプリケーションアイコンとして使用されます。
- 120 x 120 ピクセルのアイコン - このアイコンは、高解像度の iPhone および iPod でアプリケーションアイコンとして使用されます。
- 152 x 152 ピクセルのアイコン - このアイコンは、高解像度の iPad でアプリケーションアイコンとして使用されます。
- 167 x 167 ピクセルのアイコン - このアイコンは、高解像度の iPad Pro でアプリケーションアイコンとして使用されます。
- 512 x 512 ピクセルのアイコン - このアイコンは、低解像度の iPhone、iPod および iPad でアプリケーションアイコンとして使用されます。iTunes では、このアイコンが表示されます。512 ピクセルの PNG ファイルが使用されるのは、開発版のアプリケーションをテストする場合のみです。Apple App Store に最終版のアプリケーションを送信する場合、512 イメージは別途 JPG ファイルとして送信します。このイメージは IPA には含まれません。
- 1024 x 1024 ピクセルのアイコン - このアイコンは、Retina ディスプレイの iPhone、iPod および iPad でアプリケーションアイコンとして使用されます。

アイコンのグレア効果は、iOS によって追加されます。ソースイメージに対して効果を適用する必要はありません。デフォルトのグレア効果を取り消すには、アプリケーション記述ファイルの InfoAdditions エレメントに以下を追加します。

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

注意：iOS では、アドビが Apple iOS の App Store で提供されている AIR アプリケーションの数を追跡できるように、アプリケーションのメタデータが png メタデータとしてアプリケーションアイコンに挿入されます。このアイコンのメタデータによってアプリケーションが AIR アプリケーションとして識別されないようにする場合は、IPA ファイルのパッケージ化を解除し、アイコンのメタデータを削除して、再パッケージ化する必要があります。この手順については、Opt-out of AIR application analytics for iOS の記事で説明されています。

関連項目

220 ページの「icon」

222 ページの「imageNxN」

Android Developers: Icon Design Guidelines

iOS Human Interface Guidelines：Custom Icon and Image Creation Guidelines

iOS 起動イメージ

アプリケーションアイコンに加えて、**Default.png** という名前の起動イメージを少なくとも 1 つ指定する必要があります。必要に応じて、アプリケーション開始時の向き、解像度（高解像度の Retina ディスプレイと 16:9 の縦横比を含む）およびデバイスのそれぞれに対応する個別の起動イメージを含めることができます。また、アプリケーションが URL 経由で呼び出されるときに使用される起動イメージを個別に含めることもできます。

起動イメージファイルはアプリケーション記述子では参照されず、ルートアプリケーションディレクトリに配置する必要があります（これらのファイルはサブディレクトリには配置しないでください）。

ファイル名スキーム

次のスキームに従って、イメージに名前を付けます。

```
basename + screen size modifier + urischeme + orientation + scale + device + .png
```

ファイル名の **basename** の部分のみが必須です。この部分は、「**Default**」（最初の文字は大文字 D）か、アプリケーション記述子の InfoAdditions エレメントの UILaunchImageFile キーを使用して指定した名前のいずれかです。

screen size modifier の部分では、画面サイズが標準のサイズではない場合に画面サイズを指定します。この修飾子は、iPhone 5 や iPod touch（第 5 世代）など、画面の縦横比が 16:9 の iPhone および iPod touch モデルにのみ適用されます。この修飾子でサポートされる値は -568h のみです。また、これらのデバイスは高解像度（Retina）ディスプレイをサポートするので、画面サイズの修飾子は常に @2x の修飾子を持つイメージと共に使用されます。これらのデバイスの完全なデフォルトの起動イメージ名は Default-568h@2x.png です。

urischeme 部分は、URI スキームの識別に使用する文字列です。この部分は、アプリケーションが 1 つ以上のカスタム URL スキームをサポートする場合にのみ適用されます。例えば、アプリケーションを example://foo などのリンク経由で呼び出すことができる場合、"example" を起動イメージファイル名のスキーム部として使用します。

orientation の部分は、アプリケーションの起動時にデバイスの向きによって使用する複数の起動イメージを指定する方法を提供します。この部分は iPad アプリケーションのイメージにのみ適用されます。この部分には、アプリケーション開始時のデバイスの向きを参照する次の値のいずれかを指定できます。

- -Portrait
- -PortraitUpsideDown
- -Landscape
- -LandscapeLeft
- -LandscapeRight

scale 部分には、高解像度（Retina）ディスプレイで使用される起動イメージ用に @2x（iPhone 4、iPhone 5 および iPhone 6 の場合）または @3x（iPhone 6 Plus の場合）を指定します（標準解像度ディスプレイで使用されるイメージでは、scale 部をすべて省略してください）。iPhone 5 や iPod touch（第 5 世代）などのより高さのあるデバイスの起動イメージには、basename 部分の後、その他の部分の前に、画面サイズ修飾子 -528h も指定する必要があります。

device 部分は、携帯機器や携帯電話の起動イメージを指定するために使用されます。この部分は、アプリケーションが 1 つのバイナリで携帯機器とタブレットの両方をサポートするユニバーサルアプリである場合に使用します。~ipad または ~iphone のいずれかの値を（iPhone および iPod Touch の両方に）使用する必要があります。

iPhone の場合に含めることができるのは、縦横比が縦長のイメージだけです。ただし、iPhone 6 Plus の場合は、横長のイメージを追加することもできます。標準解像度のデバイスには 320 x 480 ピクセルのイメージ、高解像度デバイスには 640 x 960 ピクセルのイメージ、iPhone 5 や iPod touch（第 5 世代）などの縦横比 16:9 のデバイスには 640 x 1136 ピクセルのイメージを使用します。

iPad の場合は、次のようにイメージを含めます。

- AIR 3.3 以前 - フルスクリーン以外のイメージ：縦横比が横長（標準解像度の場合 1024 x 748、高解像度の場合 2048 x 1496）および縦長（標準解像度の場合 768 x 1004、高解像度の場合 1536 x 2008）の両方のイメージを含めます。

- AIR 3.4 以降 - フルスクリーンのイメージ：縦横比が横長（標準解像度の場合 1024 x 768、高解像度の場合 2048 x 1536）および縦長（標準解像度の場合 768 x 1024、高解像度の場合 1536 x 2048）の両方のイメージを含めます。フルスクリーン以外のアプリケーション用にフルスクリーンのイメージをパッケージ化すると、上部 20 ピクセル（高解像度の場合は上部 40 ピクセル）がステータスバーで隠されます。この領域には、重要な情報を表示しないでください。

例

次の表に、可能な限り広範囲におよぶデバイスと向きをサポートし、example:// スキームを使用して URL から起動される、仮想アプリケーションに含めることができる起動イメージの例を示します。

ファイル名	イメージのサイズ	使用方法
Default.png	320 x 480	iPhone（標準解像度）
Default@2x.png	640 x 960	iPhone（高解像度）
Default-568h@2x.png	640 x 1136	iPhone、高解像度、16:9 縦横比
Default-Portrait.png	768 x 1004（AIR 3.3 以前） 768 x 1024（AIR 3.4 以降）	iPad（縦長モードの向き）
Default-Portrait@2x.png	1536 x 2008（AIR 3.3 以前） 1536 x 2048（AIR 3.4 以降）	iPad（高解像度、縦長モードの向き）
Default-PortraitUpsideDown.png	768 x 1004（AIR 3.3 以降） 768 x 1024（AIR 3.4 以降）	iPad（上下逆の縦長モードの向き）
Default-PortraitUpsideDown@2x.png	1536 x 2008（AIR 3.3 以降）1536 x 2048（AIR 3.4 以降）	iPad（高解像度、上下逆の縦長モードの向き）
Default-Landscape.png	1024 x 768	iPad（左向きの横長モードの向き）
Default-LandscapeLeft@2x.png	2048 x 1536	iPad（高解像度、左向きの横長モードの向き）
Default-LandscapeRight.png	1024 x 768	iPad（右向きの横長モードの向き）
Default-LandscapeRight@2x.png	2048 x 1536	iPad（高解像度、右向きの横長モードの向き）
Default-example.png	320 x 480	標準 iPhone で URL「example://」
Default-example@2x.png	640 x 960	高解像度 iPhone で URL「example://」
Default-example~ipad.png	768 x 1004	縦長モードの向きの iPad で URL「example://」
Default-example-Landscape.png	1024 x 768	横長モードの向きの iPad で URL「example://」

この例では、1 つの方法のみを説明しています。例えば、Default.png イメージを iPad に使用し、iPhone および iPod 用の特定の起動イメージを Default~iphone.png および Default@2x~iphone.png で指定できます。

参照項目
iOS Application Programming Guide: Application Launch Images

iOS デバイス用にパッケージ化する起動イメージ

デバイス	解像度（ピクセル）	起動イメージ名	方向

iPhone			
iPhone 4（Retina 以外）	640 x 960	Default~iphone.png	縦長
iPhone 4、4s	640 x 960	Default@2x~iphone.png	縦長
iPhone 5、5c、5s	640 x 1136	Default-568h@2x~iphone.png	縦長
iPhone6、iPhone7	750 x 1334	Default-375w-667h@2x~iphone.png	縦長
iPhone6+, iPhone7+	1242 x 2208	Default-414w-736h@3x~iphone.png	縦長
iPhone6+、iPhone7+	2208 x 1242	Default-Landscape-414w-736h@3x~iphone.png	横長
iPad			
iPad 1、2	768 x 1024	Default-Portrait~ipad.png	縦長
iPad 1、2	768 x 1024	Default-PortraitUpsideDown~ipad.png	上下逆の縦長
iPad 1、2	1024 x 768	Default-Landscape~ipad.png	左向きの横長
iPad 1、2	1024 x 768	Default-LandscapeRight~ipad.png	右向きの横長
iPad 3、Air	1536 x 2048	Default-Portrait@2x~ipad.png	縦長
iPad 3、Air	1536 x 2048	Default-PortraitUpsideDown@2x~ipad.png	上下逆の縦長
iPad 3、Air	2048 x 1536	Default-LandscapeLeft@2x~ipad.png	左向きの横長
iPad 3、Air	2048 x 1536	Default-LandscapeRight@2x~ipad.png	右向きの横長
iPad Pro	2048 x 2732	Default-Portrait@2x.png	縦長
iPad Pro	2732 x 2048	Default-Landscape@2x.png	横長

アートのガイドライン

サイズが正しければ、どのようなアートでも起動イメージとして作成できます。ただし、多くの場合には、アプリケーションの初期状態と同じイメージを使用することをお勧めします。このような起動イメージは、アプリケーションの起動画面のスクリーンショットを取得することで作成できます。

1. iOS デバイスでアプリケーションを開きます。ユーザーインターフェイスの最初の画面が表示されたら、画面の下にあるホームボタンを長押しします。ホームボタンを押したまま、電源 / スリープボタン（iPhone デバイスの上部）を押します。これでスクリーンショットが撮影され、カメラロールに送信されます。
2. iPhoto またはその他の写真転送アプリケーションから、このスクリーンショットイメージを開発コンピューターに転送します。

アプリケーションが複数の言語にローカライズされる場合は、起動イメージにテキストを含めないでください。起動イメージは静的であり、テキストが他の言語に適合しません。

参照項目

iOS Human Interface Guidelines：Launch images

無視される設定

モバイルデバイス上のアプリケーションは、ネイティブの Windows オペレーティングシステム機能またはデスクトップオペレーティングシステム機能に適用されるアプリケーション設定を無視します。次の設定が無視されます。

- allowBrowserInvocation
- customUpdateUI
- fileTypes
- height
- installFolder
- maximizable
- maxSize
- minimizable
- minSize
- programMenuFolder
- resizable
- systemChrome
- title
- transparent
- visible
- width
- x
- y

モバイル AIR アプリケーションのパッケージ化

ADT -package コマンドを使用して、モバイルデバイスでの使用を目的とした AIR アプリケーション用のアプリケーションパッケージを作成します。-target パラメーターでは、パッケージの作成対象となるモバイルプラットフォームを指定します。

Android パッケージ

Android 上の AIR アプリケーションは、AIR パッケージ形式ではなく、Android アプリケーションパッケージ形式（APK）を使用します。

ターゲットの種類として **APK** を使用して ADT が生成したパッケージは、Android Market に送信可能な形式になります。送信するアプリケーションが Android Market に受け入れられるには、アプリケーションが満たす必要のある要件があります。最終パッケージを作成する前に、最新の要件を確認してください。「Android Developers：Publishing on the Market」を参照してください。

iOS アプリケーションとは異なり、Android アプリケーションの署名には通常の AIR コードサイニング証明書を使用できますが、Android Market にアプリケーションを送信するには、証明書が Market のルール（少なくとも 2033 年までの有効期間が必要）に準拠している必要があります。ADT -certificate コマンドを使用して、このような証明書を作成できます。

Google マーケットからの AIR ダウンロードの要求をアプリケーションに許可しないその他のマーケットにアプリケーションを送信する場合には、ADT の -airDownloadURL パラメーターを使用して代替ダウンロード URL を指定できます。AIR ランタイムの必須バージョンを保持していないユーザーがアプリケーションを起動する際には、ユーザーは指定した URL に転送されます。詳しくは、163 ページの「ADT package コマンド」を参照してください。

デフォルトで、ADT は共有ランタイムで Android アプリケーションをパッケージ化します。このため、アプリケーションを実行するには、ユーザーはデバイスに別の AIR ランタイムをインストールする必要があります。

注意：ADT がキャプティブランタイムを使用する APK を強制的に作成するようにするには、target apk-captive-runtime を使用します。

iOS パッケージ

iOS 上の AIR アプリケーションは、ネイティブの AIR 形式ではなく、iOS パッケージ形式（IPA）を使用します。

ターゲットの種類として **ipa-app-store** を使用して ADT が生成したパッケージ、正しいコードサイニング証明書およびプロビジョニングプロファイルは、Apple App Store に送信可能な形式になります。アドホックな配布用のアプリケーションをパッケージ化するには、ターゲットの種類として **ipa-ad-hoc** を使用します。

Apple が発行した正しい開発用証明書を使用して、アプリケーションに署名する必要があります。テストビルドの作成に使用する証明書と、アプリケーション送信前の最終的なパッケージ化に使用する証明書は異なります。

Ant を使用して iOS アプリケーションをパッケージ化する方法の例については、「Piotr Walczyszyn: Packaging AIR application for iOS devices with ADT command and ANT script」を参照してください。

ADT によるパッケージ化

AIR SDK バージョン 2.6 以降は、iOS と Android の両方のパッケージ化をサポートします。パッケージ化の前に、すべての ActionScript、MXML および拡張コードをコンパイルする必要があります。また、コード署名証明書が必要です。

ADT コマンドとオプションについて詳しくは、162 ページの「AIR 開発ツール（ADT）」を参照してください。

Android APK パッケージ

APK パッケージの作成

APK パッケージを作成するには、ADT パッケージコマンドを使用します。ターゲットのタイプは、リリースビルドの場合は **apk**、デバッグビルドの場合は **apk-debug**、エミュレーター上で実行するためのリリースモードビルドの場合は **apk-emulator** に設定します。

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

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。また、この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

アプリケーションファイルが含まれているディレクトリからコマンドを実行する必要があります。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf および icons ディレクトリです。

上に示したコマンドを実行するとき、ADT によって、キーストアパスワードの入力を求めるプロンプトが表示されます（入力するパスワードの文字は表示されません。入力が完了したら Enter キーを押します）。

注意：デフォルトでは、すべての AIR Android アプリケーションのパッケージ名に air. という接頭辞が付けられます。このデフォルトの動作を使用しないようにするには、コンピューターで環境変数 AIR_NOANDROIDFLAIR を **true** に設定します。

ネイティブ拡張を使用するアプリケーション用の APK パッケージの作成

ネイティブ拡張を使用するアプリケーション用の APK パッケージを作成するには、通常のパッケージングオプションに加えて -extdir オプションを指定します。複数の ANE でリソースとライブラリを共有する場合、ADT では単一のリソースとライブラリのみを選択し、その他の重複エントリを無視します。その後で警告を発行します。このオプションでは、アプリケーションで使用する ANE ファイルを含むディレクトリを指定します。次に、例を示します。

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

独自バージョンの AIR ランタイムを含む APK パッケージの作成

アプリケーションと AIR ランタイムのキャプティブバージョンの両方を含む APK パッケージを作成するには、apk-captive-runtime ターゲットを使用します。このオプションでは、アプリケーションで使用する ANE ファイルを含むディレクトリを指定します。次に、例を示します。

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

この方法には次の欠点があります。

- アドビからセキュリティパッチが発行されても、重要なセキュリティの修正が自動的に入手可能にならない
- アプリケーションで使用される RAM 容量が大きい

注意：ランタイムをバンドルすると、ADT によって INTERNET 権限と BROADCAST_STICKY 権限がアプリケーションに追加されます。これらの権限は、AIR ランタイムで必要とされます。

デバッグ APK パッケージの作成

デバッガーで使用できるアプリケーションのバージョンを作成するには、apk-debug をターゲットとして使用し、接続オプションを指定します。

```
adt -package
                                -target apk-debug
                                -connect 192.168.43.45
                                -storetype pkcs12 -keystore ../codesign.p12
                                myApp.apk
                                myApp-app.xml
                                myApp.swf icons
```

-connect フラグは、デバイス上の AIR ランタイムに対して、ネットワーク経由でリモートデバッガーへ接続するための場所をを伝えます。USB 経由でデバッグするには、-listen フラグを代わりに使用し、デバッグ接続に使用する TCP ポートを指定する必要があります。

```
adt -package
                                -target apk-debug
                                -listen 7936
                                -storetype pkcs12 -keystore ../codesign.p12
                                myApp.apk
                                myApp-app.xml
                                myApp.swf icons
```

また、ほとんどのデバッグ機能を動作させるために、デバッグを有効にして、アプリケーション SWF と SWC をコンパイルする必要があります。-connect フラグおよび -listen フラグについて詳しくは、179 ページの「デバッガー接続のオプション」を参照してください。

注意：デフォルトで、ADT は Android アプリケーションで AIR ランタイムのキャプティブコピーをパッケージ化し、apk-debug ターゲットを使用してアプリケーションをパッケージ化します。ADT が外部ランタイムを使用する APK を強制的に作成するようにするには、AIR_ANDROID_SHARED_RUNTIME 環境変数を true に設定します。

Android では、ネットワーク経由でデバッガーを実行しているコンピューターに接続するアプリケーションには、インターネットへのアクセス権限も必要です。76 ページの「Android 権限」を参照してください。

Android エミュレーターで使用する APK パッケージの作成

Android エミュレーターではデバッグ APK パッケージを使用できますが、リリースモードのパッケージを使用することはできません。エミュレーターで使用するリリースモードの APK パッケージを作成するには、ADT package コマンドを使用し、ターゲットの種類を **apk-emulator** に設定します。

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

この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

AIR または AIRI ファイルからの APK パッケージの作成

既存の AIR または AIRI ファイルから、直接 APK パッケージを作成できます。

```
adt -target apk -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp.air
```

AIR ファイルは、アプリケーション記述ファイル内で AIR 2.5（またはそれ以降）の名前空間を使用する必要があります。

Android x86 プラットフォーム用の APK パッケージの作成

AIR 14 以降では、引数 -arch を使用して Android x86 プラットフォーム用の APK をパッケージ化できます。次に、例を示します。

```
    adt     -package
                                -target apk-debug
                                -listen 7936
                                -arch x86
                                -storetype pkcs12 -keystore ../codesign.p12
                                myApp.apk
                                myApp-app.xml
                                myApp.swf icons
```

iOS パッケージ

iOS では、ADT によって SWF ファイルのバイトコードと他のソースファイルが、ネイティブ iOS アプリケーションに変換されます。

1. Flash Builder、Flash Professional またはコマンドラインコンパイラーを使用して、SWF ファイルを作成します。
2. コマンドシェルまたは端末を開き、iPhone アプリケーションのプロジェクトフォルダーに移動します。
3. 次に、ADT ツールを使用して IPA ファイルを作成します。次のシンタックスを使用します。

```
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
```

参照 adt を、ADT アプリケーションへの完全なパスを含むように変更します。ADT アプリケーションは、AIR SDK の bin サブディレクトリにインストールされています。

作成する iPhone アプリケーションのタイプに対応する -target オプションを選択します。

- -target ipa-test — 開発 iPhone でアプリケーションのテスト用のバージョンを迅速にコンパイルするには、このオプションを選択します。ipa-test-interpreter を使用すると、コンパイルをさらに迅速に行うことができます。また、ipa-test-interpreter-simulator を使用すると、iOS シミュレーターで実行できます。
- -target ipa-debug — 開発 iPhone でアプリケーションのデバッグ用のバージョンをコンパイルするには、このオプションを選択します。このオプションを選択すると、デバッグセッションを使用して、iPhone アプリケーションから trace() 出力を受け取ることができます。

  次のいずれかの -connect オプション（CONNECT_OPTIONS）を含めて、デバッガを実行する開発コンピューターの IP アドレスを指定できます。

  - -connect - アプリケーションは、アプリケーションのコンパイルに使用している開発コンピューター上のデバッグセッションに Wi-Fi 接続しようとします。
  - -connect IP_ADDRESS - アプリケーションは、指定した IP アドレスのコンピューター上のデバッグセッションに Wi-Fi 接続しようとします。次に、例を示します。

    ```
    -target ipa-debug -connect 192.0.32.10
    ```

  - -connect HOST_NAME - アプリケーションは、指定したホスト名のコンピューター上のデバッグセッションに Wi-Fi 接続しようとします。次に、例を示します。

    ```
    -target ipa-debug -connect bobroberts-mac.example.com
    ```

  -connect オプションは必須ではありません。指定しない場合は、実行されるデバッグアプリケーションはホストされたデバッガーへの接続を試みません。また、-connect の代わりに -listen を指定すると、USB デバッグを有効化できます。詳しくは、105 ページの「FDB による USB 経由のリモートデバッグ」を参照してください。

  デバッグ接続が失敗した場合、アプリケーションはユーザーに、デバッグ用ホストマシンの IP アドレスを入力するように求めるダイアログを表示します。デバイスが Wi-Fi に接続されていない場合、接続が失敗する可能性があります。また、デバイスが接続されていても、デバッグ用ホストマシンのファイアウォールに阻まれる場合に、接続が失敗することがあります。

  ipa-debug-interpreter を使用すると、コンパイルをさらに迅速に行うことができます。また、ipa-debug-interpreter-simulator を使用すると、iOS シミュレーターで実行できます。

  詳しくは、99 ページの「モバイル AIR アプリケーションのデバッグ」を参照してください。
- -target ipa-ad-hoc — アドホックデプロイ用のアプリケーションを作成するには、このオプションを選択します。Apple iPhone デベロッパーセンターを参照してください。
- -target ipa-app-store — Apple App Store にデプロイする IPA ファイルの最終版を作成するには、このオプションを選択します。

PROFILE_PATH を、アプリケーション用プロビジョニングプロファイルへのパスで置き換えます。プロビジョニングプロファイルについて詳しくは、66 ページの「iOS 用の設定」を参照してください。

-platformsdk オプションを使用して、iOS シミュレーターでアプリケーションを実行するように構築する場合に iOS シミュレーター SDK を参照します。

SIGNING_OPTIONS を、iPhone 開発用証明書およびパスワードを参照するように置き換えます。次のシンタックスを使用します。

```
-storetype pkcs12 -keystore P12_FILE_PATH -storepass PASSWORD
```

P12_FILE_PATH を、P12 証明書ファイルへのパスに置き換えます。**PASSWORD** を、証明書のパスワードに置き換えます（後述の例を参照）。P12 証明書ファイルについて詳しくは、193 ページの「P12 キーストアファイルへの開発用証明書の変換」を参照してください。

注意：iOS シミュレーター用にパッケージ化する場合は自己署名証明書を使用できます。

APP_DESCRIPTOR を、アプリケーション記述ファイルを参照するように置き換えます。

SOURCE_FILES を、プロジェクトのメインの SWF ファイルを参照するように置き換え、その他に含めるアセットをその後ろに追加します。Flash Professional のアプリケーション設定ダイアログボックスまたはカスタムアプリケーション記述ファイルで定義したすべてのアイコンファイルのパスを含めます。また、起動画面アートファイル（Default.png）も追加します。

-extdir ***extension-directory*** オプションを使用して、アプリケーションで使用する ANE ファイル（ネイティブ拡張）を含むディレクトリを指定します。アプリケーションでネイティブ拡張を使用しない場合は、このオプションは含めないでください。

重要：アプリケーションディレクトリには、Resources という名前のサブディレクトリを作成しないでください。IPA パッケージ構造に準拠するために、この名前のフォルダーがランタイムによって自動的に作成されます。独自の Resources フォルダーを作成すると、致命的な競合が発生します。

デバッグ用 iOS パッケージの作成

テストデバイスにインストールする iOS パッケージを作成するには、ADT package コマンドを使用し、ターゲットの種類を **ios-debug** に設定します。このコマンドの実行前に、Apple から開発用コードサイニング証明書とプロビジョニングプロファイルを取得しておく必要があります。

```
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
```

注意：ipa-debug-interpreter を使用すると、コンパイルをさらに迅速に行うことができます。また、ipa-debug-interpreter-simulator を使用すると、iOS シミュレーターで実行できます。

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。また、この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

アプリケーションファイルが含まれているディレクトリからコマンドを実行する必要があります。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf、icons ディレクトリ、および Default.png ファイルです。

Apple が発行した正しい配布用証明書を使用してアプリケーションに署名する必要があります。他のコードサイニング証明書を使用することはできません。

Wi-Fi デバッグの場合は -connect オプションを使用します。アプリケーションは、指定した IP またはホスト名で実行されている Flash デバッガー（FDB）を使用してデバッグセッションを開始しようとします。USB デバッグの場合は -listen オプションを使用します。まずアプリケーションを起動し、次に FDB を起動して、実行中のアプリケーションのデバッグセッションを開始します。詳しくは、103 ページの「Flash デバッガーへの接続」を参照してください。

Apple App Store への送信用 iOS パッケージの作成

Apple App Store への送信用 iOS パッケージを作成するには、ADT package コマンドを使用し、ターゲットの種類を **ios-app-store** に設定します。このコマンドの実行前に、Apple から配布用コードサイニング証明書とプロビジョニングプロファイルを取得しておく必要があります。

```
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
```

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。また、この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

アプリケーションファイルが含まれているディレクトリからコマンドを実行する必要があります。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf、icons ディレクトリ、および Default.png ファイルです。

Apple が発行した正しい配布用証明書を使用してアプリケーションに署名する必要があります。他のコードサイニング証明書を使用することはできません。

重要：Apple では、App Store へアプリケーションをアップロードするために、Apple **Application Loader** プログラムを使用することを要求しています。Apple は、Mac OS X 用 **Application Loader** のみを公開しています。このため、Windows コンピューターを使用して iPhone 用 AIR アプリケーションを開発することはできますが、App Store にアプリケーションを送信するには、OS X（バージョン 10.5.3 以降）を実行するコンピューターにアクセスできる必要があります。Application Loader プログラムは Apple iOS Developer Center から入手できます。

アドホック配布用 iOS パッケージの作成

アドホック配布用 iOS パッケージを作成するには、ADT package コマンドを使用し、ターゲットの種類を **ios-ad-hoc** に設定します。このコマンドの実行前に、Apple から適切なアドホック配布用コードサイニング証明書とプロビジョニングプロファイルを取得しておく必要があります。

```
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
```

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。また、この例では、ADT ツールへのパスがコマンドラインシェルのパス定義に従っていることを前提としています（300 ページの「PATH 環境変数」を参照してください）。

アプリケーションファイルが含まれているディレクトリからコマンドを実行する必要があります。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf、icons ディレクトリ、および Default.png ファイルです。

Apple が発行した正しい配布用証明書を使用してアプリケーションに署名する必要があります。他のコードサイニング証明書を使用することはできません。

ネイティブ拡張を使用するアプリケーション用の iOS パッケージの作成

ネイティブ拡張を使用するアプリケーション用に iOS パッケージを作成するには、ADT パッケージコマンドを、-extdir オプションを指定して使用します。ADT コマンドは、ターゲットに合わせて使用してください（ipa-app-store、ipa-debug、ipa-ad-hoc、ipa-test）。次に、例を示します。

```
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
```

コマンド全体を 1 行で入力します。上記の例では、コマンドをわかりやすくするために改行を使用しました。実際には改行を使用しません。

ネイティブ拡張に関して、この例では、extensionsDir という名前のディレクトリが、コマンドを実行するディレクトリに存在すると想定しています。extensionsDir ディレクトリには、アプリケーションで使用する ANE ファイルが含まれています。

モバイル AIR アプリケーションのデバッグ

モバイル AIR アプリケーションをデバッグする方法はいくつか用意されています。アプリケーションロジックの問題を明らかにする最も単純な方法は、ADL または iOS シミュレーターを使用して、開発コンピューター上でデバッグを実行することです。また、デバイスにアプリケーションをインストールし、デスクトップコンピューター上で実行される Flash デバッガーを使用して、リモートでデバッグすることもできます。

ADL を使用したデバイスシミュレーション

ほとんどのモバイルアプリケーション機能をテストおよびデバッグするための最も簡単で早い方法は、AIR Debug Launcher（ADL）ユーティリティを使用して、アプリケーションを開発コンピューター上で実行する方法です。ADL では、アプリケーション記述子内の supportedProfiles エレメントを使用して、どのプロファイルを使用するかを判断します。複数のプロファイルがリストされている場合、ADL は、リスト上の最初のプロファイルを使用します。ADL の -profile パラメーターを使用して、supportedProfiles リスト内の別のプロファイルを選択することもできます（アプリケーション記述子に supportedProfiles エレメントを含めない場合、任意のプロファイルを -profile 引数に指定できます）。例えば、モバイルデバイスプロファイルをシミュレートするアプリケーションを起動するには、次のコマンドを使用します。

```
adl -profile mobileDevice myApp-app.xml
```

このようなデスクトップ上のモバイルプロファイルをシミュレートする場合、ターゲットモバイルデバイスにより近い環境でアプリケーションを実行します。モバイルプロファイルの一部ではない ActionScript API は使用できません。ただし、ADL では異なるモバイルデバイスの機能が区別されません。例えば、実際のターゲットデバイスがソフトキーを使用しない場合でも、シミュレートされたソフトキーを押す操作をアプリケーションに送信できます。

ADL は、メニューコマンドを介したデバイスの方向の変更とソフトキー入力のシミュレーションをサポートします。モバイルデバイスプロファイルで ADL を実行すると、ADL は、（アプリケーションウィンドウまたはデスクトップメニューバーに）メニューを表示します。このメニューを使用して、デバイスの回転を入力したり、ソフトキー入力を実行できます。

ソフトキー入力

ADL は、モバイルデバイスのソフトキーボタン（戻るボタン、メニューボタン、検索ボタン）をシミュレートします。モバイルプロファイルを使用して ADL を起動したときに表示されるメニューを使用して、これらのキーをシミュレーション用のデバイスに送信できます。

デバイスの回転
モバイルプロファイルを使用して ADL を起動したときに表示されるメニューを使用して、デバイスの回転をシミュレートすることができます。シミュレーション用のデバイスを、右または左に回転できます。

回転シミュレーションは、自動回転を有効にしたアプリケーションにのみ影響します。自動回転は、アプリケーション記述子で autoOrients エレメントを true に設定することで有効にできます。

画面のサイズ
ADL -screensize パラメーターを設定することで、様々なサイズの画面上でアプリケーションをテストできます。事前定義済みの画面の種類の 1 つ、または通常の画面と最大化画面のピクセルサイズを表す 4 つの値を含む文字列を、コード内で渡すことができます。

縦長レイアウト用のピクセルサイズを必ず指定してください。つまり、幅の値を、高さよりも小さな値に指定してください。例えば、次のコマンドは、Motorola Droid で使用される画面をシミュレートする ADL を開きます。

```
adl -screensize 480x816:480x854 myApp-app.xml
```

事前定義済みの画面の種類のリストについては、157 ページの「ADL の使用」を参照してください。

制限
デスクトッププロファイルでサポートされていない API は、ADL でシミュレートできません。シミュレートされない API の例は次のとおりです。

- Accelerometer
- cacheAsBitmapMatrix
- CameraRoll
- CameraUI
- Geolocation
- Multitouch およびデスクトップオペレーティングシステム上でサポートされていない機能の操作
- SystemIdleMode

アプリケーションがこれらのクラスを使用する場合、実際のデバイスまたはエミュレーター上で機能をテストする必要があります。

同様に、デスクトップ上の ADL で実行する際には機能するが、一部の種類のモバイルデバイスでは機能しない API があります。これには、以下が含まれます。

- Speex および AAC オーディオコーデック
- アクセシビリティおよびスクリーンリーダーのサポート
- RTMPE
- ActionScript バイトコードを含む SWF ファイルのロード
- PixelBender シェーダー

ADL は実行環境を完全に複製するわけではないので、これらの機能を使用するアプリケーションはターゲットデバイス上で必ずテストしてください。

iOS シミュレーターを使用したデバイスシミュレーション

iOS シミュレーター（Mac のみ）を使用すると、iOS アプリケーションを迅速に実行およびデバッグできます。iOS シミュレーターを使用してテストを行う場合、開発用証明書やプロビジョニングプロファイルは不要です。p12 証明書は作成する必要がありますが、自己署名証明書を使用できます。

デフォルトでは、ADT は常に iPhone シミュレーターを起動します。シミュレーターデバイスを変更するには、次の操作を実行します。

- 以下のコマンドを使用して、使用可能なシミュレーターを表示します。

```
xcrun simctl list devices
```

以下のような出力が表示されます。

```
== 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)
....
```

- 環境変数 AIR_IOS_SIMULATOR_DEVICE を次のように設定して、特定のシミュレーターを選択できます。

```
export AIR_IOS_SIMULATOR_DEVICE = 'iPad Retina'
```

環境変数を設定した後にプロセスを再起動して、選択したシミュレーターデバイスでアプリケーションを実行します。

注意： iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

iOS シミュレーターでアプリケーションを実行するには、次の手順を実行します。

1 次の例に示すように、adt -package コマンドで -target ipa-test-interpreter-simulator または -target ipa-debug-interpreter-simulator を指定します。

```
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
```

注意： シミュレーターで署名オプションを使用する必要はなくなりました。そのため、-keystore フラグに任意の値を指定できます。これは ADT でこのフラグが無視されるためです。

2 次の例に示すように、adt -installApp コマンドを使用して、iOS シミュレーターにアプリケーションをインストールします。

```
adt -installApp
                                -platform ios
                                -platformsdk
/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator5.0.sdk
                                -device ios-simulator
                                -package sample_ipa_name.ipa
```

3 次の例に示すように、adt -launchApp コマンドを使用して、iOS シミュレーターでアプリケーションを実行します。

注意： adt -launchApp コマンドは、デフォルトではアプリケーションを iPhone シミュレーター内で実行します。アプリケーションを iPad シミュレーター内で実行するには、環境変数 AIR_IOS_SIMULATOR_DEVICE = "iPad" を書き出してから、adt -launchApp コマンドを使用します。

```
adt -launchApp
                                -platform ios
                                -platformsdk
/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator5.0.sdk
                                -device ios-simulator
                                -appid sample_ipa_name
```

iOS シミュレーターでネイティブ拡張をテストするには、次の extension.xml の例に示すように、extension.xml でプラットフォーム名として iPhone-x86 を使用し、nativeLibrary エレメントに library.a（静的ライブラリ）を指定します。

```
<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>
```

注意：iOS シミュレーターでネイティブ拡張をテストする場合は、デバイス用にコンパイルされている静的ライブラリ（.a ファイル）を使用しないでください。代わりに、シミュレーター用にコンパイルされている静的ライブラリを使用します。

トレースステートメント

デスクトップ上でモバイルアプリケーションを実行すると、デバッガーまたは ADL の起動に使用したターミナルウィンドウに、トレース出力が印刷されます。デバイスまたはエミュレーター上でアプリケーションを実行すると、トレース出力を表示するためのリモートデバッグセッションを設定できます。サポートされている場合、デバイスまたはオペレーティングシステムの製造元が提供するソフトウェア開発ツールを使用して、トレース出力を表示することもできます。

いずれの場合も、ランタイムがトレースステートメントを出力するようにデバッグを有効にした状態で、アプリケーション内の SWF ファイルをコンパイルする必要があります。

Android 上のリモートトレースステートメント

Android デバイスまたはエミュレーター上で実行する場合、Android SDK にある Android Debug Bridge（ADB）を使用して、Android システムログのトレースステートメント出力を表示できます。アプリケーションの出力を表示するには、コマンドプロンプトまたは開発コンピューターのターミナルウィンドウから、次のコマンドを実行します。

```
tools/adb logcat air.MyApp:I *:S
```

ここで、**MyApp** は、アプリケーションの AIR アプリケーション ID です。引数 *:S は、他のすべてのプロセスからの出力を抑制します。トレース出力に加えてシステム情報を表示するには、ActivityManager を logcat フィルター指定に含めます。

```
tools/adb logcat air.MyApp:I ActivityManager:I *:S
```

これらのコマンド例は、ADB を、Android SDK フォルダーから実行するか、PATH 環境変数に追加した SDK フォルダーから実行することを前提としています。

注意：AIR 2.6 以降では、ADB ユーティリティは AIR SDK に含まれており、lib/android/bin フォルダーにあります。

iOS 上のリモートトレースステートメント

iOS デバイスで実行するアプリケーションによるトレースステートメントの出力を表示するには、Flash デバッガー（FDB）を使用してリモートデバッグセッションを確立する必要があります。

関連項目

Android Debug Bridge：Enable logcat Logging

300 ページの「PATH 環境変数」

Flash デバッガーへの接続

モバイルデバイス上で実行するアプリケーションをデバッグするため、開発コンピューター上で Flash デバッガーを実行し、そこにネットワーク経由で接続することができます。リモートデバッグを有効にするには、次の手順を実行します。

- Android では、アプリケーション記述子で android:permission.INTERNET 権限を指定します。
- デバッグを有効にしてアプリケーションの SWF をコンパイルします。
- Android の場合は -target apk-debug、iOS の場合は -target ipa-debug と、さらに -connect（Wi-Fi デバッグ）フラグまたは -listen（USB デバッグ）フラグを使用してアプリケーションをパッケージ化します。

Wi-Fi 経由のリモートデバッグの場合、デバイスは、IP アドレスまたは完全修飾ドメイン名を使用して、Flash デバッガーを実行するコンピューターの TCP ポート 7935 にアクセスできる必要があります。USB 経由のリモートデバッグの場合、デバイスは、TCP ポート 7936 または -listen フラグで指定されたポートにアクセスできる必要があります。

iOS の場合は、-target ipa-debug-interpreter または -target ipa-debug-interpreter-simulator を指定することもできます。

Flash Professional によるリモートデバッグ

アプリケーションをデバッグ用に準備して、アプリケーション記述子に権限を設定したら、次の手順を実行します。

1. AIR Android 設定ダイアログを開きます。
2. 「デプロイ」タブで、次の設定を行います。
   - 「デプロイタイプ」で、「デバッグ」を選択します。
   - 「パブリッシュ後」で、「接続している Android デバイスにアプリケーションをインストール」を選択します。
   - 「パブリッシュ後」で、「接続している Android デバイスのアプリケーションを起動」を選択します。
   - 必要に応じて、Android SDK へのパスを設定します。
3. 「パブリッシュ」をクリックします。

   アプリケーションがデバイスにインストールされ、起動します。
4. AIR Android 設定ダイアログを閉じます。
5. Flash Professional メニューから、デバッグ／リモートデバッグセッションを開始／ ActionScript 3 を選択します。

   出力パネルに、「Player が接続されるのを待っています」と表示されます。
6. デバイス上でアプリケーションを起動します。
7. Adobe AIR 接続ダイアログで、Flash デバッガーを実行するコンピューターの IP アドレスまたはホスト名を入力し、「OK」をクリックします。

FDB によるネットワーク接続経由のリモートデバッグ

コマンドライン Flash デバッガー（FDB）を使用して、リモートデバイス上で実行するアプリケーションをデバッグするには、まず、開発コンピューター上でデバッガーを実行してから、デバイス上でアプリケーションを開始します。以下の手順では、AMXMLC、FDB、ADT ツールを使用して、デバイス上のアプリケーションのコンパイル、パッケージ化およびデバッグを実行します。この例は、Flex と共に AIR SDK を使用し、bin ディレクトリを PATH 環境変数に含めることを前提としています（この前提は、コマンド例を単純化するためのものです）。

1. ターミナルまたはコマンドプロンプトウィンドウを開いて、アプリケーションのソースコードを含むディレクトリに移動します。
2. デバッグを有効にして、amxmlc でアプリケーションをコンパイルします。

   ```
   amxmlc -debug DebugExample.as
   ```

3. apk-debug ターゲットまたは ipa-debug ターゲットを使用してアプリケーションをパッケージ化します。

   ```
   Android
                                       adt -package -target apk-debug -connect -storetype pkcs12 -keystore
   ../../AndroidCert.p12 DebugExample.apk DebugExample-app.xml DebugExample.swf
                                       iOS
                                       adt -package -target ipa-debug -connect -storetype pkcs12 -keystore
   ../../AppleDeveloperCert.p12 -provisioning-profile test.mobileprovision DebugExample.apk DebugExample-
   app.xml DebugExample.swf
   ```

   常に同じホスト名または IP アドレスを使用する場合、その値を -connect フラグの後に指定できます。アプリケーションは、その IP アドレスまたはホスト名への接続を自動的に試行します。または、デバッグを開始するたびに、接続情報をデバイス上で指定します。
4. アプリケーションをインストールします。

   Android では、ADT -installApp コマンドを使用できます。

   ```
   adt -installApp -platform android -package DebugExample.apk
   ```

   iOS では、ADT -installApp コマンドまたは iTunes を使用してアプリケーションをインストールできます。
5. 別のターミナルまたはコマンドウィンドウで FDB を実行します。

   ```
   fdb
   ```

6. FDB ウィンドウで、run コマンドを入力します。

   ```
   Adobe fdb (Flash Player Debugger) [build 14159]
                                       Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
                                       (fdb) run
                                       Waiting for Player to connect
   ```

7. デバイス上でアプリケーションを起動します。
8. デバイスまたはエミュレーター上でアプリケーションが起動すると、Adobe AIR 接続ダイアログが開きます（アプリケーションをパッケージ化するときに -connect オプションを使用してホスト名または IP アドレスを指定した場合、自動的にその値を使用して接続が試行されます）。適切なアドレスを入力して、「OK」をタップします。

   このモードでデバッガーに接続するには、デバイスが、アドレスまたはホスト名を解決し、TCP ポート 7935 にアクセスできる必要があります。ネットワーク接続が必要です。
9. リモートランタイムがデバッガーに接続するとき、ブレークポイントを FDB の break コマンドで設定して、continue コマンドで実行を開始できます。

```
(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
```

FDB による USB 経由のリモートデバッグ

AIR 2.6（Android）AIR 3.3（iOS）

USB 接続経由でアプリケーションをデバッグするには、-connect オプションではなく -listen オプションを使用してアプリケーションをパッケージ化します。-listen オプションを指定すると、アプリケーションの起動時に、ランタイムは TCP ポート 7936 で Flash デバッガー（FDB）からの接続を監視します。-p オプションを指定して FDB を実行すると、FDB が接続を開始します。

Android 用の USB デバッグの手順

デスクトップコンピューター上で実行する Flash デバッガーが、デバイスまたはエミュレーター上で実行する AIR ランタイムに接続するには、Android Debug Bridge（ADB：Android SDK に含まれるユーティリティ）または iOS Debug Bridge（IDB：AIR SDK に含まれるユーティリティ）を使用して、デスクトップポートにデバイスポートを転送する必要があります。

1. ターミナルまたはコマンドプロンプトウィンドウを開いて、アプリケーションのソースコードを含むディレクトリに移動します。
2. デバッグを有効にして、amxmlc でアプリケーションをコンパイルします。

   ```
   amxmlc -debug DebugExample.as
   ```

3. 適切なデバッグターゲット（apk-debug など）を使用し、-listen オプションを指定して、アプリケーションをパッケージ化します。

   ```
   adt -package -target apk-debug -listen -storetype pkcs12 -keystore ../../AndroidCert.p12 DebugExample.apk
   DebugExample-app.xml DebugExample.swf
   ```

4. USB ケーブルを使用してデバイスとデバッグコンピューターを接続します（エミュレーターで実行するアプリケーションをデバッグする場合も、この手順を使用できます。この場合、USB 接続はできないため、USB ケーブルによる接続は必要ありません）。
5. アプリケーションをインストールします。

   ADT -installApp コマンドを使用できます。

   ```
   adt -installApp -platform android -package DebugExample.apk
   ```

6. Android ADB ユーティリティを使用して、デバイスまたはエミュレーターからデスクトップコンピューターに TCP ポート 7936 を転送します。

   ```
   adb forward tcp:7936 tcp:7936
   ```

7. デバイス上でアプリケーションを起動します。
8. ターミナルまたはコマンドウィンドウで、-p オプションを使用して FDB を実行します。

   ```
   fdb -p 7936
   ```

9. FDB ウィンドウで、run コマンドを入力します。

```
Adobe fdb (Flash Player Debugger) [build 14159]
                                    Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
                                    (fdb) run
```

10 FDB ユーティリティがアプリケーションへの接続を試みます。

11 リモート接続が確立されると、FDB の break コマンドでブレークポイントを設定して、continue コマンドで実行を開始できます。

```
(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
```

注意：ポート番号 7936 は、AIR ランタイムと FDB の両方で、USB デバッグのデフォルトとして使用されます。ADT -listen ポートパラメーターと FDB -p ポートパラメーターで使用するポートには、異なるポートを指定できます。この場合、Android Debug Bridge ユーティリティを使用して、FDB で指定したポートに、ADT で指定したポート番号を転送する必要があります。このコマンドは、adb forward tcp:adt_listen_port# tcp:fdb_port# のようになります。

iOS 用の USB デバッグの手順

デスクトップコンピューター上で実行する Flash デバッガーが、デバイスまたはエミュレーター上で実行する AIR ランタイムに接続するには、iOS Debug Bridge（IDB：AIR SDK に含まれるユーティリティ）を使用して、デスクトップポートにデバイスポートを転送する必要があります。

1 ターミナルまたはコマンドプロンプトウィンドウを開いて、アプリケーションのソースコードを含むディレクトリに移動します。

2 デバッグを有効にして、amxmlc でアプリケーションをコンパイルします。

```
amxmlc -debug DebugExample.as
```

3 適切なデバッグターゲット（ipa-debug や ipa-debug-interpreter など）を使用し、-listen オプションを指定して、アプリケーションをパッケージ化します。

```
adt -package -target ipa-debug-interpreter -listen 16000
                                    xyz.mobileprovision -storetype pkcs12 -keystore Certificates.p12
                                    -storepass pass123 OutputFile.ipa InputFile-app.xml InputFile.swf
```

4 USB ケーブルを使用してデバイスとデバッグコンピューターを接続します（エミュレーターで実行するアプリケーションをデバッグする場合も、この手順を使用できます。この場合、USB 接続はできないため、USB ケーブルによる接続は必要ありません）。

5 iOS デバイスにアプリケーションをインストールして起動します。AIR 3.4 以降では、adt -installApp を使用して、USB 経由でアプリケーションをインストールできます。

6 idb -devices コマンドを使用してデバイスハンドルを確認します（IDB は ***air_sdk_root***/lib/aot/bin/iOSBin/idb にあります）。

```
./idb -devices
                                    List of attached devices
                                    Handle    UUID
                                        1     91770d8381d12644df91fbcee1c5bbdacb735500
```

注意：（AIR 3.4 以降）idb -devices の代わりに adt -devices を使用して、デバイスハンドルを確認できます。

7 IDB ユーティリティおよび前の手順で確認したデバイス ID を使用して、adt -listen パラメーターで指定したポート（この場合は 16000。デフォルトは 7936）にデスクトップのポートを転送します。

```
idb -forward 7936 16000 1
```

この例では、7936 はデスクトップポートです。16000 は接続されたデバイスが監視するポートです。1 は接続されたデバイスのデバイス ID です。

8 ターミナルまたはコマンドウィンドウで、-p オプションを使用して FDB を実行します。

```
fdb -p 7936
```

9 FDB ウィンドウで、run コマンドを入力します。

```
Adobe fdb (Flash Player Debugger) [build 23201]
                                    Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
(fdb) run
```

10 FDB ユーティリティがアプリケーションへの接続を試みます。

11 リモート接続が確立されると、FDB の break コマンドでブレークポイントを設定して、continue コマンドで実行を開始できます。

注意：ポート番号 7936 は、AIR ランタイムと FDB の両方で、USB デバッグのデフォルトとして使用されます。IDB -listen ポートパラメーターと FDB -p ポートパラメーターで使用するポートには、異なるポートを指定できます。

モバイルデバイスへの AIR と AIR アプリケーションのインストール

アプリケーションのエンドユーザーは、デバイス向けの通常のアプリケーションメカニズムと配布メカニズムを使用して、AIR ランタイムと AIR アプリケーションをインストールできます。

例えば、Android の場合、ユーザーは Android Market からアプリケーションをインストールできます。または、アプリケーション設定で、不明なソースからアプリケーションをインストールすることが許可されている場合、Web ページのリンクをクリックしてアプリケーションをインストールしたり、アプリケーションパッケージをデバイスにコピーして開いたりできます。Android アプリケーションをインストールしようとしたが、AIR ランタイムがまだインストールされていない場合、ユーザーは、ランタイムをインストールできる Market へ自動的に移動されます。

iOS では、エンドユーザーへのアプリケーションの配布方法は 2 種類あります。主要な配布チャンネルは Apple App Store です。また、アドホック配布によって、App Store 経由でなくとも、限られた数のユーザーがアプリケーションをインストールできます。

開発を目的とした AIR ランタイムとアプリケーションのインストール

モバイルデバイス上の AIR アプリケーションはネイティブパッケージとしてインストールされるので、テスト用アプリケーションのインストールに通常のプラットフォーム機能を使用できます。サポートされている場合は、ADT コマンドを使用して、AIR ランタイムと AIR アプリケーションをインストールできます。現在、この方法は Android でサポートされています。

iOS では、iTunes を使用してテスト用アプリケーションをインストールできます。テストアプリケーションは、アプリケーション開発専用に発行された Apple コードサイニング証明書によって署名し、開発用プロビジョニングプロファイルを使用してパッケージ化する必要があります。AIR アプリケーションは、iOS 上では自己完結型パッケージです。単独のランタイムは使用されません。

ADT を使用した AIR アプリケーションのインストール

AIR アプリケーションを開発するときは、ADT を使用して、ランタイムとアプリケーションの両方のインストールとアンインストールを実行できます（IDE では、ADT を実行しなくても済むように、これらのコマンドを統合できる場合があります）。

AIR ADT ユーティリティを使用して、デバイスまたはエミュレーターに AIR ランタイムをインストールできます。デバイス用に提供される SDK をインストールする必要があります。次のように、-installRuntime コマンドを使用します。

```
adt -installRuntime -platform android -device deviceID -package path-to-runtime
```

-package パラメーターを指定しない場合、インストールされている AIR SDK で使用できるランタイムパッケージから、デバイスまたはエミュレーターに適切なランタイムパッケージが選択されます。

Android または iOS（AIR 3.4 以降）上に AIR アプリケーションをインストールするには、類似する -installApp コマンドを使用します。

```
adt -installApp -platform android -device deviceID -package path-to-app
```

-platform 引数に設定される値は、インストール先のデバイスと一致する必要があります。

注意： AIR ランタイムまたは AIR アプリケーションを再インストールする前に、既存のバージョンを削除する必要があります。

iTunes を使用した iOS デバイスへの AIR アプリケーションのインストール

テストのために、iOS デバイスに AIR アプリケーションをインストールするには：

1. iTunes アプリケーションを開きます。
2. このアプリケーションのプロビジョニングプロファイルを iTunes にまだ追加していない場合は、追加します。iTunes で、ファイル／ライブラリに追加を選択します。次に、ファイルタイプ mobileprovision のプロビジョニングプロファイルを選択します。
3. 一部のバージョンの iTunes では、同じバージョンのアプリケーションが既にインストールされている場合、アプリケーションは置き換えられません。この場合は、デバイスおよび iTunes のアプリケーションリストから、該当するアプリケーションを削除します。
4. アプリケーションの IPA ファイルをダブルクリックします。このファイルが iTunes のアプリケーションのリストに表示されます。
5. コンピューターの USB ポートにデバイスを接続します。
6. iTunes で、このデバイスの「アプリケーション」タブを確認し、インストールするアプリケーションのリストで該当するアプリケーションが選択されていることを確認します。
7. iTunes アプリケーションの左側のリストでデバイスを選択します。「同期」ボタンをクリックします。同期が完了すると、Hello World アプリケーションが iPhone に表示されます。

新しいバージョンがインストールされない場合は、デバイスおよび iTunes のアプリケーションリストからアプリケーションを削除し、この手順を再度実行します。現在インストールされているアプリケーションで、インストールしようとしているアプリケーションと同じアプリケーション ID とバージョンが使用されている場合に、この問題が発生することがあります。

関連項目

173 ページの「ADT installRuntime コマンド」

171 ページの「ADT installApp コマンド」

デバイス上の AIR アプリケーションの実行

デバイスユーザーインターフェイスを使用して、インストールした AIR アプリケーションを起動できます。サポートされている場合は、AIR ADT ユーティリティを使用して、アプリケーションをリモートで起動することもできます。

```
adt -launchApp -platform android -device deviceID -appid applicationID
```

-appid 引数の値は、起動する AIR アプリケーションのアプリケーション ID にする必要があります。AIR アプリケーション記述子に指定した値を使用します（パッケージ化により追加された **air.** 接頭辞は除きます）。

接続され動作しているデバイスまたはエミュレーターが 1 つだけの場合、-device フラグを省略できます。-platform 引数に設定される値は、インストール先のデバイスと一致する必要があります。現在サポートされている値は **android** だけです。

AIR ランタイムとアプリケーションの削除

デバイスのオペレーティングシステムに用意された通常の方法を使用して、アプリケーションを削除できます。サポートされている場合は、AIR ADT ユーティリティを使用して、AIR ランタイムと AIR アプリケーションを削除できます。ランタイムを削除するには、-uninstallRuntime コマンドを使用します。

```
adt -uninstallRuntime -platform android -device deviceID
```

アプリケーションをアンインストールするには、-uninstallApp コマンドを使用します。

```
adt -uninstallApp -platform android -device deviceID -appid applicationID
```

接続され動作しているデバイスまたはエミュレーターが 1 つだけの場合、-device フラグを省略できます。-platform 引数に設定される値は、インストール先のデバイスと一致する必要があります。現在サポートされている値は **android** だけです。

エミュレーターの設定

デバイスエミュレーター上の AIR アプリケーションを実行するには、通常、デバイス用の SDK を使用して、開発コンピューター上でエミュレーターインスタンスを作成し実行する必要があります。その後で、エミュレーターに AIR ランタイムと AIR アプリケーションのエミュレーターバージョンをインストールできます。通常、エミュレーター上のアプリケーションは、実際のデバイス上よりも実行速度が遅くなります。

Android エミュレーターの作成

1. Android SDK and AVD Manager アプリケーションを起動します。
   - Windows の場合、Android SDK ディレクトリのルートで SDK Setup.exe ファイルを実行します。
   - Mac OS の場合、Android SDK ディレクトリの tools サブディレクトリで Android アプリケーションを実行します。
2. 「Settings」オプションを選択し、「Force https://」オプションを選択します。
3. 「Available Packages」オプションを選択します。使用可能な Android SDK の一覧が表示されます。
4. 互換性のある Android SDK（Android 2.3 以降）を選択し、「Install Selected」ボタンをクリックします。
5. 「Virtual Devices」オプションを選択し、「New」ボタンをクリックします。
6. 次の設定を行います。
   - 仮想デバイスの名前
   - ターゲット API（Android 2.3、API レベル 8 など）
   - SD カードのサイズ（1024 など）
   - スキン（デフォルト HVGA など）
7. 「Create AVD」ボタンをクリックします。

仮想デバイスの作成は、システム構成によっては時間がかかる場合があります。

これで、新しい仮想デバイスを起動できます。

1. AVD Manager アプリケーションで、「Virtual Device」を選択します。上で作成した仮想デバイスが表示されます。
2. 仮想デバイスを選択して、「Start」ボタンをクリックします。

3 次の画面で「Launch」ボタンをクリックします。

デスクトップにエミュレーターウィンドウが開くのを確認します。これには数秒かかります。また、Android オペレーティングシステムを初期化するのにある程度の時間がかかります。**apk-debug** および **apk-emulator** を使用して、アプリケーションパッケージをエミュレーターにインストールできます。**apk** ターゲットでパッケージ化したアプリケーションはエミュレーター上で機能しません。

関連項目

http://developer.android.com/guide/developing/tools/othertools.html#android

http://developer.android.com/guide/developing/tools/emulator.html

モバイル AIR アプリケーションのアップデート

モバイル AIR アプリケーションはネイティブパッケージとして配布されるので、プラットフォーム上の他のアプリケーションの標準的なアップデートメカニズムを使用してください。通常、このメカニズムでは、オリジナルのアプリケーションの配布に使用したのと同じマーケットプレースや App Store に送信します。

モバイル AIR アプリケーションは、AIR Updater クラスやフレームワークを使用することができません。

Android での AIR アプリケーションのアップデート

Android Market で配布されているアプリケーションの場合、次の条件がすべて満たされていれば、新しいバージョンを Market に配置することによってアプリケーションを更新できます（これらのポリシーは Market によって施行されているものであり、AIR が施行しているものではありません）。

- APK パッケージが同じ証明書で署名されている。
- AIR ID が同じである。
- アプリケーション記述子の versionNumber 値がより大きい値になっている（versionLabel を使用する場合は、この値も増やす必要があります）。

Android Market からアプリケーションをダウンロードしたユーザーには、アップデートが利用可能になったことが、使用するデバイスソフトウェアによって通知されます。

関連項目

Android Developers：Publishing Updates on Android Market

iOS での AIR アプリケーションのアップデート

iTunes App Store 経由で配布された AIR アプリケーションの場合、次の条件をすべて満たす限りは、App Store にアップデートを送信することでアプリケーションを更新できます（これらのポリシーは AIR ではなく Apple App Store によって適用されるものです）。

- コードサイニング証明書とプロビジョニングプロファイルが同じ Apple ID に発行されていること
- IPA パッケージが同じ Apple Bundle ID を使用していること
- アップデートによって、サポートされるデバイスの範囲が縮小しないこと（つまり、オリジナルのアプリケーションが iOS 3 を実行するデバイスをサポートする場合、iOS 3 のサポートを除外するアップデートを作成することはできません）

重要：AIR SDK バージョン 2.6 以降は iOS 3 をサポートせず、AIR 2 は iOS 3 をサポートするので、AIR 2.6 以降で開発されたアップデートを使用して、AIR 2 で開発されたパブリッシュ済みの iOS アプリケーションを更新することはできません。

プッシュ通知の使用

プッシュ通知を使用すると、リモート通知プロバイダーから、モバイルデバイス上で実行されているアプリケーションへ、通知を送信できます。AIR 3.4 は、Apple Push Notification サービス（APN）を使用して、iOS デバイス用にプッシュ通知をサポートしています。

注意：AIR for Android アプリケーション向けのプッシュ通知を有効にするには、Adobe エバンジェリストの Piotr Walczyszyn 氏が開発した as3c2dm などのネイティブエクステンションを使用してください。

この節の残りの部分では、AIR for iOS アプリケーションでプッシュ通知を有効にする方法について説明します。

注意：この説明は、Apple のデベロッパー ID を持っていて、iOS の開発ワークフローに精通していて、iOS デバイスで 1 つ以上のアプリケーションを開発済みであることを前提としています。

プッシュ通知の概要

Apple Push Notification サービス（APN）を使用すると、リモート通知プロバイダーから、iOS デバイス上で実行されているアプリケーションへ、通知を送信できます。APN では次の通知タイプがサポートされています。

- 警告
- バッジ
- サウンド

APN について詳しくは、developer.apple.com を参照してください。

アプリケーションでプッシュ通知を使用するには、次のような複数の要素が必要です。

- **クライアントアプリケーション** - プッシュ通知の登録、リモート通知プロバイダーとの通信、プッシュ通知の受信を行います。
- **iOS** - クライアントアプリケーションと APN の間の相互動作を管理します。
- **APN** - クライアントの登録時に tokenID を提供し、リモート通知プロバイダーから iOS へ、通知を渡します。
- **リモート通知プロバイダー** - tokenId クライアントアプリケーション情報を保存し、通知を APN にプッシュします。

登録のワークフロー

プッシュ通知をサーバー側のサービスに登録するワークフローは次のとおりです。

1. クライアントアプリケーションが、プッシュ通知を有効にするように iOS に要求します。
2. iOS が、この要求を APN に転送します。
3. APN サーバーが tokenId を iOS に返します。
4. iOS が tokenId をクライアントアプリケーションに返します。
5. クライアントアプリケーション（アプリケーション固有のメカニズムを使用）が、tokenId をリモート通知プロバイダーに提供し、そこでプッシュ通知に使用する tokenId が保存されます。

通知のワークフロー

通知のワークフローは次のとおりです。

1. リモート通知プロバイダーが通知を生成し、通知ペイロードを、tokenId と共に APN に渡します。
2. APN がデバイス上の iOS に通知を転送します。
3. iOS が通知ペイロードをアプリケーションにプッシュします。

プッシュ通知 API

AIR 3.4 では、iOS のプッシュ通知をサポートする一連の API が導入されました。これらの API は flash.notifications パッケージ内にあり、次のクラスが含まれています。

- NotificationStyle - 通知タイプの定数、ALERT、BADGE および SOUND を定義します。
- RemoteNotifier - プッシュ通知をサブスクライブおよびアンサブスクライブすることができます。
- RemoteNotifierSubscribeOptions - 受信する通知タイプを選択できます。複数の通知タイプに登録する文字列のベクトルを定義するには、notificationStyles プロパティを使用します。

AIR 3.4 には、flash.events.RemoteNotificationEvent も含まれています。これは、RemoteNotifier によって、次の場合に送出されます。

- アプリケーションのサブスクリプションが正常に作成されて、APN から新しい tokenId を受信したとき。
- 新しいリモート通知を受信したとき。

また、サブスクリプションプロセスでエラーが発生した場合、RemoteNotifier から flash.events.StatusEvent が送出されます。

アプリケーションでのプッシュ通知の管理

アプリケーションをプッシュ通知に登録するには、次の手順を実行する必要があります。

- アプリケーションでコードを作成し、プッシュ通知をサブスクライブします。
- アプリケーションの XML ファイルでプッシュ通知を有効にします。
- iOS のプッシュサービスを有効にするプロビジョニングプロファイルと証明書を作成します。

次の注釈付きのサンプルコードは、プッシュ通知をサブスクライブし、プッシュ通知イベントを処理するためのものです。

```
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 notfications 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());
}
}
}
```

アプリケーション XML ファイルでのプッシュ通知の有効化

アプリケーションでプッシュ通知を使用するには、Entitlements タグ（iphone タグの下）で、次のように指定します。

```
<iphone>
    ...
        <Entitlements>
            <![CDATA[
                <key>aps-environment</key>
                <string>development</string>
            ]]>
        </Entitlements>
    </iphone>
```

アプリケーションを App Store にプッシュする準備ができたら、<string> エレメントを development から production に変更します。

```
<string>production</string>
```

ローカライズされた文字列に対応するアプリケーションの場合は、intialWindow タグの下の supportedLanguages タグで、次の例に従って言語を指定します。

```
<supportedLanguages>en de cs es fr it ja ko nl pl pt</supportedLanguages>
```

iOS のプッシュサービスを有効にするプロビジョニングプロファイルと証明書の作成

アプリケーションと APN の通信を有効にするには、次のように、iOS のプッシュサービスを有効にするプロビジョニングプロファイルと証明書を含めてアプリケーションをパッケージ化する必要があります。

1. Apple デベロッパーアカウントにログインします。
2. Provisioning Portal に移動します。
3. 「App IDs」タブをクリックします。
4. 「New App ID」ボタンをクリックします。
5. 説明とバンドル ID を指定します（バンドル ID には * は使用しないでください）。
6. 「Submit」をクリックします。Provisioning Portal で App ID が生成され、App IDs ページが再度表示されます。
7. 「Configure」（App ID の右側）をクリックします。Configure App ID ページが表示されます。

8 「Enable for Apple Push Notification service」チェックボックスを選択します。プッシュ SSL 証明書には、開発 / テスト用と本番用の 2 種類があります。

9 「Development Push SSL Certificate」の右側にある「Configure」ボタンをクリックします。Generate Certificate Signing Request (CSR) ページが表示されます。

10 ページの指示に従って、Keychain Access ユーティリティを使用して CSR を生成します。

11 SSL 証明書を生成します。

12 SSL 証明書をダウンロードしてインストールします。

13 （オプション）本番用プッシュ SSL 証明書に関して、手順 9 から 12 を繰り返します。

14 「Done」をクリックします。Configure App ID ページが表示されます。

15 「Done」をクリックします。App IDs ページが表示されます。App ID のプッシュ通知の横に緑色の丸が付いていることを確認してください。

16 SSL 証明書は、後でアプリケーションとプロバイダーの通信に使用するので、必ず保存してください。

17 「Provisioning」タブをクリックして Provisioning Profiles ページを表示します。

18 新しい App ID のプロビジョニングプロファイルを作成してダウンロードします。

19 「Certificates」タブをクリックして、新しいプロビジョニングプロファイル用の新しい証明書をダウンロードします。

プッシュ通知にサウンドを使用

アプリケーションでサウンド通知を有効にするには、サウンドファイルを他のアセットと同様に、ただし SWF ファイルや app-xml ファイルと同じディレクトリでバンドルします。次に、例を示します。

```
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 では、次のサウンドデータ形式をサポートしています（aiff、wav または caf ファイル）。

- リニア PCM
- MA4（IMA/ADPCM）
- uLaw
- aLaw

ローカライズされた警告通知の使用

ローカライズされた警告通知をアプリケーションで使用するには、ローカライズされた文字列を、lproj フォルダーの形式でバンドルします。例えば、スペイン語の警告をサポートするには、次の手順に従います。

1 プロジェクト内で、es.lproj フォルダーを app-xml ファイルと同じレベルで作成します。

2 es.lproj フォルダー内で、Localizable.Strings というテキストファイルを作成します。

3 Localizable.Strings をテキストエディターで開き、メッセージキーおよび対応するローカライズ文字列を追加します。次に、例を示します。

```
"PokeMessageFormat" = "La notificación de alertas en español."
```

4 ファイルを保存します。

5 アプリケーションがこのキー値の警告通知を受信したとき、デバイス言語がスペイン語であれば、翻訳された警告テキストが表示されます。

リモート通知プロバイダーの設定

プッシュ通知をアプリケーションに送信するには、リモート通知プロバイダーが必要です。このサーバーアプリケーションは、プロバイダーの役割を果たし、プッシュ入力を受け入れて、通知および通知データを APN に渡します。その後、APN がプッシュ通知をクライアントアプリケーションに送信します。

リモート通知プロバイダーからの通知のプッシュについて詳しくは、Apple Developer Library の「Provider Communication with Apple Push Notification Service」を参照してください。

リモート通知プロバイダーのオプション

リモート通知プロバイダーには、次のようなオプションがあります。

- APNS-php オープンソースサーバーに基づいて独自のプロバイダーを作成。http://code.google.com/p/apns-php/ を使用して PHP サーバーを設定できます。この Google Code プロジェクトでは、特定の要件を満たすインターフェイスをデザインできます。
- サービスプロバイダーを使用。例えば、http://urbanairship.com/ では、既製の APN プロバイダーが提供されています。このサービスに登録後、まず、次のようなコードを使用してデバイストークンを指定します。

```
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);
                                    }
```

その後、Urban Airship のツールを使用して、テスト通知を送信できます。

リモート通知プロバイダーの証明書

SSL 証明書と秘密キー（生成済み）を、リモート通知プロバイダーのサーバー上の適切な場所にコピーする必要があります。通常は、これら 2 つのファイルを組み合わせて 1 つの .pem ファイルにします。これを行うには、次の手順を実行します。

1. ターミナルウィンドウを開きます。
2. 次のコマンドを入力して、SSL 証明書から .pem ファイルを作成します。

```
openssl x509 -in aps_developer_identity.cer -inform der -out TestPushDev.pem
```

3. 次のコマンドを入力して、秘密キー（.p12）ファイルの .pem ファイルを作成します。

```
openssl pkcs12 -nocerts -out TestPushPrivateKey.pem -in certificates.p12
```

4 次のコマンドを入力して、2 つの .pem ファイルを組み合わせて 1 つのファイルにします。

```
cat TestPushDev.pem TestPushPrivateKey.pem > FinalTestPush.pem
```

5 サーバー側のプッシュアプリケーションを作成するときに、組み合わせた .pem ファイルをサーバープロバイダーに提供します。

詳しくは、Apple の『Local and Push Notification Programming Guide』の「Installing the SSL Certificate and Key on the Server」を参照してください。

アプリケーションでのプッシュ通知の処理

アプリケーションでのプッシュ通知の処理には、次の項目が含まれます。

- グローバルユーザー設定とプッシュ通知の受け入れ
- 個々のプッシュ通知のユーザーによる受け入れ
- プッシュ通知および通知ペイロードデータの処理

プッシュ通知の設定と受け入れ

プッシュ通知が有効なアプリケーションを初めて起動すると、iOS に、**"appname" はあなたにプッシュ通知を送信します。よろしいですか？**ダイアログが表示されます。このダイアログには「許可しない」ボタンと「OK」ボタンがあります。「OK」を選択すると、アプリケーションは、サブスクライブしているすべてのスタイルの通知を受信します。「許可しない」を選択すると、通知はまったく受信されません。

注意：設定／通知に移動して、プッシュが有効なアプリケーションごとに、受信できる通知タイプを制御することもできます。

Apple では、アプリケーションが起動するたびにプッシュ通知をサブスクライブすることを推奨しています。アプリケーションが RemoteNotifier.subscribe() を呼び出すと、token タイプの RemoteNotificationEvent を受信します。これには、そのデバイス上のアプリケーションを一意に識別する、32-byte の一意な数値の tokenId が含まれています。

プッシュ通知を受信したデバイスには、「閉じる」ボタンと「起動」ボタンを含むポップアップが表示されます。「閉じる」にタッチすると、何も起こりません。「起動」にタッチすると、iOS でアプリケーションが呼び出され、アプリケーションは、以下で説明するように、notification タイプの flash.events.RemoteNotificationEventnotification を受信します。

通知とペイロードデータの処理

リモート通知プロバイダーがデバイスに通知を送信すると（tokenID を使用）、アプリケーションは、実行中かどうかにかかわらず、notification タイプの flash.events.RemoteNotificationEvent を受信します。この時点で、アプリケーションでは、アプリケーションに固有の通知処理が実行されます。アプリケーションが通知データを処理する場合は、JSON 形式の RemoteNotificationEvent.data プロパティを使用してアクセスします。

第 8 章：テレビデバイス用 AIR アプリケーションの開発

テレビ用の AIR 機能

テレビ、デジタルビデオレコーダー、Blu-ray プレーヤーなどのテレビデバイスが Adobe AIR for TV を搭載している場合は、それらのデバイス向けに Adobe® AIR® アプリケーションを作成できます。AIR for TV はテレビデバイス向けに最適化されており、例えば、デバイスのハードウェアアクセラレーターを活用して高パフォーマンスのビデオやグラフィックを実現します。

テレビデバイス用 AIR アプリケーションは、HTML ベースではなく、SWF ベースのアプリケーションです。AIR for TV アプリケーションでは、ハードウェアアクセラレーションやその他の「リビングルーム」環境に適した AIR 機能を活用できます。

デバイスプロファイル

AIR ではプロファイルを使用して、同様の機能を持つデバイスのターゲットセットを定義します。AIR for TV アプリケーションでは、次のプロファイルを使用します。

- tv プロファイル。AIR for TV デバイスをターゲットとする AIR アプリケーションではこのプロファイルを使用します。
- extendedTV プロファイル。AIR for TV アプリケーションでネイティブ拡張を使用する場合は、このプロファイルを使用します。

これらのプロファイル用に定義されている ActionScript の機能については、241 ページの「デバイスプロファイル」のトピックで説明しています。AIR for TV アプリケーション固有の ActionScript の相違点については、「Adobe Flash Platform 用 ActionScript 3.0 リファレンスガイド」に記載されています。

AIR for TV のプロファイルについて詳しくは、139 ページの「サポートされるプロファイル」を参照してください。

ハードウェアアクセラレーション

テレビデバイスは、AIR アプリケーションのグラフィックとビデオのパフォーマンスを飛躍的に向上するハードウェアアクセラレーターを備えています。ハードウェアアクセラレーターを活用する方法については、121 ページの「AIR for TV アプリケーションのデザイン上の考慮事項」を参照してください。

コンテンツ保護

AIR for TV により、ハリウッド超大作から自主制作映画やテレビエピソードまで、高品質ビデオコンテンツに対するリッチなコンシューマーエクスペリエンスの作成が可能になります。コンテンツプロバイダーは、アドビのツールを使用することでインタラクティブなアプリケーションを作成できます。これにより、アドビのサーバー製品をコンテンツ配信インフラストラクチャに統合したり、アドビのエコシステムパートナーと連携したりできます。

コンテンツ保護は、高品質ビデオ配信の重要な要件です。AIR for TV は、コンテンツ保護と収益に繋がるソリューションである Adobe® Flash® Access™ をサポートしています。これにより、主要な映画スタジオなどのコンテンツ所有者が持つ厳格なセキュリティ要件に対応できます。

Flash Access は以下をサポートしています。

- ビデオのストリーミングとダウンロード

- 広告サポート、購読、レンタル、ダウンロード販売などの多様なビジネスモデル
- HTTP Dynamic Streaming、Flash® Media Server を使用した RTMP（Real Time Media Protocol）経由のストリーミング、HTTP によるプログレッシブダウンロードなどの様々なコンテンツ配信技術

AIR for TV には、RTMP の暗号化バージョンである RTMPE 用のサポートが組み込まれています。RTMPE は、セキュリティ要件のより低い既存のストリーミングソリューションのために使用されます。Flash Media Server では、RTMPE および関連する SWF 検証技術がサポートされます。

詳しくは、「Adobe Flash Access」を参照してください。

マルチチャンネルオーディオ

AIR 3 以降の AIR for TV では、HTTP サーバーから徐々にダウンロードされるビデオでマルチチャンネルオーディオがサポートされます。このサポートには、次のコーデックが含まれます。

- AC-3（Dolby Digital）
- E-AC-3（Enhanced Dolby Digital）
- DTS Digital Surround
- DTS Express
- DTS-HD High Resolution Audio
- DTS-HD Master Audio

注意：Adobe Flash Media Server からストリーミングされるビデオについては、マルチチャネルオーディオのサポートはまだ行われていません。

ゲーム入力

AIR 3 以降の AIR for TV では、接続されているジョイスティック、ゲームパッド、棒状のコントローラーなどのゲーム入力デバイスとアプリケーションが通信できる ActionScript API がサポートされています。これらのデバイスはゲーム入力デバイスと呼ばれていますが、ゲームに限らず、すべての AIR for TV アプリケーションで使用できます。

さまざまな機能を持つ幅広いゲーム入力デバイスが使用可能です。したがって、アプリケーションがさまざまな種類の（場合によっては未知の）ゲーム入力デバイスで正常に機能するように、デバイスは API で一般化されます。

GameInput クラスはゲーム入力 ActionScript API へのエントリポイントです。詳しくは、「GameInput」を参照してください。

Stage 3D によって高速化されるグラフィックレンダリング

AIR 3 以降の AIR for TV では、Stage 3D によって高速化されるグラフィックレンダリングがサポートされます。Stage3D の ActionScript API は、低レベルの GPU アクセラレーション API のセットであり、これによって 2D および 3D の高度な機能が実現されます。この低レベルの API により、開発者は GPU のハードウェアアクセラレーションを柔軟に利用でき、パフォーマンスを大きく向上させることができます。また、Stage3D ActionScript API をサポートするゲーム用エンジンを使用することもできます。

詳しくは、「Gaming engines, 3D, and Stage 3D」を参照してください。

ネイティブ拡張

アプリケーションが extendedTV プロファイルを対象とする場合は、ANE（AIR ネイティブ拡張）パッケージを使用できます。

通常、特に AIR がサポートしないデバイス機能を使用するための ANE パッケージはデバイス製造元により提供されます。例えば、ネイティブ拡張を使用して、テレビのチャンネルを変えたり、ビデオプレーヤーの再生を一時停止したりすることができます。

ANE パッケージを使用する AIR for TV アプリケーションをパッケージ化する場合は、AIR ファイルではなく AIRN ファイルにアプリケーションをパッケージ化してください。

AIR for TV デバイスに対するネイティブ拡張は、常に「デバイスバンドル」のネイティブ拡張です。デバイスバンドルとは、拡張ライブラリが AIR for TV デバイスにインストールされていることを意味します。アプリケーションパッケージに含める ANE パッケージには、拡張のネイティブライブラリは含まれません。場合によっては、ActionScript 専用バージョンのネイティブ拡張が含まれることがあります。この ActionScript 専用バージョンは、拡張のスタブまたはシミュレーターです。デバイス製造元が、ネイティブライブラリなどの実際の拡張をデバイスにインストールします。

ネイティブ拡張を開発する際は、次のことに注意してください。

- デバイス用の AIR for TV のネイティブ拡張を作成する場合は、そのデバイスの製造元に必ず相談してください。
- 一部の AIR for TV デバイスでは、デバイス製造元だけがネイティブ拡張を作成します。
- いずれの AIR for TV デバイスでも、インストールするネイティブ拡張はデバイスの製造元が決定します。
- AIR for TV のネイティブ拡張を作成する開発ツールは、製造元によって異なります。

AIR アプリケーションでのネイティブ拡張の使用について詳しくは、147 ページの「Adobe AIR 用ネイティブ拡張の使用」を参照してください。

ネイティブ拡張の作成について詳しくは、『Adobe AIR 用 ActionScript 拡張の開発』を参照してください。

AIR for TV アプリケーションのデザイン上の考慮事項

ビデオに関する考慮事項

ビデオエンコーディングのガイドライン
テレビデバイスにビデオをストリーミングする場合の、推奨されるエンコーディングのガイドラインを次に示します。

ビデオコーデック：	H.264、メインプロファイルまたはハイプロファイル、プログレッシブエンコーディング
解像度：	720i、720p、1080i または 1080p
フレームレート：	1 秒あたり 24 フレームまたは 1 秒あたり 30 フレーム
オーディオコーデック：	AAC-LC または AC-3、44.1 kHz、ステレオ、または以下のマルチチャネルオーディオコーデック：E-AC-3、DTS、DTS Express、DTS-HD High Resolution Audio、または DTS-HD Master Audio
結合ビットレート：	利用可能帯域幅に応じて最大 8M bps
オーディオビットレート：	最大 192 Kbps
ピクセル縦横比：	1 x 1

AIR for TV デバイスに配信されるビデオには H.264 コーデックを使用することをお勧めします。

注意： AIR for TV は Sorenson Spark または On2 VP6 コーデックでエンコードされたビデオもサポートしています。ただし、ハードウェアではこれらのコーデックのデコードや表示は実行されません。代わりに、ランタイムはソフトウェアを使用してこれらのコーデックのデコードや表示を実行します。これが原因で、ビデオはかなり低いフレームレートで再生されます。このような場合は、可能であれば H.264 を使用してください。

StageVideo クラス
AIR for TV は、H.264 でエンコードされたビデオのハードウェアデコードと表示をサポートしています。この機能を有効にするには、StageVideo クラスを使用します。

次の項目について詳しくは、『ActionScript 3.0 開発ガイド』の「ハードウェアアクセラレーションによる表示のための StageVideo クラスの使用」を参照してください。

- StageVideo クラスおよび関連クラスの API
- StageVideo クラスの使用に関する制限事項

Video オブジェクト（H.264 でエンコードされたビデオ用）を使用する既存の AIR アプリケーションに対して最適なサポートを提供するために、AIR for TV では StageVideo オブジェクトを内部で使用しています。これにより、ビデオ再生ではハードウェアデコードおよび表示が活用されます。ただし、Video オブジェクトは StageVideo オブジェクトと同様の制約を受けます。例えば、アプリケーションがビデオを回転しようとしても、ランタイムではなくハードウェアによってビデオが表示されるので、回転は実行されません。

このため、新しいアプリケーションを作成するときには、H.264 でエンコードされたビデオについては StageVideo オブジェクトを使用してください。

StageVideo クラスを使用した例については、「Delivering video and content for the Flash Platform on TV」を参照してください。

ビデオ配信のガイドライン
AIR for TV デバイスでは、ビデオ再生中にネットワークの利用可能な帯域幅が変化することがあります。例えば、他のユーザーが同じインターネット接続の使用を開始したときにこのような変化が起こることがあります。

このため、ビデオ配信システムにはアダプティブビットレート機能を使用することをお勧めします。例えば、サーバー側では、Flash Media Server はアダプティブビットレート機能をサポートしています。クライアント側では、Open Source Media Framework（OSMF）を使用できます。

ネットワーク経由で AIR for TV アプリケーションにビデオコンテンツを配信する場合、次のプロトコルが利用可能です。

- HTTP および HTTPS Dynamic Streaming（F4F 形式）
- RTMP、RTMPE、RTMFP、RTMPT、および RTMPTE Streaming
- HTTP および HTTPS Progressive Download

詳しくは、以下のトピックを参照してください。

- Adobe Flash Media Server デベロッパーズガイド
- Open Source Media Framework

オーディオに関する考慮事項

AIR for TV アプリケーションでの音声再生用 ActionScript には、その他の AIR アプリケーションとの違いはありません。詳しくは、『ActionScript 3.0 開発ガイド』の「サウンドの操作」を参照してください。

AIR for TV でのマルチチャネルオーディオのサポートについては、次のことを考慮してください。

- AIR for TV では、HTTP サーバーから徐々にダウンロードされるビデオでマルチチャンネルオーディオがサポートされます。Adobe Flash Media Server からストリーミングされるビデオについては、マルチチャネルオーディオのサポートはまだ行われていません。
- AIR for TV では多数のオーディオコーデックがサポートされていますが、AIR for TV のすべてのデバイスでオーディオコーデックのセット全体がサポートされているわけではありません。flash.system.Capabilities のメソッド hasMultiChannelAudio() を使用して、AIR for TV デバイスが AC-3 などの特定のマルチチャンネルオーディオコーデックをサポートするかどうかを確認してください。

例えば、サーバーからビデオファイルを徐々にダウンロードするアプリケーションについて考えます。このサーバーには、様々なマルチチャンネルオーディオコーデックをサポートする複数の H.264 ビデオファイルが含まれているとします。アプリケーションでは、hasMultiChannelAudio() を使用することで、サーバーのどのビデオファイルを要求するかが決まります。または、アプリケーションからサーバーに対して、Capabilities.serverString に含まれている文字列を送信することもできます。この文字列は、どのマルチチャンネルオーディオコーデックが使用できるかを示しています。この文字列を使用することで、サーバーは適切なビデオファイルを選択できます。

- DTS オーディオコーデックを使用している場合は、hasMultiChannelAudio() が true を返しても DTS オーディオが再生されないという状況が発生することがあります。

  例えば、S/PDIF 出力を備えたブルーレイプレーヤーを古いアンプに接続している場合について考えてみます。古いアンプは DTS に対応していませんが、S/PDIF にはブルーレイプレーヤーに通知するプロトコルがありません。ブルーレイプレーヤーから古いアンプに DTS ストリームが送信されても、ユーザーには何も聞こえません。したがって、ベストプラクティスとして、DTS を使用するときは、サウンドが再生されていないときにユーザーが指摘できるように、ユーザーインターフェイスを提供してください。そうすれば、アプリケーションは別のコーデックに復帰できます。

次の表は、AIR for TV アプリケーションでそれぞれのオーディオコーデックを使用する状況についてまとめたものです。この表は、AIR for TV デバイスがハードウェアアクセラレーターを使用してオーディオコーデックをデコードする状況も示しています。ハードウェアでデコードすると、パフォーマンスが向上し、CPU の負荷が軽減されます。

オーディオコーデック	AIR for TV での使用	ハードウェアでのデコード	このオーディオコーデックを使用する状況	詳細情報
AAC	常に可	常に可	H.264 でエンコードされたビデオで使用。 インターネットでの音楽ストリーミングサービスなどのオーディオストリーミングが対象。	オーディオ専用 AAC ストリームを使用する場合、MP4 コンテナにオーディオストリームをカプセル化します。
MP3	常に可	不可	アプリケーションの SWF ファイルでのサウンドが対象。 ビデオでは、Sorenson Spark または On2 VP6 でエンコード。	オーディオに mp3 を使用する H.264 ビデオは、AIR for TV デバイスでは再生できません。
AC-3（Dolby Digital） E-AC-3（Enhanced Dolby Digital） DTS Digital Surround DTS Express DTS-HD High Resolution Audio DTS-HD Master Audio	チェック	可	H.264 でエンコードされたビデオで使用。	通常、マルチチャンネルオーディオストリームが AIR for TV から外部オーディオやビデオのレシーバーに渡され、オーディオがデコードされて再生されます。
Speex	常に可	不可	ライブ音声ストリーミングの受信時。	オーディオに Speex を使用する H.264 ビデオは、AIR for TV デバイスでは再生できません。Speex は、Sorenson Spark または On2 VP6 でエンコードされたビデオでのみ使用してください。
NellyMoser	常に可	不可	ライブ音声ストリーミングの受信時。	オーディオに NellyMoser を使用する H.264 ビデオは、AIR for TV デバイスでは再生できません。NellyMoser は、Sorenson Spark または On2 VP6 でエンコードされたビデオでのみ使用してください。

注意：ビデオファイルには、オーディオストリームが 2 つ含まれているものもあります。例えば、1 つのビデオファイルに AAC ストリームと AC3 ストリームの両方を含めることができます。AIR for TV ではこのようなビデオファイルをサポートしていないので、このようなファイルを使用すると、ビデオの音声が出力されない場合があります。

グラフィックのハードウェアアクセラレーション

グラフィックのハードウェアアクセラレーションの使用

AIR for TV デバイスによって、2D グラフィック処理用のハードウェアアクセラレーションが提供されます。デバイスのハードウェアグラフィックアクセラレーターは、CPU の負荷を軽減して、次の操作を実行します。

- ビットマップレンダリング
- ビットマップ拡大／縮小
- ビットマップブレンド
- 長方形の単色の塗り

このグラフィックのハードウェアアクセラレーションによって、AIR for TV アプリケーションのグラフィック操作の多くが高いパフォーマンスを発揮できます。グラフィック操作には、次のものが含まれます。

- スライドトランジション
- 拡大／縮小トランジション
- フェードインとフェードアウト
- アルファによる複数イメージの合成

これらの種類の操作でグラフィックのハードウェアアクセラレーションによるパフォーマンス上のメリットを得るには、次の方法のいずれかを使用してください。

- コンテンツがほとんど変化しない MovieClip オブジェクトおよび他の表示オブジェクトの cacheAsBitmap プロパティを true に設定します。次に、これらのオブジェクトのスライドトランジション、フェードトランジションおよびアルファブレンドを実行します。
- 拡大／縮小や移動（x および y の位置の変更）を実行する表示オブジェクトには cacheAsBitmapMatrix プロパティを使用します。

  拡大／縮小および移動に Matrix クラスの操作を使用することによって、デバイスのハードウェアアクセラレーターがその操作を実行します。または、cacheAsBitmap プロパティが true に設定されている表示オブジェクトのサイズを変更するシナリオを検討してください。サイズを変更するときに、ランタイムのソフトウェアによってビットマップが再描画されます。ソフトウェアによる再描画では、Matrix 操作を使用したハードウェアアクセラレーションによる拡大／縮小よりもパフォーマンスが低下します。

  例えば、エンドユーザーが選択したときにイメージが拡大されて表示されるアプリケーションについて考えてみます。この場合、Matrix の拡大／縮小操作を複数回使用することで、イメージが拡大しているような錯覚が得られます。ただし、オリジナルのイメージと最終のイメージのサイズによっては、最終イメージの画質が要件を満たさないものになることがあります。このため、拡大操作の完了後は、表示オブジェクトのサイズをリセットしてください。cacheAsBitmap が true であるので、ランタイムソフトウェアは表示オブジェクトを再描画しますが、これは一度だけであり、高画質のイメージをレンダリングします。

  注意：通常、AIR for TV デバイスはハードウェアによって高速化される回転と傾斜をサポートしていません。このため、Matrix クラスで回転と傾斜を指定すると、AIR for TV はソフトウェアですべての Matrix 操作を実行します。これらのソフトウェア操作により、パフォーマンスが悪影響を受ける場合があります。
- BitmapData クラスを使用して、カスタムのビットマップキャッシュ機能を作成します。

ビットマップキャッシュ機能について詳しくは、以下のトピックを参照してください。

- 表示オブジェクトのキャッシュ
- ビットマップキャッシュ
- 手動によるビットマップキャッシュ

グラフィックメモリの管理

ハードウェアアクセラレーターは、高速化されるグラフィック操作を実行するために、特別なグラフィックメモリを使用します。アプリケーションですべてのグラフィックメモリを使用する場合、AIR for TV はグラフィックを操作するためのソフトウェアの使用に戻るので、アプリケーションの実行速度は低下します。

アプリケーションによるグラフィックメモリの使用を管理するには：

- イメージや他のビットマップデータの使用が終了したら、関連するグラフィックメモリを解放します。これを実行するには、Bitmap オブジェクトの bitmapData プロパティの dispose() メソッドを呼び出します。次に、例を示します。

  ```
  myBitmap.bitmapData.dispose();
  ```

  注意：BitmapData オブジェクトへの参照を解放しても、グラフィックメモリはすぐには解放されません。ランタイムのガベージコレクターが最終的にグラフィックメモリを解放しますが、dispose() を呼び出すことでアプリケーションをより詳細に制御できます。

- アドビが提供する AIR アプリケーションである PerfMaster Deluxe を使用することで、ターゲットデバイスにおけるグラフィックのハードウェアアクセラレーションについて詳しく理解できます。このアプリケーションでは、様々な操作を実行するための 1 秒あたりのフレーム数が表示されます。PerfMaster Deluxe を使用すれば、同じ操作の異なる実装を比較できます。例えば、ビットマップイメージの移動とベクトルイメージの移動の比較が可能です。PerfMaster Deluxe は「Flash Platform for TV」で入手できます。

表示リストの管理

表示オブジェクトを非表示にするには、オブジェクトの visible プロパティを false に設定します。そうすると、オブジェクトは依然として表示リストに含まれますが、そのオブジェクトは AIR for TV でレンダリングや表示されません。このテクニックは、処理のオーバーヘッドがわずかなので、表示と消去が頻繁に繰り返されるオブジェクトで役立ちます。ただし、visible プロパティを false に設定しても、オブジェクトのリソースは一切解放されません。そのため、オブジェクトの表示を終了したとき、または少なくとも長い間表示しないときは、表示リストからオブジェクトを削除します。さらに、このオブジェクトへの参照をすべて null に設定します。これらのアクションによって、ガベージコレクターがオブジェクトのリソースを解放できます。

PNG 画像および JPEG 画像の使用

アプリケーションで一般的な画像形式は PNG と JPEG の 2 つです。AIR for TV アプリケーションでこれらの画像形式を使用する場合は、次のことを考慮してください。

- 通常、AIR for TV では JPEG ファイルのデコードにハードウェアアクセラレーションを使用します。
- 通常、AIR for TV では PNG ファイルのデコードにソフトウェアを使用します。ソフトウェアによる PNG ファイルのデコードは高速です。
- PNG は、異なるプラットフォーム間で動作し、透明度（アルファチャンネル）機能に対応している唯一のビットマップ形式です。

このため、アプリケーションでは次の画像形式を使用してください。

- ハードウェアによって高速化されるデコードを活用するために、写真には JPEG ファイルを使用します。
- ユーザーインターフェイスエレメントには PNG イメージファイルを使用します。ユーザーインターフェイスエレメントではアルファ設定ができ、ソフトウェアデコードによって、十分に高速なパフォーマンスを実現できます。

AIR for TV アプリケーションのステージ

AIR for TV アプリケーションを作成する際に、Stage クラスを操作する場合は次の点を考慮してください。

- 画面解像度
- 安全表示領域

- ステージの拡大／縮小モード
- ステージの配置
- ステージの表示状態
- 複数の画面サイズのデザイン
- ステージの画質設定

画面解像度

現在では、通常のテレビデバイスの画面解像度は 540p、720p、1080p のいずれかです。これらの画面解像度は ActionScript Capabilities クラスでは次の値となります。

画面解像度	Capabilities.screenResolutionX	Capabilities.screenResolutionY
540p	960	540
720p	1280	720
1080p	1920	1080

特定のデバイス向けにフルスクリーンの AIR for TV アプリケーションを作成する場合は、デバイスの画面解像度に Stage.stageWidth および Stage.stageHeight をハードコーディングします。一方、複数のデバイスで実行するフルスクリーンのアプリケーションを作成する場合は、Capabilities.screenResolutionX プロパティおよび Capabilities.screenResolutionY プロパティを使用して、Stage のサイズを設定します。

次に、例を示します。

```
stage.stageWidth = Capabilities.screenResolutionX;
stage.stageHeight = Capabilities.screenResolutionY;
```

安全表示領域

テレビの「安全表示領域」とは、画面の境界からのインセットとなる画面の領域のことです。この領域は、エンドユーザーが領域全体を見ることのできる十分な距離があり、テレビのベゼル部で隠れる部分のないインセットです。画面周囲にある物理的なフレームであるベゼルは製造元によって異なるため、必要となるインセットも異なります。安全表示領域では、画面の可視領域の確保を試みます。安全表示領域は、「タイトル安全区域」とも呼ばれています。

「オーバースキャン」は、ベゼルの後方にあるので表示されない画面領域です。

画面の各エッジのインセットは 7.5% とすることをお勧めします。次に、例を示します。

画面解像度 1920 x 1080 の場合の安全表示領域

フルスクリーンの AIR for TV アプリケーションをデザインする場合は、次のように常に安全表示領域を考慮してください。

- 背景イメージや背景色などの背景には画面全体を使用します。
- テキスト、グラフィック、ビデオ、ボタンなどのユーザーインターフェイスアイテムを含む重要なアプリケーションエレメントには、安全表示領域のみを使用します。

次の表に、7.5% のインセットを使用した一般的な画面解像度のそれぞれについて、安全表示領域のサイズを示します。

画面解像度	安全表示領域の幅と高さ	左右のインセットの幅	上下のインセットの高さ
960 x 540	816 x 460	72	40
1280 x 720	1088 x 612	96	54
1920 x 1080	1632 x 918	144	81

ただし、ベストプラクティスは常に安全表示領域を動的に計算することです。次に、例を示します。

```
var horizontalInset, verticalInset, safeAreaWidth, safeAreaHeight:int;

horizontalInset = .075 * Capabilities.screenResolutionX;
verticalInset = .075 * Capabilities.screenResolutionY;
safeAreaWidth = Capabilities.screenResolutionX - (2 * horizontalInset);
safeAreaHeight = Capabilities.screenResolutionY - (2 * verticalInset);
```

ステージの拡大／縮小モード

Stage.scaleMode を StageScaleMode.NO_SCALE に設定し、ステージのサイズ変更イベントを監視します。

```
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.addEventListener(Event.RESIZE, layoutHandler);
```

この設定によって、ステージ座標がピクセル単位の座標と同じになります。この設定を FULL_SCREEN_INTERACTIVE 表示状態および TOP_LEFT ステージ配置と共に使用すると、安全表示領域を効果的に使用できます。

特に、フルスクリーンアプリケーションでは、この拡大／縮小モードは、Stage クラスの stageWidth プロパティと stageHeight プロパティが、Capabilities クラスの screenResolutionX プロパティと screenResolutionY プロパティに一致するということを意味します。

さらに、アプリケーションのウィンドウサイズが変更されても、ステージコンテンツは定義されたサイズで維持されます。ランタイムはレイアウトや拡大／縮小を自動的に実行しません。また、ウィンドウサイズが変更された場合、ランタイムは Stage クラスの resize イベントを送出します。このため、アプリケーションの開始時およびアプリケーションウィンドウのサイズ変更時におけるアプリケーションのコンテンツの調整方法を完全に制御できます。

注意：NO_SCALE の動作はあらゆる AIR アプリケーションで同様です。ただし、AIR for TV アプリケーションでは、この設定の使用が安全表示領域を使用する上で不可欠になります。

ステージの配置

Stage.align を StageAlign.TOP_LEFT に設定します。

```
stage.align = StageAlign.TOP_LEFT;
```

この配置により、画面の左上隅に 0,0 座標が配置されます。これは、ActionScript を使用したコンテンツの配置に便利です。

この設定を NO_SCALE 拡大／縮小モードと FULL_SCREEN_INTERACTIVE 表示状態と共に使用すると、安全表示領域を効果的に使用できます。

ステージの表示状態

フルスクリーンの AIR for TV アプリケーションの Stage.displayState を StageDisplayState.FULL_SCREEN_INTERACTIVE に設定します。

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

この値により、ユーザー入力が許可され、画面全体にステージを拡大するように AIR アプリケーションが設定されます

FULL_SCREEN_INTERACTIVE 設定を使用することをお勧めします。この設定を NO_SCALE 拡大／縮小モードと TOP_LEFT ステージ配置と共に使用すると、安全表示領域を効果的に使用できます。

このため、フルスクリーンのアプリケーションの場合は、メインのドキュメントクラスの ADDED_TO_STAGE イベントのハンドラー内で、次のように指定します。

```
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;
}
```

次に、RESIZE イベントのハンドラー内で、次のように指定します。

- 画面解像度のサイズとステージの幅および高さを比較します。同じ場合には、ステージ表示状態が FULL_SCREEN_INTERACTIVE に変更されたことが原因で RESIZE イベントが発生しています。
- 安全表示領域および対応するインセットのサイズを計算して保存します。

```
private function onResize(evt:Event):void
{
    if ((Capabilities.screenResolutionX == stage.stageWidth) &&
        (Capabilities.screenResolutionY == stage.stageHeight))
    {

        // Calculate and save safe viewing area dimensions.
    }
}
```

ステージの横と縦のサイズが Capabilities.screenResolutionX と screenResolutionY に等しい場合、AIR for TV によってハードウェアで可能な最高のビデオとグラフィックの再現性がもたらされます。

注意：グラフィックおよびビデオをテレビ画面に表示したときのサイズは、Capabilities.screenResolutionX 値と screenResolutionY 値で指定されたサイズとは異なる可能性があります。これらの値は、AIR for TV を実行するデバイスによって決まります。例えば、AIR for TV を実行するセットトップボックスのスクリーン解像度が 1280 x 720 で、接続されたテレビのスクリーン解像度が 1920 x 1080 の場合があります。ただし、AIR for TV によってハードウェアで可能な最高の再現性がもたらされます。したがって、この例では、ハードウェアによって、スクリーン解像度 1920 x 1080 を使用して 1080p ビデオが表示されます。

複数の画面サイズのデザイン

複数の AIR for TV デバイス上で動作し正常に表示される、フルスクリーンの AIR for TV アプリケーションを開発することができます。次の手順を実行してください。

1. ステージプロパティの scaleMode、align および displayState を、それぞれ、StageScaleMode.NO_SCALE、StageAlign.TOP_LEFT、StageDisplayState.FULL_SCREEN_INTERACTIVE という推奨値に設定します。
2. Capabilities.screenResolutionX および Capabilities.screenResolutionY に基づいて安全表示領域を設定します。
3. 安全表示領域の幅と高さに従ってコンテンツのサイズとレイアウトを調節します。

   コンテンツのオブジェクトは大きなものとなります（特にモバイルデバイスアプリケーションと比較した場合）。ただし、動的レイアウト、相対位置指定およびアダプティブコンテンツなどの概念に変わりはありません。これらの概念をサポートする ActionScript について詳しくは、「複数の画面サイズ向けのモバイル Flash コンテンツのオーサリング」を参照してください。

ステージの画質

AIR for TV アプリケーション向けの Stage.quality プロパティは、常に StageQuality.High です。これを変更することはできません。

このプロパティは、すべての Stage オブジェクトのレンダリング画質を指定します。

リモートコントローラー入力処理

通常、ユーザーはリモートコントローラーを使用して AIR for TV アプリケーションを操作します。ただし、キー入力の処理は、デスクトップアプリケーションのキーボードからのキー入力と同様に処理します。具体的には、KeyboardEvent.KEY_DOWN イベントを処理してください。詳しくは、『ActionScript 3.0 開発ガイド』の「キーボード入力のキャプチャ」を参照してください。

リモートコントローラーの各キーは、ActionScript 定数にマップされます。例えば、リモートコントローラーの方向キーパッドのキーは次のようにマップされます。

リモートコントローラーの方向キーパッドのキー	ActionScript 3.0 定数
上	Keyboard.UP
下	Keyboard.DOWN
左	Keyboard.LEFT
右	Keyboard.RIGHT
「OK」または「選択」	Keyboard.ENTER

AIR 2.5 では、リモートコントローラー入力をサポートするために、多くの Keyboard 定数が追加されています。これらの定数の一覧については、『Adobe Flash Platform 用 ActionScript 3.0 リファレンスガイド』の Keyboard クラスを参照してください。

アプリケーションが可能な限り多くのデバイスで動作するように、以下の実施をお勧めします。

- 可能であれば、方向キーパッドのキーのみを使用します。

  リモートコントローラーデバイスによってキーのセットは異なりますが、キーのセットには通常、方向キーパッドのキーが含まれています。

  例えば、通常 Blu-ray プレーヤーのリモートコントローラーには、「上のチャンネル」および「下のチャンネル」キーが含まれていません。再生、一時停止、停止のキーも、すべてのリモートコントローラーに組み込まれているとは限りません。

- アプリケーションで方向キーパッドのキー以外のキーが必要な場合は、メニューキーおよび情報キーを使用します。

  メニューキーと情報キーは、リモートコントローラーでは方向キーパッドのキーに次いで一般的なキーです。

- 汎用リモートコントローラーの使用頻度を考慮します。

  特定のデバイス向けにアプリケーションを作成している場合でも、多くのユーザーはデバイスに付属するリモートコントローラーを使用しないことを認識してください。このようなユーザーは、代わりに汎用リモートコントローラーを使用しています。また、ユーザーがデバイスのリモートコントローラーのすべてのキーと一致するように、汎用リモートコントローラーをプログラムしているとは限りません。このため、最も一般的なキーのみを使用することをお勧めします。

- ユーザーが方向キーパッドのキーの 1 つを使用して問題を回避できるようにします。

  また場合によっては、アプリケーションで、リモートコントローラー上の最も一般的なキー以外のキーを使用する必要もあります。方向キーパッドのキーの 1 つを使用した回避ルートがあれば、すべてのデバイスでアプリケーションの動作が安定します。

- ターゲットの AIR for TV デバイスにポインター入力機能があることがわかっていない限り、ポインター入力を要求しないでください。

  多くのデスクトップアプリケーションはマウス入力を前提にしていますが、ほとんどのテレビはポインター入力をサポートしていません。したがって、デスクトップアプリケーションをテレビで動作するように変更する場合は、マウス入力を要求しないようにアプリケーションを変更します。こうした変更には、イベント処理の変更およびユーザーに対する操作指示の変更が含まれます。例えば、アプリケーションの起動画面が表示されるときに、「クリックして開始します」というテキストを表示しないようにします。

フォーカスの管理

ユーザーインターフェイスエレメントにデスクトップアプリケーションのフォーカスが含まれている場合、フォーカスはキーボードイベントやマウスイベントなどのユーザー入力イベントのターゲットになります。さらに、アプリケーションではユーザーインターフェイスエレメントをフォーカスでハイライト表示します。AIR for TV アプリケーションでのフォーカスの管理は、次の理由により、デスクトップアプリケーションでのフォーカスの管理と異なります。

- デスクトップアプリケーションでは多くの場合、Tab キーを使用して次のユーザーインターフェイスエレメントにフォーカスを変更します。Tab キーの使用は、AIR for TV アプリケーションには適用されません。通常、リモートコントローラーデバイスには Tab キーがありません。このため、デスクトップなどでの DisplayObject の tabEnabled プロパティを使用したフォーカスの管理は適用されません。
- デスクトップアプリケーションでは多くの場合、マウスを使用して任意のユーザーインターフェイスエレメントにフォーカスを設定します。

このため、アプリケーションでは、以下のことを行ってください。

- KeyboardEvent.KEY_DOWN などの Keyboard イベントを監視するイベントリスナーを Stage に追加します。

- エンドユーザーに対してどのユーザーインターフェイスエレメントをハイライト表示するかを判断するアプリケーションロジックを記述します。アプリケーションの開始時にユーザーインターフェイスエレメントをハイライト表示するようにします。
- アプリケーションロジックに基づいて、適切なユーザーインターフェイスエレメントオブジェクトに、Stage が受信した Keyboard イベントを送出します。

  Stage.focus または Stage.assignFocus() を使用して、ユーザーインターフェイスエレメントにフォーカスを割り当てることもできます。その後で、イベントリスナーがキーボードのイベントを受信するように、その DisplayObject にイベントリスナーを追加します。

ユーザーインターフェイスデザイン

AIR for TV アプリケーションのユーザーインターフェイスがテレビで正しく機能するように、次のことに関する推奨事項を取り入れてください。

- アプリケーションの応答性
- アプリケーションのユーザビリティ
- ユーザーの個性と期待

応答性

AIR for TV アプリケーションの応答性を可能な限り適切なものにするには、次のヒントを参考にしてください。

- アプリケーションの初期 SWF ファイルをなるべく小さなサイズにします。

  初期 SWF ファイルでは、アプリケーションを開始するために必要なリソースのみをロードするようにします。例えば、アプリケーションの起動画面イメージのみをロードします。

  この推奨事項はデスクトップ AIR アプリケーションでも有効ですが、AIR for TV デバイスではさらに重要になります。例えば、AIR for TV デバイスには、デスクトップコンピューターに相当する処理能力はありません。また、AIR for TV デバイスではアプリケーションをフラッシュメモリに保存しますが、フラッシュメモリはデスクトップコンピューターのハードディスクほどアクセス速度が早くありません。

- アプリケーションが少なくとも 1 秒あたり 20 フレームのフレームレートで実行するようにします。

  この目標を達成するようにグラフィックをデザインしてください。グラフィック操作が複雑になると、1 秒あたりのフレーム数に影響することがあります。レンダリングのパフォーマンスを向上するヒントについては、『Flash Platform のパフォーマンスの最適化』を参照してください。

  注意：通常、AIR for TV デバイスのグラフィックハードウェアは 60 Hz または 120 Hz（1 秒あたり 60 回または 120 回）のレートで画面を更新します。例えば、ハードウェアは、60 Hz または 120 Hz の画面で表示する場合に、1 秒あたり 30 フレームまたは 60 フレームで、更新のためにステージをスキャンします。ただし、ユーザーがこのような高いフレームレートを利用できるかどうかは、アプリケーションのグラフィック操作の複雑さによって異なります。

- ユーザー入力の 100 ～ 200 ミリ秒以内に画面を更新します。

  更新に時間がかかるとユーザーは待ちきれなくなり、キーを複数回押してしまう場合があります。

ユーザビリティ

AIR for TV アプリケーションのユーザーは、「リビングルーム」環境にいます。テレビから約 3 メートル離れたところに座ってテレビを操作します。また、部屋が暗い場合もあります。通常、ユーザーはリモートコントローラーデバイスを使用して入力します。複数のユーザーがこのアプリケーションを同時に使用したり、順番に使用したりする場合があります。

このため、テレビのユーザビリティを考慮してユーザーインターフェイスをデザインする場合は、次のことを考慮してください。

- ユーザーインターフェイスエレメントを大きなサイズに設定します。

  テキスト、ボタンなどのユーザーインターフェイスエレメントをデザインするときは、ユーザーがテレビから離れて座っていることを考慮してください。例えば、約 3 メートル離れた場所からでもすべてのエレメントが見やすく、読みやすいように指定します。画面が大きいという理由だけで、画面に多くのエレメントを表示しないようにしてください。
- 適切なコントラストを使用して、テレビから離れた場所からでも見やすく、読みやすいコンテンツにします。
- フォーカスが設定されているユーザーインターフェイスエレメントを明るくして、そのエレメントを強調します。
- 必要な場合のみモーションを使用します。例えば、連続性を示すために画面間をスライドして表示することが便利な場合があります。ただし、モーションがユーザーのナビゲーションに役立っていない場合や、アプリケーションにとって不要である場合は、モーションが邪魔になることもあります。
- ユーザーインターフェイス全体を対象として、ユーザーが前の状態に戻るための明確な方法を必ず提示します。

リモートコントローラーの使用について詳しくは、129 ページの「リモートコントローラー入力処理」を参照してください。

ユーザーの個性と期待

一般的に、AIR for TV アプリケーションのユーザーは、楽しくリラックスした環境でテレビ向けのエンターテインメントを求めていることを考慮してください。ユーザーがコンピューターやテクノロジーに詳しいとは限りません。

このため、次の特徴を持つように、AIR for TV アプリケーションをデザインします。

- 技術用語は使用しないでください。
- モーダルダイアログの使用は避けてください。
- 職場や技術的な環境ではなくリビングルーム環境に適した、フレンドリーで形式張らない表現によって説明を示します。
- テレビ視聴者の要求に応える製品レベルの高画質なグラフィックを使用します。
- リモートコントローラーデバイスで簡単に操作できるユーザーインターフェイスを作成します。デスクトップまたはモバイル向けのアプリケーションに適したユーザーインターフェイスやデザインエレメントは使用しないでください。たとえば、デスクトップデバイスやモバイルデバイス上のユーザーインターフェイスでは、マウスや指でボタンを指したりクリックすることが行われます。

フォントとテキスト

AIR for TV アプリケーションでは、デバイスフォントと埋め込みフォントのいずれかを使用できます。

デバイスフォントは、デバイスにインストールされているフォントです。すべての AIR for TV デバイスには次のデバイスフォントがインストールされています。

フォント名	説明
_sans	_sans デバイスフォントは、sans-serif 書体です。すべての AIR for TV デバイスにインストールされている _sans デバイスフォントは、Myriad Pro です。通常、テレビでは sans-serif 書体の方が serif 書体よりも見栄えがよくなります。これは、視聴するときの距離が理由です。
_serif	_serif デバイスフォントは、serif 書体です。すべての AIR for TV デバイスにインストールされている _serif デバイスフォントは Minion Pro です。
_typewriter	_typewriter デバイスフォントは、等幅フォントです。すべての AIR for TV デバイスにインストールされている _typewriter デバイスフォントは、Courier Std です。

すべての AIR for TV デバイスには次のアジア地域のデバイスフォントもインストールされています。

フォント名	言語	書体カテゴリー	ロケールコード
RyoGothicPlusN-Regular	日本語	sans	ja
RyoTextPlusN-Regular	日本語	serif	ja
AdobeGothicStd-Light	韓国語	sans	ko
AdobeHeitiStd-Regular	簡体字中国語	sans	zh_CN
AdobeSongStd-Light	簡体字中国語	serif	zh_CN
AdobeMingStd-Light	繁体字中国語	serif	zh_TW および zh_HK

これらの AIR for TV デバイスフォントには、次の特徴があります。

- Adobe® Type Library のフォントである
- テレビ上できれいに表示される
- ビデオの字幕設定用にデザインされている
- フォントアウトラインであり、ビットマップフォントではない

注意：多くの場合、デバイス製造元はその他のデバイスフォントもデバイスに含めています。AIR for TV デバイスフォントに加えて、このような製造元提供のデバイスフォントもインストールされます。

アドビでは、デバイス上のすべてのデバイスフォントを表示する FontMaster Deluxe というアプリケーションを提供しています。このアプリケーションは、「Flash Platform for TV」で入手できます。

また、AIR for TV アプリケーションにフォントを埋め込むこともできます。埋め込みフォントについて詳しくは、『ActionScript 3.0 開発ガイド』の「高度なテキストレンダリング」を参照してください。

TLF テキストフィールドの使用については、以下の実施をお勧めします。

- アプリケーションの実行対象となっているロケールを活用するために、アジア言語のテキストには TLF テキストフィールドを使用します。TLFTextField オブジェクトに関連付けられた TextLayoutFormat オブジェクトの locale プロパティを設定します。現在のロケールを判断するには、『ActionScript 3.0 開発ガイド』の「ロケールの選択」を参照してください。
- フォントが AIR for TV デバイスフォントに含まれていない場合は、TextLayoutFormat オブジェクトの fontFamily プロパティにフォント名を指定します。デバイス上でこのフォントを使用できる場合は、AIR for TV はこれを使用します。要求したフォントがデバイス上にない場合は、locale 設定に従って、AIR for TV では、適切な AIR for TV デバイスフォントを代わりに使用します。
- AIR for TV が適切な AIR for TV デバイスフォントを選択できるように、locale プロパティの設定と共に、fontFamily プロパティに _sans、_serif または _typewriter を指定します。ロケールに従って、AIR for TV は、アジア地域のデバイスフォントのセットまたはアジア地域以外のデバイスフォントのセットからフォントを選択します。これらの設定により、アジア地域の主要な 4 つのロケールおよび英語に対して、適切なフォントを簡単かつ自動的に使用できるようになります。

注意：アジア地域の言語テキストにクラシックテキストフィールドを使用する場合は、適切なレンダリングが実行されるように、AIR for TV デバイスフォントのフォント名を指定してください。ターゲットデバイスに他のフォントがインストールされていることがわかっている場合には、そのフォントを指定することもできます。

アプリケーションのパフォーマンスについて、次の点を考慮してください。

- クラシックテキストフィールドは、TLF テキストフィールドよりもパフォーマンスが高速になります。
- ビットマップフォントを使用するクラシックテキストフィールドは、最高のパフォーマンス速度を実現します。

ビットマップフォントは、アウトラインフォントとは異なり、各文字のビットマップを提供します。これに対してアウトラインフォントは、各文字のアウトラインデータのみを提供します。デバイスフォントも埋め込みフォントもビットマップフォントにすることができます。

- デバイスフォントを指定する場合は、デバイスフォントがターゲットデバイスにインストールされていることを確認します。デバイスにインストールされていない場合は、AIR for TV では、デバイスにインストールされている他のフォントを検索し使用します。ただし、この動作によりアプリケーションのパフォーマンス速度が低下します。
- 他の表示オブジェクトと同様に、TextField オブジェクトがほとんど変化しない場合は、オブジェクトの cacheAsBitmap プロパティを true に設定します。この設定によって、フェード、スライド、アルファブレンドなどのトランジションのパフォーマンスが向上します。拡大／縮小および移動には、cacheAsBitmapMatrix を使用します。詳しくは、124 ページの「グラフィックのハードウェアアクセラレーション」を参照してください。

ファイルシステムセキュリティ

AIR for TV アプリケーションは AIR アプリケーションであり、デバイスのファイルシステムにアクセスできます。ただし、「リビングルーム」デバイス上では、アプリケーションがデバイスのシステムファイルや他のアプリケーションのファイルにアクセスできなくすることが非常に重要となります。テレビや関連するデバイスのユーザーは、テレビを見ているのであり、デバイスの障害を予期したり我慢することはありません。

このため、AIR for TV アプリケーションに対しては、デバイスのファイルシステムの閲覧が制限されます。ActionScript 3.0 を使用すると、アプリケーションに対して、特定のディレクトリ（およびそのサブディレクトリ）だけにアクセスを許可することができます。また、ActionScript で使用するディレクトリ名は、デバイス上の実際のディレクトリ名ではありません。この追加レイヤーによって、関係のないローカルファイルへの不当なアクセスや不注意なアクセスを行わないよう AIR for TV アプリケーションを保護します。

詳しくは、「AIR for TV アプリケーションのディレクトリ表示」を参照してください。

AIR アプリケーションサンドボックス

AIR for TV アプリケーションは、「AIR アプリケーションサンドボックス」の説明にあるように、AIR アプリケーションサンドボックス内で実行されます。

AIR for TV アプリケーションにおける唯一の相違点は、ファイルシステムへのアクセスが制限されるということです（134 ページの「ファイルシステムセキュリティ」を参照）。

アプリケーションのライフサイクル

デスクトップ環境とは異なり、エンドユーザーは、AIR for TV アプリケーションが実行中のウィンドウを閉じることはできません。このため、アプリケーションを終了するためのユーザーインターフェースメカニズムを取り入れる必要があります。

通常、デバイスでは、エンドユーザーはリモートコントローラーの終了キーを使用してアプリケーションを無条件に終了することができます。しかし、AIR for TV は flash.events.Event.EXITING イベントをアプリケーションに送出しません。このため、アプリケーションの次回起動時に問題のない状態で復元できるよう、アプリケーションの状態を頻繁に保存してください。

HTTP Cookie

AIR for TV は、HTTP 永続 Cookie とセッション Cookie をサポートします。AIR for TV では、各 AIR アプリケーションの Cookie を、アプリケーション固有のディレクトリに保存します。

```
/app-storage/<app id>/Local Store
```

Cookie ファイルには cookies という名前が付けられます。

注意：デスクトップデバイスなどの他のデバイス上の AIR では、Cookie はアプリケーションごとに保存されません。アプリケーションごとに Cookie が保存されることにより、AIR for TV のアプリケーションとシステムのセキュリティモデルがサポートされます。

次のように、ActionScript の URLRequest.manageCookies プロパティを使用します。

- manageCookies を true に設定します。この値はデフォルトです。これは、AIR for TV が自動的に HTTP 要求に Cookie を追加し、HTTP 応答内に Cookie を記録しておくことを意味します。

  注意：manageCookies が true の場合でも、アプリケーションでは URLRequest.requestHeaders を使用することで HTTP 要求に手動で Cookie を追加できます。この Cookie の名前が、AIR for TV で管理している Cookie の名前と同じ場合、要求には 2 つの Cookie が同じ名前で格納されます。この 2 つの Cookie には異なる値を設定できます。

- manageCookies を false に設定します。この値は、アプリケーションが、HTTP 要求内の Cookie を送信し、その Cookie を HTTP 応答内に記録することを意味します。

詳しくは、「URLRequest」を参照してください。

AIR for TV アプリケーションの開発ワークフロー

次の Adobe Flash Platform 開発ツールを使用して、AIR for TV アプリケーションを開発できます。

- Adobe Flash Professional

  Adobe Flash Professional CS5.5 は、AIR for TV アプリケーションをサポートする最初のバージョンの AIR である、AIR 2.5 for TV をサポートします。

- Adobe Flash® Builder®

  Flash Builder 4.5 は AIR 2.5 for TV をサポートします。

- AIR SDK

  AIR 2.5 以降では、AIR SDK に用意されているコマンドラインツールを使用して、アプリケーションを開発できます。AIR SDK のダウンロードについては、http://www.adobe.com/jp/products/air/sdk/ を参照してください。

Flash Professional の使用

Flash Professional を使用した AIR for TV アプリケーションの開発、テストおよびパブリッシュは、AIR デスクトップアプリケーション用のツールを使用する場合と類似しています。

ただし、ActionScript 3.0 コードを記述するときは、tv および extendedTV の各 AIR プロファイルがサポートするクラスとメソッドのみを使用してください。詳しくは、241 ページの「デバイスプロファイル」を参照してください。

プロジェクト設定

AIR for TV アプリケーションのプロジェクトを設定するには、次の操作を実行してください。

- パブリッシュ設定ダイアログボックスの「Flash」タブで、「Player」の値を「AIR 2.5」以上に設定します。
- Adobe AIR 設定ダイアログボックス（アプリケーションとインストーラーの設定）の「一般」タブで、プロファイルを「テレビ」または「拡張テレビ」に設定します。

デバッグ

Flash Professional の AIR Debug Launcher を使用してアプリケーションを実行できます。次の手順を実行してください。

- デバッグモードでアプリケーションを実行するには、次のメニューを選択します。

  デバッグ／ムービーをデバッグ／ AIR Debug Laucher（デスクトップ）

  このメニューを一度選択すると、その後のデバッグの実行時に、次のメニューを選択できるようになります。

  デバッグ／ムービーをデバッグ／デバッグ

- デバッグモード機能を使用せずにアプリケーションを実行するには、次のメニューを選択します。

  制御／ムービープレビュー／ AIR Debug Launcher（デスクトップ）を使用

  このメニューを一度選択すると、その後の実行時に、制御／ムービープレビュー／プレビューを選択できるようになります。

AIR プロファイルをテレビまたは拡張テレビに設定しているので、AIR Debug Launcher には「リモートコントロールボタン」というメニューが表示されます。このメニューを使用して、リモートコントローラーデバイス上のキーを押す操作をシミュレートできます。

詳しくは、144 ページの「Flash Professional によるリモートデバッグ」を参照してください。

ネイティブ拡張の使用

アプリケーションでネイティブ拡張を使用する場合は、149 ページの「ネイティブ拡張を使用するためのタスクリスト」の説明に従ってください。

ただし、アプリケーションでネイティブ拡張を使用している場合は、次の制限があります。

- Flash Professional を使用してアプリケーションをパブリッシュすることはできません。アプリケーションをパブリッシュするには、ADT を使用してください。141 ページの「ADT によるパッケージ化」を参照してください。
- Flash Professional を使用してアプリケーションを実行またはデバッグすることはできません。開発マシン上でアプリケーションをデバッグするには、ADL を使用してください。142 ページの「ADL を使用したデバイスシミュレーション」を参照してください。

Flash Builder の使用

Flash Builder 4.5 以降、Flash Builder では AIR for TV の開発をサポートしています。Flash Builder を使用した AIR for TV アプリケーションの開発、テストおよびパブリッシュは、AIR デスクトップアプリケーション用のツールを使用した場合と類似しています。

アプリケーションの設定

アプリケーションは以下のように設定してください。

- MXML ファイルを使用している場合は、MXML ファイルのコンテナクラスとして Application エレメントを使用します。

```
<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>.
```

  重要：AIR for TV アプリケーションは、WindowedApplication エレメントをサポートしていません。

注意：MXML ファイルを使用する必要はありません。代わりに、ActionScript 3.0 プロジェクトを作成できます。

- tv および extendedTV の各 AIR プロファイルがサポートする ActionScript 3.0 クラスとメソッドのみを使用します。詳しくは、241 ページの「デバイスプロファイル」を参照してください。

さらに、アプリケーションの XML ファイルでは、次のように設定します。

- application エレメントの xmlns 属性を AIR 2.5 に設定します。

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

- supportedProfiles エレメントには、tv または extendedTV を含めます。

```
<supportedProfiles>tv</supportedProfiles>
```

アプリケーションのデバッグ

Flash Builder の AIR Debug Launcher を使用してアプリケーションを実行できます。次の手順を実行してください。

1. 実行／デバッグの構成を選択します。
2. 「プロファイル」フィールドを「デスクトップ」に設定します。
3. 実行／デバッグを選択してデバッグモードで実行するか、実行／実行を選択してデバッグモード機能を使用せずに実行します。

supportedProfiles エレメントをテレビまたは拡張テレビに設定しているので、AIR Debug Launcher には「リモートコントロールボタン」というメニューが表示されます。このメニューを使用して、リモートコントローラーデバイス上のキーを押す操作をシミュレートできます。

詳しくは、144 ページの「Flash Builder によるリモートデバッグ」を参照してください。

ネイティブ拡張の使用

アプリケーションでネイティブ拡張を使用する場合は、149 ページの「ネイティブ拡張を使用するためのタスクリスト」の説明に従ってください。

ただし、アプリケーションでネイティブ拡張を使用している場合は、次の制限があります。

- Flash Builder を使用してアプリケーションをパブリッシュすることはできません。アプリケーションをパブリッシュするには、ADT を使用してください。141 ページの「ADT によるパッケージ化」を参照してください。
- Flash Builder を使用してアプリケーションを実行またはデバッグすることはできません。開発マシン上でアプリケーションをデバッグするには、ADL を使用してください。142 ページの「ADL を使用したデバイスシミュレーション」を参照してください。

AIR for TV アプリケーション記述子のプロパティ

他の AIR アプリケーションと同様に、アプリケーション記述ファイルで基本的なアプリケーションプロパティを設定します。テレビプロファイルアプリケーションでは、ウィンドウサイズや透明度など、デスクトップ固有のプロパティのいくつかが無視されます。extendedTV プロファイル内のデバイスをターゲットにしているアプリケーションは、ネイティブ拡張を使用できます。これらのアプリケーションは、extensions エレメントに使用されているネイティブ拡張を識別します。

一般設定

アプリケーション記述子の設定のいくつかは、すべてのテレビアプリケーションで重要なものです。

必要な AIR ランタイムのバージョン

アプリケーション記述ファイルの名前空間を使用して、アプリケーションに必要な AIR ランタイムのバージョンを指定します。

application エレメントに割り当てられている名前空間によって、アプリケーションが使用する機能の大部分が決まります。例えば、アプリケーションでは AIR 2.5 名前空間を使用しているが、ユーザーが将来のバージョンの AIR をインストールしている場合を考えてみましょう。この場合、将来のバージョンの AIR では動作が異なっていても、アプリケーションでは AIR 2.5 の動作を参照します。名前空間を変更し、アップデートをパブリッシュした場合にのみ、アプリケーションは新しい動作と機能にアクセスできます。セキュリティの修正は、この規則には該当しない重要な例外です。

ルートの application エレメントの xmlns 属性を使用して、名前空間を指定します。

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

AIR 2.5 は、テレビアプリケーションをサポートする最初の AIR バージョンです。

アプリケーション ID

いくつかの設定は、パブリッシュするアプリケーションごとに一意にする必要があります。例えば、id エレメント、name エレメントおよび filename エレメントの設定が該当します。

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

アプリケーションのバージョン

versionNumber エレメントでアプリケーションのバージョンを指定します。versionNumber の値を指定するとき、一連の数値をドットで 3 つまで区切って指定することができます。例えば、「0.1.2」のようにします。バージョン番号の各セグメントには、3 桁までの数字を指定できます（つまり、「999.999.999」が、許可される最大のバージョン番号になります）。番号に 3 つのセグメントをすべて含める必要はありません。「1」や「1.0」なども有効なバージョン番号です。

versionLabel エレメントを使用して、バージョンのラベルを指定することもできます。バージョンのラベルを追加すると、バージョン番号の代わりに表示されます。

```
<versionNumber>1.23.7<versionNumber>
<versionLabel>1.23 Beta 7</versionLabel>
```

メインアプリケーションの SWF

メインアプリケーションの SWF ファイルを、initialWindow エレメントの子である versionLabel に指定します。tv プロファイルでデバイスをターゲットにするとき、SWF ファイルを使用する必要があります（HTML ベースのアプリケーションはサポートされていません）。

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

ファイルは AIR パッケージに含める必要があります（ADT または IDE を使用します）。アプリケーション記述子内の名前を参照するだけでは、ファイルはパッケージに自動的には取り込まれません。

メイン画面のプロパティ

initialWindow エレメントの子エレメントのいくつかは、メインアプリケーション画面の最初の外観と動作を制御します。これらのプロパティのほとんどはテレビプロファイル内のデバイスでは無視されますが、次の fullScreen エレメントは使用することができます。

- fullScreen - アプリケーションが、デバイスの画面全体を使用するか、通常のオペレーティングシステムクロムと画面を共有するかを指定します。

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

visible エレメント

visible エレメントは、initialWindow エレメントの子エレメントです。AIR for TV デバイスではアプリケーションのコンテンツが常に表示されるため、AIR for TV では visible エレメントは無視されます。

ただし、アプリケーションがデスクトップデバイスもターゲットにしている場合は、visible エレメントを true に設定します。

デスクトップデバイスで、このエレメントのデフォルト値は false です。したがって、visible エレメントを含めないと、アプリケーションのコンテンツはデスクトップデバイスで表示されなくなります。ActionScript クラスの NativeWindow を使用することによってデスクトップデバイスにコンテンツを表示することもできますが、テレビのデバイスプロファイルでは NativeWindow クラスはサポートされません。AIR for TV デバイスで実行されているアプリケーションで NativeWindow クラスを使用すると、そのアプリケーションは読み込めなくなります。NativeWindow クラスのメソッドを呼び出すかどうかに関係なく、このクラスを使用しているアプリケーションを AIR for TV デバイスに読み込むことはできません。

サポートされるプロファイル

アプリケーションがテレビデバイスにのみ対応する場合は、他の種類のコンピューティングデバイスへのインストールを防ぐことができます。supportedProfiles エレメント内のサポートされるプロファイルのリストから、その他のプロファイルを除外します。

```
<supportedProfiles>tv extendedTV</supportedProfiles>
```

アプリケーションでネイティブ拡張を使用している場合は、サポートされるプロファイルリストに extendedTV プロファイルのみを含めます。

```
<supportedProfiles>extendedTV</supportedProfiles>
```

supportedProfiles エレメントを省略すると、アプリケーションはすべてのプロファイルをサポートするものと見なされます。

supportedProfiles リストに tv プロファイルのみを含めることは避けてください。一部のテレビデバイスでは、AIR for TV は常に、extendedTV プロファイルに対応するモードで実行されます。この動作は、ネイティブ拡張を使用するアプリケーションを実行できるようにするためです。supportedProfiles エレメントで tv しか指定しないと、コンテンツに AIR for TV の extendedTV 用モードとの互換性がないと宣言していることになります。tv プロファイルしか指定されていないアプリケーションが一部のテレビデバイスで読み込まれないのは、そのためです。

tv および extendedTV プロファイルでサポートされている ActionScript クラスの一覧については、242 ページの「様々なプロファイルの機能」を参照してください。

必要なネイティブ拡張

extendedTV プロファイルをサポートするアプリケーションでは、ネイティブ拡張を使用できます。

アプリケーション記述子の extensions エレメントおよび extensionID エレメントを使用して、AIR アプリケーションで使用するすべてのネイティブ拡張を宣言します。次の例に、2 つの必須のネイティブ拡張を指定するシンタックスを示します。

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

拡張がリストにリストされていない場合、アプリケーションはその拡張を使用できません。

extensionID エレメントは、拡張記述ファイルの id エレメントと同じ値です。拡張記述ファイルは、extension.xml という名前の XML ファイルです。これは、デバイス製造元から受け取る ANE ファイル内にパッケージ化されています。

拡張を extensions エレメントにリストしても、AIR for TV デバイスにその拡張がインストールされていない場合、アプリケーションは実行できません。このルールの例外は、AIR for TV アプリケーションと共にパッケージする ANE ファイルに、その拡張のスタブバージョンが含まれている場合です。この場合、アプリケーションは実行でき、拡張のスタブバージョンが使用されます。スタブバージョンには ActionScript コードが含まれますが、ネイティブコードは含まれません。

アプリケーションアイコン

テレビデバイスでのアプリケーションアイコンの要件は、デバイスに依存します。例えば、デバイス製造元は次の要件を指定します。

- 必須のアイコンとアイコンのサイズ
- 必須のファイルタイプと命名規則
- アプリケーションのアイコンの提供方法（アプリケーションと共にアイコンをパッケージ化するかどうかなど）
- アプリケーション記述ファイルの icon エレメントにアイコンを指定するかどうか
- アプリケーションがアイコンを提供しない場合の動作

詳しくは、デバイス製造元にお問い合わせください。

無視される設定

テレビデバイス上のアプリケーションでは、モバイル、ネイティブウィンドウ、またはデスクトップのオペレーティングシステム機能に適用されるアプリケーション設定が無視されます。次の設定が無視されます。

- allowBrowserInvocation
- aspectRatio
- autoOrients
- customUpdateUI
- fileTypes
- height
- installFolder
- maximizable
- maxSize
- minimizable
- minSize
- programMenuFolder
- renderMode
- resizable
- systemChrome
- title
- transparent
- visible
- width
- x

- y

AIR for TV アプリケーションのパッケージ化

ADT によるパッケージ化

AIR ADT コマンドラインツールを使用して、AIR for TV アプリケーションをパッケージ化できます。AIR SDK バージョン 2.5 以降、ADT はテレビデバイス用のパッケージ化をサポートします。パッケージ化の前に、すべての ActionScript および MXML コードをコンパイルします。また、コード署名証明書が必要です。ADT -certificate コマンドを使用して、証明書を作成できます。

ADT コマンドとオプションについて詳しくは、162 ページの「AIR 開発ツール（ADT）」を参照してください。

AIR パッケージの作成

AIR パッケージを作成するには、ADT package コマンドを使用します。

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

この例では、次のことを前提としています。

- ADT ツールへのパスがコマンドラインシェルのパス定義に従っていること（300 ページの「PATH 環境変数」を参照）。
- 証明書の codesign.p12 が、ADT コマンドを実行するための親ディレクトリ内にあること。

アプリケーションファイルが含まれているディレクトリからコマンドを実行します。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf および icons ディレクトリです。

上述のコマンドを実行するとき、ADT によって、キーストアパスワードの入力を求めるプロンプトが表示されますシェルプログラムの一部には、入力したパスワード文字を表示しないものもあります。この場合は、入力の完了後に Enter キーを押してください。また、この代わりに storepass パラメーターを使用して ADT コマンドにパスワードを含めることもできます。

AIRN パッケージの作成

AIR for TV アプリケーションがネイティブ拡張を使用する場合、AIR パッケージではなく AIRN パッケージを作成します。AIRN パッケージを作成するには、ADT package コマンドを使用し、ターゲットの種類を airn に設定します。

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp-app.xml myApp.swf
icons -extdir C:\extensions
```

この例では、次のことを前提としています。

- ADT ツールへのパスがコマンドラインシェルのパス定義に従っていること（300 ページの「PATH 環境変数」を参照）。
- 証明書の codesign.p12 が、ADT コマンドを実行するための親ディレクトリ内にあること。
- -extdir パラメーターを使用すると、アプリケーションで使用する ANE ファイルを含むディレクトリに名前を付けることができます。

  ANE ファイルには、拡張の ActionScript 専用スタブまたは拡張のシミュレーターのバージョンが含まれます。ネイティブコードを含む拡張のバージョンは、AIR for TV デバイスにインストールされます。

アプリケーションファイルが含まれているディレクトリからコマンドを実行します。この例で使用されているアプリケーションファイルは、myApp-app.xml（アプリケーション記述ファイル）、myApp.swf および icons ディレクトリです。

上述のコマンドを実行するとき、ADT によって、キーストアパスワードの入力を求めるプロンプトが表示されますシェルプログラムの一部には、入力したパスワード文字を表示しないものもあります。この場合は、入力の完了後に Enter キーを押してください。また、この代わりに storepass パラメーターを使用してコマンドにパスワードを含めることもできます。

ネイティブ拡張を使用する AIR for TV アプリケーション向けに AIRI ファイルを作成することもできます。AIRI ファイルは AIRN ファイルとほぼ同様ですが、署名されていない点が異なります。次に、例を示します。

```
adt -prepare myApp.airi myApp.xml myApp.swf icons -extdir C:\extensions
```

この後で、アプリケーションへの署名の準備が整ったときに、AIRI ファイルから AIRN ファイルを作成できます。

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp.airi
```

詳しくは、「Adobe AIR 用ネイティブ拡張の開発」を参照してください。

Flash Builder または Flash Professional によるパッケージ化

Flash Professional および Flash Builder を使用すると、AIR パッケージのパブリッシュまたは書き出しが可能です。このとき、ADT を実行する必要はありません。AIR アプリケーションの AIR パッケージを作成する手順は、これらのプログラムのドキュメントで説明しています。

ただし、現時点では、ネイティブ拡張を使用する AIR for TV 向けのアプリケーションパッケージである AIRN パッケージを作成できるのは ADT だけです。

詳しくは、以下のトピックを参照してください。

- Flash Builder を使用した AIR アプリケーションのパッケージ化
- Flash Professional を使用した Adobe AIR のパブリッシュ

AIR for TV アプリケーションのデバッグ

ADL を使用したデバイスシミュレーション

ほとんどのアプリケーション機能をテストおよびデバッグするために最も迅速かつ簡単に実行できる方法は、AIR Debug Launcher（ADL）ユーティリティを使用して、開発コンピューター上でアプリケーションを実行することです。

ADL では、アプリケーション記述子内の supportedProfiles エレメントを使用して、どのプロファイルを使用するかを選択します。設定値には次のような意味があります。

- 複数のプロファイルがリストされている場合、ADL は、リスト上の最初のプロファイルを使用します。
- ADL の -profile パラメーターを使用して、supportedProfiles リスト内の別のプロファイルを選択することができます。
- アプリケーション記述子に supportedProfiles エレメントを含めない場合、任意のプロファイルを -profile 引数に指定できます。

例えば、tv プロファイルをシミュレートするアプリケーションを起動するには、次のコマンドを使用します。

```
adl -profile tv myApp-app.xml
```

ADL を使用してデスクトップ上で tv プロファイルや extendedTV プロファイルをシミュレートする場合は、アプリケーションはターゲットデバイスにより近い環境で実行されます。次に、例を示します。

- -profile 引数のプロファイルに含まれていない ActionScript API は使用できません。
- ADL では、リモートコントローラーなどのデバイス入力コントローラーからの入力をメニューコマンドから受け入れることができます。
- -profile 引数に tv または extendedTV を指定すると、ADL はデスクトップ上で StageVideo クラスをシミュレートすることができます。
- -profile 引数に extendedTV を指定すると、アプリケーションでは、アプリケーション AIRN ファイルと共にパッケージ化されたネイティブ拡張のスタブやシミュレーターを使用することができます。

ただし、ADL はデスクトップ上でアプリケーションを実行するので、ADL を使用した AIR for TV アプリケーションのテストには次の制限が伴います。

- デバイス上でのアプリケーションのパフォーマンスは反映されません。ターゲットデバイス上でパフォーマンステストを実行してください。
- StageVideo オブジェクトの制限はシミュレートされません。通常、AIR for TV デバイスを対象とする場合は、Video クラスではなく StageVideo クラスを使用してビデオを再生します。StageVideo クラスはデバイスのハードウェアによるパフォーマンス上のメリットを活用できますが、表示の制限事項が伴います。ADL はこれらの制限なしにデスクトップ上でビデオを再生します。このため、ターゲットデバイス上でビデオ再生をテストしてください。
- ネイティブ拡張のネイティブコードはシミュレートされません。ただし、ADL -profile 引数では、ネイティブ拡張をサポートする extendedTV プロファイルを指定できます。ADL を使用すると、ANE パッケージに含まれる拡張の ActionScript 専用スタブまたは拡張のシミュレーターバージョンを利用したテストを実行できます。ただし通常は、対応する拡張（デバイスにインストールされる）にも、ネイティブコードが含まれます。ネイティブコードが含まれる拡張を使用してテストするには、ターゲットデバイス上でアプリケーションを実行します。

詳しくは、157 ページの「AIR Debug Launcher（ADL）」を参照してください。

ネイティブ拡張の使用

アプリケーションでネイティブ拡張を使用する場合、ADL コマンドは次の例のようになります。

```
adl -profile extendedTV -extdir C:\extensionDirs myApp-app.xml
```

この例では、次のことを前提としています。

- ADL ツールへのパスがコマンドラインシェルのパス定義に従っていること（300 ページの「PATH 環境変数」を参照）。
- 現在のディレクトリにアプリケーションファイルが含まれていること。これらのファイルには、SWF ファイルやアプリケーション記述ファイル（例では、myApp-app.xml）などがあります。
- -extdir パラメーターを使用すると、アプリケーションで使用する各ネイティブ拡張用のディレクトリを含むディレクトリに名前を付けることができます。これらの各ディレクトリには、ネイティブ拡張の「パッケージ化されていない」ANE ファイルが含まれます。次に、例を示します。

```
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
```

パッケージ化されていない ANE ファイルには、拡張の ActionScript 専用スタブまたは拡張のシミュレーターのバージョンが含まれます。ネイティブコードを含む拡張のバージョンは、AIR for TV デバイスにインストールされます。

詳しくは、「Adobe AIR 用ネイティブ拡張の開発」を参照してください。

入力制御
ADL はテレビデバイスのリモートコントロールボタンをシミュレートします。テレビプロファイルのいずれかを使用して ADL を起動したときに表示されるメニューによって、これらのボタン入力をシミュレーション用のデバイスに送信できます。

画面のサイズ
ADL -screensize パラメーターを設定することで、様々なサイズの画面上でアプリケーションをテストできます。通常の画面と最大化した画面の幅と高さを表す、4 つの値を含む文字列を指定できます。

縦長レイアウト用のピクセルサイズを必ず指定してください。つまり、幅の値を、高さよりも小さな値に指定してください。次に、例を示します。

```
adl -screensize 728x1024:768x1024 myApp-app.xml
```

トレースステートメント

デスクトップ上でテレビアプリケーションを実行すると、デバッガーまたは ADL の起動に使用したターミナルウィンドウに、トレース出力が表示されます。

Flash Professional によるリモートデバッグ

Flash Professional を使用して、AIR for TV アプリケーションをターゲットデバイスで実行しているときに、このアプリケーションをリモートでデバッグすることができます。ただし、リモートデバッグの設定手順はデバイスによって異なります。例えば、Adobe® AIR® for TV MAX 2010 Hardware Development Kit には、このデバイス用の詳細な手順に関するドキュメントが含まれています。

ターゲットデバイスの種類に関わらず、リモートデバッグを準備するために次の手順を実行します。

1. パブリッシュ設定ダイアログボックスの「Flash」タブで、「デバッグを許可」を選択します。

   このオプションを使用すると、Flash Professional では、FLA ファイルから作成されるすべての SWF ファイル内にデバッグ情報が取り込まれます。

2. Adobe AIR 設定ダイアログボックス（アプリケーションとインストーラーの設定）の「署名」タブで、AIR Intermediate（AIRI）ファイルの準備をするためのオプションを選択します。

   アプリケーションの開発途中では、デジタル署名を必要としない AIRI ファイルを使用すれば十分です。

3. AIRI ファイルを作成しアプリケーションをパブリッシュします。

最後の手順により、アプリケーションがターゲットデバイスにインストールされ実行されます。ただし、これらの手順はデバイスによって異なります。

Flash Builder によるリモートデバッグ

Flash Builder を使用して、AIR for TV アプリケーションをターゲットデバイスで実行しているときに、このアプリケーションをリモートでデバッグすることもできます。ただし、リモートデバッグの手順はデバイスによって異なります。

ターゲットデバイスの種類に関わらず、リモートデバッグを準備するために次の手順を実行します。

1. プロジェクト／リリースビルドの書き出しを選択します。AIR Intermediate（AIRI）ファイルを準備するオプションを選択します。

   アプリケーションの開発途中では、デジタル署名を必要としない AIRI ファイルを使用すれば十分です。

2. AIRI ファイルを作成しアプリケーションをパブリッシュします。

3. アプリケーションの AIRI パッケージを、デバッグ情報を持つ SWF ファイルを含めるように変更します。

デバッグ情報を持つ SWF ファイルは、アプリケーション用の Flash Builder プロジェクトディレクトリの下の bin-debug というディレクトリにあります。AIRI パッケージ内の SWF ファイルを、bin-debug ディレクトリ内の SWF ファイルに置き換えます。

Windows 開発マシンでは、次の操作によってこの置き換えを実行できます。

1. AIRI パッケージファイルの拡張子を、.airi から .zip に変更します。
2. ZIP ファイルの中身を抽出します。
3. 抽出されたディレクトリ構造内の SWF ファイルを bin-debug の SWF に置き換えます。
4. 抽出されたディレクトリ内のファイルを再度圧縮します。
5. 圧縮したファイルの拡張子をもう一度 .airi に変更します。

Mac 開発マシンを使用している場合は、この置き換えの手順はデバイスによって異なります。ただし、一般的には次の手順を行います。

1. ターゲットデバイスに AIRI パッケージをインストールします。
2. ターゲットデバイス上にあるアプリケーションインストールディレクトリ内の SWF ファイルを、bin-debug ディレクトリの SWF ファイルに置き換えます。

   例えば、Adobe AIR for TV MAX 2010 Hardware Development Kit に含まれるデバイスの場合、キットに含まれるドキュメントの説明に従って、AIRI パッケージをインストールします。その後で、Mac 開発マシンのコマンドラインで telnet を使用して、ターゲットデバイスにアクセスします。アプリケーションインストールディレクトリ /opt/adobe/stagecraft/apps/**<application name>**/ にある SWF ファイルを、bin-debug ディレクトリの SWF ファイルに置き換えます。

次に、Flash Builder と Adobe AIR for TV MAX 2010 Hardware Development Kit に含まれる デバイスを使用したリモートデバッグの手順を示します。

1. Flash Builder を実行中のコンピューター（開発コンピューター）で、MAX 2010 Hardware Development Kit に付属する AIR for TV Device Connector を実行します。開発コンピューターの IP アドレスが表示されます。
2. ハードウェアキットデバイス上で DevMaster アプリケーション（開発キットに含まれる）を起動します。
3. DevMaster アプリケーションで、AIR for TV Device Connector に表示されている開発コンピューターの IP アドレスを入力します。
4. DevMaster アプリケーションで、「Enable Remote Debugging」が選択されていることを確認します。
5. DevMaster アプリケーションを終了します。
6. 開発コンピューターで、AIR for TV Connector の「Start」を選択します。
7. ハードウェアキットデバイスで、他のアプリケーションを開始します。AIR for TV Device Connector にトレース情報が表示されているかどうかを確認します。

   トレース情報が表示されていない場合、開発コンピューターとハードウェアキットデバイスが接続されていません。トレース情報に使用される開発コンピューター上のポートが利用可能であることを確認してください。AIR for TV Device Connector で異なるポートを選択できます。また、ファイアウォールが選択したポートへのアクセスを許可していることを確認してください。

次に、Flash Builder でデバッガーを開始します。次の手順を実行してください。

1. Flash Builder で、実行／デバッグの構成を選択します。
2. ローカルデバッグ用の既存のデバッグの構成から、プロジェクトの名前をコピーします。
3. デバッグの構成ダイアログボックスで、「Web アプリケーション」を選択します。次に、「新規の起動構成」アイコンを選択します。
4. 「プロジェクト」フィールドにプロジェクト名をペーストします。

5 「起動する URL またはパス」セクションで、「デフォルトを使用」のチェックを解除します。また、テキストフィールドに "**about:blank**" を入力します。

6 「適用」を選択して、変更を保存します。

7 「デバッグ」を選択し、Flash Builder デバッガーを開始します。

8 ハードウェアキットデバイス上でアプリケーションを開始します。

この時点で、Flash Builder デバッガーを使用して、ブレークポイントの設定や変数の調査などを実行できます。

第 9 章：Adobe AIR 用ネイティブ拡張の使用

Adobe AIR 用のネイティブ拡張では ActionScript API が提供され、ネイティブコードでプログラムされているデバイス固有の機能にアクセスできます。ネイティブ拡張の開発者は、デバイス製造元と共同作業することもあれば、サードパーティの開発者である場合もあります。

ネイティブ拡張を開発する場合は、『Adobe AIR 用ネイティブ拡張の開発』を参照してください。

ネイティブ拡張とは、次の 2 つを組み合わせたものです。

- ActionScript クラス
- ネイティブコード

ただし、ネイティブ拡張を使用している AIR アプリケーション開発者は、ActionScript クラスのみを利用して作業します。

ネイティブ拡張は、次のような状況で役立ちます。

- ネイティブコード実装を使用すると、プラットフォーム固有の機能にアクセスできます。これらのプラットフォーム固有の機能は、ビルトイン ActionScript クラスでは利用できません。また、アプリケーション固有の ActionScript クラスで実装することもできません。ネイティブコード実装では、デバイス固有のハードウェアおよびソフトウェアにアクセスできるので、このような機能を提供できます。
- ネイティブコード実装は ActionScript のみを使用する実装よりも速い場合があります。
- ネイティブコード実装を使用すると、ActionScript から従来のネイティブコードにアクセスできます。

ネイティブ拡張に関するいくつかの例が、アドビデベロッパーセンターに掲載されています。その 1 つとして、あるネイティブ拡張を使用することで、AIR アプリケーションから Android のバイブレーション機能にアクセスする例があります。「Native extensions for Adobe AIR」を参照してください。

AIR ネイティブ拡張（ANE）ファイル

ネイティブ拡張の開発者は、ネイティブ拡張を ANE ファイルにパッケージ化します。ANE ファイルは、ネイティブ拡張に必要なライブラリやリソースを含むアーカイブファイルです。

デバイスによっては、ネイティブ拡張で使用するネイティブコードライブラリが ANE ファイルに含まれている場合もあれば、ネイティブコードライブラリがデバイスにインストールされている場合もあります。また、特定のデバイスではネイティブコードがネイティブ拡張に全く含まれず、ActionScript のみで実装される場合もあります。

AIR アプリケーション開発者は、ANE ファイルを次のように使用します。

- SWC ファイルをライブラリパスに含める場合と同じ方法で、ANE ファイルをアプリケーションのライブラリパスに含めます。この操作を行うことで、アプリケーションは拡張の ActionScript クラスを参照できるようになります。

  注意：アプリケーションをコンパイルするときは、必ず ANE に動的リンクを使用してください。Flash Builder を使用する場合は、ActionScript Builder パスのプロパティパネルで「外部」を指定します。コマンドラインを使用する場合は、-external-library-path と指定します。
- ANE ファイルを AIR アプリケーションと共にパッケージ化します。

ネイティブ拡張と NativeProcess ActionScript クラスの比較

ActionScript 3.0 では、NativeProcess クラスが提供されます。このクラスを使用すると、AIR アプリケーションで、ホストオペレーティングシステム上でネイティブプロセスを実行できます。この機能はネイティブ拡張と類似しており、プラットフォーム固有の機能およびライブラリにアクセスできます。NativeProcess クラスを使用するかネイティブ拡張を使用するかを判断する際には、次の点を考慮してください。

- NativeProcess クラスをサポートするのは extendedDesktop AIR プロファイルのみです。したがって、AIR プロファイル mobileDevice および extendedMobileDevice を使用しているアプリケーションの場合は、ネイティブ拡張が唯一の選択肢です。
- ネイティブ拡張の開発者は、多くの場合、様々なプラットフォーム用のネイティブ実装を提供しますが、ネイティブ拡張の開発に伴って提供される ActionScript API は、通常、どのプラットフォームでも同じになります。NativeProcess クラスを使用する場合、ネイティブプロセスを開始するための ActionScript コードは、プラットフォームによって異なることがあります。
- NativeProcess クラスは別プロセスを開始します。一方、ネイティブ拡張は AIR アプリケーションと同じプロセスで実行されます。そのため、コードの失敗が懸念される場合は、NativeProcess クラスを使用する方が安全です。ただし、別プロセスになると、通常はプロセス間通信処理を実装することになります。

ネイティブ拡張と ActionScript クラスライブラリ（SWC ファイル）の比較

SWC ファイルは、アーカイブ形式の ActionScript クラスライブラリです。SWC ファイルには、SWF ファイルおよびその他のリソースファイルが含まれています。SWC ファイルは、ActionScript コードとリソースファイルを個々に共有するのではなく、ActionScript クラスを共有する場合に便利です。

ネイティブ拡張パッケージは ANE ファイルです。SWC ファイルと同様に、ANE ファイルも ActionScript クラスライブラリであり、SWF ファイルとその他のリソースファイルがアーカイブ形式で含まれています。ただし、ANE ファイルと SWC ファイルの最も重要な相違点は、ネイティブコードライブラリは ANE ファイルのみに含めることができるということです。

注意：アプリケーションをコンパイルするときは、必ず ANE ファイルに動的リンクを使用してください。Flash Builder を使用する場合は、ActionScript Builder パスのプロパティパネルで「外部」を指定します。コマンドラインを使用する場合は、-external-library-path と指定します。

関連項目

About SWC files

サポートされているデバイス

AIR 3 以降では、次のデバイス向けのネイティブ拡張を使用できます。

- Android デバイス（Android 2.2 以降）
- iOS デバイス（iOS 4.0 以降）
- iOS シミュレーター（AIR 3.3 以降）

- Blackberry PlayBook
- AIR 3.0 をサポートする Windows デスクトップデバイス
- AIR 3.0 をサポートする Mac OS X デスクトップデバイス

多くの場合、同じネイティブ拡張が複数のプラットフォームをターゲットにできます。拡張の ANE ファイルには、サポートされている各プラットフォーム用の ActionScript ライブラリとネイティブライブラリが含まれます。通常、ActionScript ライブラリのパブリックインターフェイスは、すべてのプラットフォームで同じです。ネイティブライブラリでは、プラットフォームごとに異なります。

ネイティブ拡張がデフォルトのプラットフォームをサポートする場合もあります。デフォルトプラットフォームの実装には、ActionScript コードのみが含まれ、ネイティブコードは含まれません。拡張で明確にサポートされていないプラットフォーム向けにアプリケーションをパッケージ化する場合、アプリケーションでは実行時にデフォルトの実装が使用されます。例えば、モバイルデバイスのみに適用される機能を備えた拡張の場合、この拡張でデフォルトの実装も提供することにより、デスクトップアプリケーションでもこの機能をシミュレーションすることができます。

サポートされているデバイスプロファイル

ネイティブ拡張は以下の AIR プロファイルでサポートされています。

- extendedDesktop（AIR 3.0 以降）
- mobileDevice（AIR 3.0 以降）
- extendedMobileDevice（AIR 3.0 以降）

関連項目

AIR プロファイルのサポート

ネイティブ拡張を使用するためのタスクリスト

アプリケーションでネイティブ拡張を使用するには、次のタスクを実行します。

1. アプリケーション記述ファイルで拡張を宣言します。
2. アプリケーションのライブラリパスに ANE ファイルを含めます。
3. アプリケーションをパッケージ化します。

アプリケーション記述ファイルでの拡張の宣言

すべての AIR アプリケーションに、アプリケーション記述ファイルがあります。ネイティブ拡張を使用するアプリケーションのアプリケーション記述ファイルには、<extensions> エレメントが含まれます。次に、例を示します。

```
<extensions>
    <extensionID>com.example.Extension1</extensionID>
    <extensionID>com.example.Extension2</extensionID>
</extensions>
```

extensionID エレメントは、拡張記述ファイルの id エレメントと同じ値です。拡張記述ファイルは、extension.xml という名前の XML ファイルです。これは ANE ファイルにパッケージ化されます。アーカイブ抽出ツールを使用して extension.xml ファイルを確認することができます。

アプリケーションのライブラリパスに ANE ファイルを含める

ネイティブ拡張を使用するアプリケーションをコンパイルするには、ANE ファイルをライブラリパスに含めます。

Flash Builder での ANE ファイルの使用

アプリケーションがネイティブ拡張を使用している場合は、ライブラリパスにネイティブ拡張用の ANE ファイルを含めます。その後で、Flash Builder を使用して、ActionScript コードをコンパイルできます。

Flash Builder 4.5.1 を使用して、次の手順を実行します。

1. ANE ファイルの拡張子を .ane から .swc に変更します。Flash Builder でファイルを検索するためには、この手順は必須です。
2. Flash Builder プロジェクトでプロジェクト／プロパティを選択します。
3. プロパティダイアログボックスの「Flex ビルドパス」を選択します。
4. 「ライブラリパス」タブで、「SWC の追加 ...」を選択します。
5. SWC ファイルを参照し、「開く」を選択します。
6. 「SWC の追加 ...」ダイアログボックスで、「OK」を選択します。

   ANE ファイルが、プロパティダイアログボックスの「ライブラリパス」タブに表示されます。
7. SWC ファイルエントリを展開します。「リンクの種類」をダブルクリックし、ライブラリパスアイテムオプションダイアログボックスを開きます。
8. ライブラリパスアイテムオプションダイアログボックスで、「リンクの種類」を「外部」に変更します。

これで、プロジェクト／プロジェクトのビルドなどを使用して、アプリケーションをコンパイルできます。

Flash Professional での ANE ファイルの使用

アプリケーションがネイティブ拡張を使用している場合は、ライブラリパスにネイティブ拡張用の ANE ファイルを含めます。その後で、Flash Professional CS5.5 を使用して、ActionScript コードをコンパイルできます。次の手順を実行してください。

1. ANE ファイルの拡張子を .ane から .swc に変更します。Flash Professional でファイルを検索するためには、この手順は必須です。
2. FLA ファイルに対して、ファイル／ ActionScript 設定を選択します。
3. ActionScript 3.0 の詳細設定ダイアログボックスの「ライブラリパス」タブを選択します。
4. 「SWC ファイルを参照」ボタンを選択します。
5. SWC ファイルを参照し、「開く」を選択します。

   SWC ファイルが、ActionScript 3.0 の詳細設定ダイアログボックスの「ライブラリパス」タブに表示されます。
6. 選択した SWC ファイルを使用して、「ライブラリのリンケージオプションを選択」ボタンを選択します。
7. ライブラリパスアイテムオプションダイアログボックスで、「リンクの種類」を「外部」に変更します。

ネイティブ拡張を使用するアプリケーションのパッケージ化

ネイティブ拡張を使用するアプリケーションをパッケージ化するには、ADT を使用します。Flash Professional CS5.5 や Flash Builder 4.5.1 を使用してアプリケーションをパッケージ化することはできません。

ADT の使用方法について詳しくは、「AIR 開発ツール（ADT）」を参照してください。

例えば、以下の ADT コマンドでは、ネイティブ拡張を使用するアプリケーションの DMG ファイル（Mac OS X 用のネイティブインストーラーファイル）が作成されます。

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

以下のコマンドでは、Android デバイス用の APK パッケージが作成されます。

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

以下のコマンドでは、iPhone アプリケーション用の iOS パッケージが作成されます。

```
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
```

次の点に注意してください。

- ネイティブインストーラーパッケージタイプを使用します。
- 拡張ディレクトリを指定します。
- ANE ファイルがアプリケーションのターゲットデバイスをサポートすることを確認します。

ネイティブインストーラーパッケージタイプの使用

アプリケーションパッケージは、ネイティブインストーラーであることが必要です。通常、ネイティブ拡張にはネイティブコードが含まれているので、ネイティブ拡張を使用するアプリケーション向けにクロスプラットフォームの AIR パッケージ（.air パッケージ）を作成することはできません。ただし、通常、ネイティブ拡張は同じ ActionScript API を備えた複数のネイティブプラットフォームをサポートします。その場合は、複数の異なるネイティブインストーラーパッケージで同じ ANE ファイルを使用できます。

以下の表は、ADT コマンドの -target オプションに使用する値をまとめたものです。

アプリケーションのターゲットプラットフォーム	-target
Mac OS X または Windows デスクトップデバイス	-target native -target bundle
Android	-target apk またはその他の Android パッケージターゲット
iOS	-target ipa-ad-hoc またはその他の iOS パッケージターゲット
iOS シミュレーター	-target ipa-test-interpreter-simulator -target ipa-debug-interpreter-simulator

拡張ディレクトリの指定

ADT オプション -extdir を使用して、ネイティブ拡張（ANE ファイル）を含むディレクトリを ADT に通知します。

このオプションについて詳しくは、178 ページの「ファイルとパスのオプション」を参照してください。

ANE ファイルがアプリケーションのターゲットデバイスをサポートすることの確認

ANE ファイルが提供されるときに、ネイティブ拡張の開発者から、拡張でサポートされるプラットフォームが伝えられます。アーカイブ抽出ツールを使用して、ANE ファイルの内容を調べることもできます。抽出されたファイルには、サポートされているプラットフォームごとのディレクトリが含まれます。

拡張でサポートされているプラットフォームを把握しておくことは、ANE ファイルを使用するアプリケーションをパッケージ化する際に重要です。次の規則を考慮してください。

- Android アプリケーションパッケージを作成するには、ANE ファイルに Android-ARM プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- iOS アプリケーションパッケージを作成するには、ANE ファイルに iPhone-ARM プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- iOS シミュレーターアプリケーションパッケージを作成するには、ANE ファイルに iPhone-x86 プラットフォームを含める必要があります。
- Mac OS X アプリケーションパッケージを作成するには、ANE ファイルに MacOS-x86 プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- Windows アプリケーションパッケージを作成するには、ANE ファイルに Windows-x86 プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。

第 10 章：ActionScript コンパイラー

ActionScript および MXML コードは、AIR アプリケーションに含める前にコンパイルする必要があります。Adobe Flash Builder または Adobe Flash Professional などの統合開発環境（IDE）を使用する場合、IDE がバックグラウンドでコンパイルを処理します。ただし、IDE を使用しない場合またはビルドスクリプトを使用する場合は、コマンドラインから ActionScript コンパイラーを呼び出して、SWF ファイルを作成することもできます。

Flex SDK の AIR コマンドラインツール

Adobe AIR アプリケーションの作成に使用するコマンドラインツールは、それぞれに対応するアプリケーション構築用のツールを呼び出します。

- amxmlc は mxmlc を呼び出してアプリケーションクラスをコンパイルします。
- acompc は compc を呼び出してライブラリとコンポーネントのクラスをコンパイルします。
- aasdoc は asdoc を呼び出してソースコードのコメントからドキュメントファイルを生成します。

ユーティリティの Flex バージョンと AIR バージョンの唯一の違いは、AIR バージョンが air-config.xml ファイルから（flex-config.xml ファイルからではなく）設定オプションを読み込むという点です。

Flex SDK ツールおよびそれらのコマンドラインオプションについて詳しくは、Flex に関するドキュメントを参照してください。ここでは、作業の開始を支援し、Flex アプリケーションの構築と AIR アプリケーションの構築の違いを示すことを目的として、Flex SDK ツールの基本レベルの情報を記載しています。

関連項目

37 ページの「Flex SDK を使用した初めてのデスクトップ AIR アプリケーションの作成」

コンパイラー設定

通常、コンパイルオプションの指定には、コマンドラインと共に 1 つ以上の設定ファイルを使用します。グローバル Flex SDK 設定ファイルには、コンパイラーの実行時に使用されるデフォルト値が含まれます。このファイルは各自の開発環境に合わせて編集できます。Flex SDK のインストール先の frameworks ディレクトリに、2 つのグローバル Flex 設定ファイルがあります。air-config.xml ファイルは、amxmlc コンパイラーの実行時に使用されます。AIR ライブラリを組み込むことによってコンパイラーを AIR 用に設定するファイルです。flex-config.xml ファイルは、mxmlc の実行時に使用されます。

デフォルト設定値は Flex と AIR の機能を確認するには適していますが、本格的なプロジェクトに着手する際は、使用できるオプションを詳しく検討してください。コンパイラーオプションについては、プロジェクト固有の値をローカル設定ファイルで指定することができます。ローカル設定ファイルの値は、所定のプロジェクトのグローバル値より優先されます。

注意：AIR アプリケーション専用のコンパイルオプションはありませんが、AIR アプリケーションをコンパイルする際は AIR ライブラリを参照する必要があります。通常、これらのライブラリは、プロジェクトレベルの設定ファイルまたは Ant などのビルドツールのファイルで参照するか、コマンドラインで直接参照します。

AIR 用 MXML および ActionScript ソースファイルのコンパイル

AIR アプリケーション用 Adobe® ActionScript® 3.0 および MXML アセットは、コマンドライン MXML コンパイラー（amxmlc）を使用してコンパイルできます（HTML ベースのアプリケーションはコンパイル不要です。SWF を Flash Professional でコンパイルする場合は、ムービーを SWF ファイルにパブリッシュするだけでコンパイルが実行されます）。

amxmlc を使用するコマンドラインの基本的な形式は次のとおりです。

```
amxmlc [compiler options] -- MyAIRApp.mxml
```

この **[compiler options]** には、AIR アプリケーションのコンパイルに使用するコマンドラインオプションを指定します。

amxmlc コマンドは、追加パラメーターとして +configname=air を指定して標準の Flex mxmlc コンパイラーを呼び出します。このパラメーターは、flex-config.xml ファイルではなく air-config.xml ファイルを使用するようコンパイラーに指示します。amxmlc の使用は、それ以外の点では mxmlc を使用することと変わりません。

コンパイラーで、AIR アプリケーションのコンパイルに通常必要とされる AIR ライブラリおよび Flex ライブラリを指定する、air-config.xml 設定ファイルが読み込まれます。ローカルのプロジェクトレベルの設定ファイルを使用して、グローバル設定をオーバーライドしたり、グローバル設定に追加オプションを追加したりすることもできます。通常、ローカル設定ファイルを作成するには、グローバルバージョンのコピーを編集するのが最も簡単です。ローカルファイルは、-load-config オプションを使用して読み込むことができます。

-load-config=project-config.xml グローバルオプションをオーバーライドします。

-load-config+=project-config.xml -library-path オプションなど、複数の値を指定できるグローバルオプションに追加の値を追加します。1 つの値しか指定できないグローバルオプションはオーバーライドされます。

ローカル設定ファイルに特別の名前付け規則を使用すると、amxmlc コンパイラーでローカルファイルが自動的に読み込まれます。例えば、メイン MXML ファイルが RunningMan.mxml である場合は、ローカル設定ファイルの名前を RunningMan-config.xml にします。今度は、アプリケーションをコンパイルするには、次のように入力するだけです。

```
amxmlc RunningMan.mxml
```

RunningMan-config.xml は、コンパイルする MXML ファイルとファイル名が一致するので自動的に読み込まれます。

amxmlc の例

以下の例では、amxmlc コンパイラーの使用を示します（アプリケーションの ActionScript および MXML のアセットのみ、コンパイルする必要があります）。

AIR MXML ファイルをコンパイルします。

```
amxmlc myApp.mxml
```

コンパイルして出力名を設定します。

```
amxmlc -output anApp.swf -- myApp.mxml
```

AIR ActionScript ファイルをコンパイルします。

```
amxmlc myApp.as
```

コンパイラーの設定ファイルを指定します。

```
amxmlc -load-config config.xml -- myApp.mxml
```

別の設定ファイルから追加オプションを追加します。

```
amxmlc -load-config+=moreConfig.xml -- myApp.mxml
```

コマンドラインでライブラリを追加します（設定ファイルに既に指定されているライブラリに加えます）。

```
amxmlc -library-path+=/libs/libOne.swc,/libs/libTwo.swc  -- myApp.mxml
```

設定ファイルを使用せずに AIR MXML ファイルをコンパイルします（Windows）。

```
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
```

設定ファイルを使用せずに AIR MXML ファイルをコンパイルします（Mac OS X または 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
```

ランタイム共有ライブラリを使用する AIR MXML ファイルをコンパイルします。

```
amxmlc -external-library-path+=../lib/myLib.swc -runtime-shared-libraries=myrsl.swf -- myApp.mxml
```

ANE を使用する AIR MXML ファイルをコンパイルします（ANE には必ず -external-library-path を使用してください）。

```
amxmlc -external-library-path+=../lib/myANE.ane -output=myAneApp.swf -- myAneApp.mxml
```

Java からのコンパイルです（クラスパスを設定して mxmlc.jar を含めます）。

```
java flex2.tools.Compiler +flexlib [Flex SDK 3]/frameworks +configname=air [additional compiler options] -
- myApp.mxml
```

flexlib オプションで Flex SDK フレームワークディレクトリの場所を指定して、コンパイラーが flex_config.xml ファイルを見つけられるようにします。

Java からのコンパイルです（クラスパスは設定しません）。

```
java -jar [Flex SDK 2]/lib/mxmlc.jar +flexlib [Flex SDK 3]/frameworks +configname=air [additional compiler
options] -- myApp.mxml
```

Apache Ant を使用してコンパイラーを起動するには（mxmlc.jar を実行する Java タスクの例）：

```
<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>
```

AIR コンポーネントまたはコードライブラリのコンパイル（Flex）

コンポーネントコンパイラーの acompc を使用すると、AIR ライブラリおよび独立したコンポーネントをコンパイルできます。acompc コンポーネントコンパイラーは amxmlc コンパイラーと同じように動作しますが、次の例外があります。

- ライブラリまたはコンポーネントに含めるコードベース内のクラスを指定する必要があります。
- acompc は、ローカル設定ファイルを自動的に検索しません。プロジェクト設定ファイルを使用するには、–load-config オプションを使用する必要があります。

acompc コマンドは、標準の Flex compc コンポーネントコンパイラーを呼び出しますが、flex-config.xml ファイルではなく air-config.xml ファイルから設定オプションを読み込みます。

コンポーネントコンパイラーの設定ファイル

ローカル設定ファイルを使用すると、コマンドラインでソースパスおよびクラス名を入力（場合によっては誤入力）しないで済みます。ローカル設定ファイルを読み込むには、-load-config オプションを acompc コマンドラインに追加します。

次の例では、com.adobe.samples.particles パッケージの ParticleManager と Particle という 2 つのクラスを使用してライブラリを作成するための設定を示します。これらのクラスは、source/com/adobe/samples/particles フォルダーにあります。

```
<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>
```

ParticleLib-config.xml という設定ファイルを使用してこのライブラリをコンパイルするには、次のように入力します。

```
acompc -load-config ParticleLib-config.xml -output ParticleLib.swc
```

コマンドラインでまったく同じコマンドを実行するには、次のように入力します。

```
acompc -source-path source -include-classes com.adobe.samples.particles.Particle
com.adobe.samples.particles.ParticleManager -output ParticleLib.swc
```

1 つの行にコマンド全体を入力するか、コマンドシェルの行継続文字を使用してください。

acompc の例

これらの例では、myLib-config.xml という設定ファイルを使用しているものとします。

AIR コンポーネントまたはライブラリをコンパイルします。

```
acompc -load-config myLib-config.xml -output lib/myLib.swc
```

ランタイム共有ライブラリをコンパイルします。

```
acompc -load-config myLib-config.xml -directory -output lib
```

注意点として、lib フォルダーは存在している必要があり、コマンドを実行する前は空である必要があります。

第 11 章：AIR Debug Launcher（ADL）

開発時、SWF ベースのアプリケーションと HTML ベースのアプリケーションの両方を実行するには、AIR Debug Launcher（ADL）を使用します。ADL を使用すると、アプリケーションをパッケージ化してインストールしなくても実行できます。デフォルトでは、ADL では SDK に付属するランタイムが使用されるので、ADL を使用するためにランタイムを個別にインストールする必要はありません。

ADL では、トレースステートメントとランタイムエラーが標準出力に出力されますが、ブレークポイントやその他のデバッグ機能はサポートされていません。複雑な問題に対するデバッグ作業には、Flash デバッガー（または、Flash Builder などの統合開発環境）を使用できます。

注意：trace() ステートメントがコンソール上に表示されない場合は、mm.cfg ファイルで ErrorRep rtingEnable または TraceOutputFileEnable を指定していないことを確認してください。このファイルのプラットフォーム別の場所について詳しくは、Editing the mm.cfg file を参照してください。

AIR は直接デバッグできるため、（Adobe® Flash® Player の場合のように）デバッグバージョンのランタイムを用意する必要はありません。コマンドラインデバッグを実行するには、Flash デバッガーと AIR Debug Launcher（ADL）を使用します。

Flash デバッガーは、Flex SDK ディレクトリにあります。ネイティブバージョン（例えば Windows では fdb.exe）は、bin サブディレクトリにあります。Java バージョンは、lib サブディレクトリにあります。AIR Debug Launcher である adl.exe は、Flex SDK のインストール先の bin ディレクトリにあります（独立した Java バージョンはありません）。

注意：fdb は AIR アプリケーションを Flash Player で起動しようとするため、fdb で直接 AIR アプリケーションを起動することはできません。代わりに、実行中の fdb セッションに AIR アプリケーションを接続します。

ADL の使用

ADL でアプリケーションを実行するには、次の形式を使用します。

```
adl application.xml
```

この **application.xml** は、実行するアプリケーションのアプリケーション記述ファイルです。

ADL の完全なシンタックスを以下に示します。

```
adl [-runtime runtime-directory]
    [-pubid publisher-id]
    [-nodebug]
    [-atlogin]
    [-profile profileName]
    [-screensize value]
    [-extdir extension-directory]
    application.xml
    [root-directory]
    [-- arguments]
```

（角括弧 [ ] で囲まれたアイテムはオプションです）

-runtime runtime-directory：使用するランタイムが含まれているディレクトリを指定します。指定しない場合、ADL プログラムと同じ SDK のランタイムディレクトリが使用されます。SDK フォルダー外へ ADL を移動している場合は、ランタイムディレクトリを指定します。Windows および Linux では、Adobe AIR ディレクトリが含まれているディレクトリを指定します。Mac OS X では、Adobe AIR.framework が含まれているディレクトリを指定します。

-pubid publisher-id：指定された値を AIR アプリケーションの発行者 ID としてこの実行に割り当てます。一時的な発行者 ID を指定すると、ローカル接続による通信など、アプリケーションを一意に識別できるようにするために発行者 ID を使用する AIR アプリケーションの機能をテストできます。AIR 1.5.3 では、アプリケーション記述ファイルに発行者 ID を指定することもできます。その場合は、このパラメーターを使用しないでください。

注意：AIR 1.5.3 では、発行者 ID の算出と AIR アプリケーションへの割り当てが自動的に実行されなくなりました。既存の AIR アプリケーションのアップデートを作成するときに、発行者 ID を指定できます。ただし、新しいアプリケーションに発行者 ID は不要なので、指定しないでください。

-nodebug：デバッグサポートをオフにします。このオプションを使用すると、アプリケーションプロセスが Flash デバッガーに接続できなくなり、処理されていない例外のダイアログが非表示になります。ただし、トレースステートメントは、引き続きコンソールウィンドウに出力されます。デバッグをオフにすると、アプリケーションの実行がやや速くなり、インストールされているアプリケーションの実行モードに動作が近くなります。

-atlogin：ログイン時にアプリケーションの起動をシミュレートします。このフラグを使用すると、アプリケーションがユーザーのログイン時に起動するように設定されている場合にのみ実行するアプリケーションロジックをテストできます。-atlogin を使用する場合、アプリケーションに送出される InvokeEvent オブジェクトの reason プロパティは、アプリケーションがすでに実行されていない限り、standard ではなく login です。

-profile profileName：指定されたプロファイルを使用して、アプリケーションをデバッグします。profileName には、次の値のいずれか 1 つを指定できます。

- desktop
- extendedDesktop
- mobileDevice

アプリケーション記述子に supportedProfiles エレメントがある場合、-profile を使用して指定するプロファイルは、サポートされるリストのメンバーである必要があります。-profile フラグを使用しないと、アプリケーション記述子の最初のプロファイルがアクティブプロファイルとして使用されます。アプリケーション記述子に supportedProfiles エレメントがない場合、-profile フラグを使用しないと、**desktop** プロファイルが使用されます。

詳しくは、234 ページの「supportedProfiles」および 241 ページの「デバイスプロファイル」を参照してください。

-screensize value：デスクトップで mobileDevice プロファイル内のアプリケーションを実行するときに使用するシミュレートされた画面サイズ。縦長レイアウト用の通常の幅と高さおよびフルスクリーンの幅と高さについては、事前定義した画面の種類またはピクセルサイズで画面サイズを指定します。種類によって値を指定するには、次の事前定義された画面の種類のいずれかを指定します。

画面の種類	通常の幅 × 高さ	フルスクリーンの幅 × 高さ
480	720 x 480	720 x 480
720	1280 x 720	1280 x 720
1080	1920 x 1080	1920 x 1080
Droid	480 × 816	480 × 854
FWQVGA	240 × 432	240 × 432
FWVGA	480 × 854	480 × 854
HVGA	320 x 480	320 x 480
iPad	768 x 1004	768 x 1024
iPadRetina	1536 x 2008	1536 x 2048
iPhone	320 × 460	320 x 480

画面の種類	通常の幅 × 高さ	フルスクリーンの幅 × 高さ
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 × 460	320 x 480
iPodRetina	640 x 920	640 x 960
iPod5Retina	640 x 1096	640 x 1136
NexusOne	480 × 762	480 × 800
QVGA	240 × 320	240 × 320
SamsungGalaxyS	480 × 762	480 × 800
SamsungGalaxyTab	600 x 986	600 x 1024
WQVGA	240 × 400	240 × 400
WVGA	480 × 800	480 × 800

画面のピクセルサイズを直接指定するには、次のフォーマットを使用します。

```
widthXheight:fullscreenWidthXfullscreenHeight
```

縦長レイアウト用のピクセルサイズを必ず指定してください。つまり、幅の値を、高さよりも小さな値に指定してください。例えば、NexusOne は次のように指定できます。

```
-screensize 480x762:480x800
```

-extdir extension-directory：ランタイムでネイティブ拡張を検索するディレクトリ。このディレクトリには、アプリケーションで使用するネイティブ拡張ごとにサブディレクトリが 1 つずつ含まれています。これらの各サブディレクトリには、拡張のパッケージ化されていない ANE ファイルが含まれます。次に、例を示します。

```
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
```

-extdir パラメーターを使用する場合は、次のことを考慮してください。

- ADL コマンドでは、指定されている各ディレクトリに .ane というファイル名拡張子を付ける必要があります。ただし、拡張子「.ane」より前のファイル名の部分には、任意の有効なファイル名を指定できます。アプリケーション記述ファイルの extensionID エレメントの値と一致させる必要はありません。
- -extdir パラメーターは複数回指定できます。
- -extdir パラメーターの使用方法は、ADT ツールと ADL ツールで異なります。ADT では、このパラメーターは、ANE ファイルを含むディレクトリを指定します。
- 環境変数 AIR_EXTENSION_PATH を使用して拡張のディレクトリを指定することもできます。184 ページの「ADT 環境変数」を参照してください。

application.xml：アプリケーション記述ファイルです。200 ページの「AIR アプリケーション記述ファイル」を参照してください。アプリケーション記述子は ADL の唯一の必須パラメーターです。多くの場合、他のパラメーターは必要ありません。

root-directory：実行するアプリケーションのルートディレクトリを指定します。指定しない場合、アプリケーション記述ファイルを含んでいるディレクトリが使用されます。

-- arguments：「--」の後に続くすべての文字ストリングが、コマンドライン引数としてアプリケーションに渡されます。

注意：既に実行されている AIR アプリケーションを起動した場合、そのアプリケーションの新しいインスタンスは開始されません。代わりに、invoke イベントが実行中のインスタンスに送出されます。

ADL の例

現在のディレクトリのアプリケーションを実行します。

```
adl myApp-app.xml
```

現在のディレクトリのサブディレクトリのアプリケーションを実行します。

```
adl source/myApp-app.xml release
```

アプリケーションを実行し、「tick」と「tock」の 2 つのコマンドライン引数を渡します。

```
adl myApp-app.xml -- tick tock
```

特定のランタイムを使用してアプリケーションを実行します。

```
adl -runtime /AIRSDK/runtime myApp-app.xml
```

デバッグサポートを使用せずにアプリケーションを実行します。

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

モバイルデバイスプロファイルのアプリケーションを実行し、Nexus One 画面サイズでシミュレートします。

```
adl -profile mobileDevice -screensize NexusOne myMobileApp-app.xml
```

アプリケーションを実行するために Apache Ant を使用するアプリケーションを実行します（次のパスは Windows の例です）。

```
<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>
```

ADL の終了コードとエラーコード

次の表で、ADL によって出力される終了コードについて説明します。

終了コード	説明
0	正常起動。ADL は、AIR アプリケーションの終了後に終了します。
1	既に実行されている AIR アプリケーションの正常な呼び出し。ADL は直ちに終了します。
2	使用法エラー。ADL に渡された引数が正しくありません。
3	ランタイムが見つかりません。
4	ランタイムを起動できません。多くの場合、アプリケーションで指定されたバージョンが、ランタイムのバージョンと一致しないために起こります。
5	原因不明のエラーが発生しました。
6	アプリケーション記述ファイルが見つかりません。
7	アプリケーション記述子の内容が無効です。このエラーは、通常、XML の形式が正しくないことを示しています。
8	アプリケーションのメインコンテンツファイル（アプリケーション記述ファイルの <content> エレメントで指定）が見つかりません。
9	アプリケーションのメインコンテンツファイルが、有効な SWF または HTML ファイルではありません。
10	-profile オプションで指定されたプロファイルがアプリケーションでサポートされていません。
11	-screensize 引数は、現在のプロファイルではサポートされていません。

第 12 章：AIR 開発ツール（ADT）

AIR 開発ツール（ADT）は、AIR アプリケーションを開発するための、多目的なコマンドラインツールです。ADT を使用して、次の作業を実行できます。

- AIR アプリケーションを .air インストールファイルとしてパッケージ化する
- AIR アプリケーションをネイティブインストーラー（Windows の .exe インストーラーファイル、iOS の ipa または Android の .apk など）としてパッケージ化する
- ネイティブ拡張を AIR ネイティブ拡張（ANE）ファイルとしてパッケージ化する
- AIR アプリケーションを電子証明書で署名する
- アプリケーションのアップデートで使用される電子署名を変更（移行）する
- コンピューターに接続されているデバイスを確認する
- 自己署名入りデジタルコード署名証明書を作成する
- モバイルデバイスに対して、アプリケーションをリモートでインストール、起動、およびアンインストールする
- モバイルデバイスに対して、AIR ランタイムをリモートでインストールおよびアンインストールする

ADT は、AIR SDK に含まれている Java プログラムです。ADT を使用するためには、Java 1.5 以上が必要です。SDK には、ADT を呼び出すためのスクリプトファイルが含まれています。このスクリプトを使用するためには、Java プログラムの場所が PATH 環境変数内に指定されている必要があります。AIR SDK bin ディレクトリも PATH 環境変数内に指定されている場合は、コマンドラインで適切な引数と共に「adt」と入力すると、ADT を呼び出すことができます（PATH 環境変数の設定方法がわからない場合は、ご使用のオペレーティングシステムのドキュメントを参照してください。この設定をさらに容易にするために、ほとんどのコンピューターシステムにおけるパスの設定手順が、300 ページの「PATH 環境変数」に説明されています）。

ADT を使用するには、少なくとも 2GB のコンピューターのメモリが必要です。メモリが 2GB よりも少ない場合、特に iOS 用アプリケーションをパッケージ化する際に、ADT がメモリ不足となる可能性があります。

Java および AIR SDK bin ディレクトリの両方が PATH 変数に含まれていれば、次の基本シンタックスを使用して、ADT を実行できます。

```
adt -command options
```

注意：Adobe Flash Builder、Adobe Flash Professional など、ほとんどの統合開発環境には、AIR アプリケーションのパッケージ化および署名を自動実行する機能があります。該当する開発環境を既に使用している場合、通常、これらの一般的な作業に ADT を使用する必要はありません。ただし、場合によっては、ご使用の統合開発環境でサポートされていない機能を実行するために、ADT をコマンドラインツールとして使用する必要があります。また、自動ビルドプロセスの一部に、ADT をコマンドラインツールとして使用することもできます。

ADT コマンド

ADT に渡される最初の引数には、次のコマンドのいずれか 1 つを指定します。

- -package：AIR アプリケーションまたは AIR ネイティブ拡張（ANE）をパッケージ化します。
- -prepare：AIR アプリケーションを中間ファイル（AIRI）としてパッケージ化しますが、署名はしません。AIRI ファイルをインストールすることはできません。

- -sign：-prepare コマンドで生成された AIRI パッケージに署名します。-prepare および -sign コマンドを使用すると、パッケージ化と署名を異なるタイミングで実行できます。また、-sign コマンドを使用して、ANE パッケージに署名または再署名することもできます。
- -migrate：署名済みの AIR パッケージに移行署名を適用します。これにより、新規または更新されたコード署名証明書を使用できるようになります。
- -certificate：自己署名入りデジタルコード署名証明書を作成します。
- -checkstore：キーストア内の電子証明書にアクセスできるかどうかを確認します。
- -installApp：デバイスまたはデバイスエミュレーターに AIR アプリケーションをインストールします。
- -launchApp：デバイスまたはデバイスエミュレーター上の AIR アプリケーションを起動します。
- -appVersion：デバイスまたはデバイスエミュレーターに現在インストールされている AIR アプリケーションのバージョンを報告します。
- -uninstallApp：デバイスまたはデバイスエミュレーターから AIR アプリケーションをアンインストールします。
- -installRuntime：デバイスまたはデバイスエミュレーターに AIR ランタイムをインストールします。
- -runtimeVersion：デバイスまたはデバイスエミュレーターに現在インストールされている AIR ランタイムのバージョンを報告します。
- -uninstallRuntime：デバイスまたはデバイスエミュレーターから、現在インストールされている AIR ランタイムをアンインストールします。
- -version：ADT のバージョン番号を報告します。
- -devices：接続されているモバイルデバイスまたはエミュレーターに関するデバイス情報を報告します。
- -help ：コマンドとオプションの一覧を表示します。

多くの ADT コマンドでは、オプションのフラグとパラメーターの関連するセットを共有します。これらのオプションのセットについては、以下で、個別に詳しく説明します。

- 176 ページの「ADT コード署名のオプション」
- 178 ページの「ファイルとパスのオプション」
- 179 ページの「デバッガー接続のオプション」
- 179 ページの「ネイティブ拡張のオプション」

ADT package コマンド

-package コマンドは、メインアプリケーションディレクトリから実行する必要があります。このコマンドでは、次のシンタックスを使用します。

コンポーネントアプリケーションファイルからの AIR パッケージの作成：

```
adt -package
    AIR_SIGNING_OPTIONS
    -target packageType
    -sampler
    -hideAneLibSymbols
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    FILE_OPTIONS
```

コンポーネントアプリケーションファイルからのネイティブパッケージの作成：

```
adt -package
    AIR_SIGNING_OPTIONS
    -target packageType
    DEBUGGER_CONNECTION_OPTIONS
    -airDownloadURL URL
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    -platformsdk path
    FILE_OPTIONS
```

コンポーネントアプリケーションファイルからの、ネイティブ拡張を含むネイティブパッケージの作成：

```
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
```

AIR または AIRI ファイルからのネイティブパッケージの作成：

```
adt -package
    -target packageType
    NATIVE_SIGNING_OPTIONS
    output
    input_package
```

コンポーネントネイティブ拡張ファイルからのネイティブ拡張パッケージの作成：

```
adt -package
    AIR_SIGNING_OPTIONS
    -target ane
    output
    ANE_OPTIONS
```

注意：ANE ファイルへの署名は必須ではありません。したがって、AIR_SIGNING_OPTIONS パラメーターは、この例ではオプションです。

AIR_SIGNING_OPTIONS：この AIR 署名オプションは、AIR インストールファイルへの署名に使用される証明書を特定します。署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。

-migrate このフラグは、移行証明書と、AIR_SIGNING_OPTIONS パラメーターで指定した証明書を使用してアプリケーションに署名することを示します。このフラグは、デスクトップアプリケーションをネイティブインストーラーとしてパッケージ化し、アプリケーションでネイティブ拡張を使用する場合にのみ有効です。それ以外の場合は、エラーが発生します。移行証明書の署名オプションは、**MIGRATION_SIGNING_OPTIONS** パラメーターに指定します。署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。-migrate フラグを使用することで、ネイティブ拡張を使用するデスクトップネイティブインストーラーアプリケーションのアップデートを作成し、元の証明書が期限切れの場合などに、そのアプリケーションのコード署名証明書を変更できます。詳しくは、195 ページの「AIR アプリケーションのアップデートバージョンの署名」を参照してください。

-package コマンドの -migrate フラグは、AIR 3.6 以降で使用できます。

-target：作成するパッケージのタイプです。サポートされているパッケージタイプは、以下のとおりです。

- air：AIR パッケージです。「air」がデフォルト値です。AIR または AIRI ファイルを作成する場合は、-target フラグを指定する必要はありません。

- airn：拡張テレビプロファイル内にあるデバイス用のネイティブアプリケーションパッケージです。
- ane：AIR ネイティブ拡張パッケージです。
- Android パッケージのターゲット：
  - apk：Android パッケージです。このターゲットで生成されるパッケージは、Android デバイスにのみインストールできます。エミュレーターにはインストールできません。
  - apk-captive-runtime：アプリケーションと、AIR ランタイムのキャプティブバージョンの両方を含む Android パッケージです。このターゲットで生成されるパッケージは、Android デバイスにのみインストールできます。エミュレーターにはインストールできません。
  - apk-debug：追加のデバッグ情報を含む Android パッケージです（アプリケーション内の SWF ファイルも、デバッグサポートを使用してコンパイルする必要があります）。
  - apk-emulator：デバッグサポートを使用せずにエミュレーターで使用するための Android パッケージです (エミュレーターおよびデバイス両方でデバッグを許可するには、apk-debug ターゲットを使用します)。
  - apk-profile：アプリケーションパフォーマンスおよびメモリプロファイリングをサポートする Android パッケージです。
- iOS パッケージのターゲット：
  - ipa-ad-hoc：アドホック配布用の iOS パッケージです。
  - ipa-app-store：Apple App Store 配布用の iOS パッケージです。
  - ipa-debug：追加のデバッグ情報を含む iOS パッケージです（アプリケーション内の SWF ファイルも、デバッグサポートを使用してコンパイルする必要があります）。
  - ipa-test：最適化やデバッグ情報を含まずにコンパイルされた iOS パッケージです。
  - ipa-debug-interpreter：機能的にはデバッグパッケージと同等ですが、コンパイル時間が短くなります。ただし、ActionScript バイトコードが解釈され、マシンコードには変換されません。そのため、インタープリターパッケージではコードの実行が遅くなります。
  - ipa-debug-interpreter-simulator：機能的には ipa-debug-interpreter と同等ですが、iOS シミュレーター用にパッケージ化されています。Macintosh 専用です。このオプションを使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションも含める必要があります。
  - ipa-test-interpreter：機能的にはテストパッケージと同等ですが、コンパイル時間が短くなります。ただし、ActionScript バイトコードが解釈され、マシンコードには変換されません。そのため、インタープリターパッケージではコードの実行が遅くなります。
  - ipa-test-interpreter-simulator：機能的には ipa-test-interpreter と同等ですが、iOS シミュレーター用にパッケージ化されています。Macintosh 専用です。このオプションを使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションも含める必要があります。
- native：ネイティブデスクトップインストーラーです。生成されるファイルのタイプは、コマンドが実行されるオペレーティングシステムのネイティブなインストールファイルの形式になります。
  - EXE - Windows
  - DMG - Mac
  - DEB - Ubuntu Linux（AIR 2.6 以前）
  - RPM — Fedora または OpenSuse Linux（AIR 2.6 以前）

  詳しくは、56 ページの「デスクトップネイティブインストーラーのパッケージ化」を参照してください。

-sampler：（iOS のみ、AIR 3.4 以降）iOS アプリケーションでテレメトリーベースの ActionScript サンプラーを有効にします。このフラグを使用すると、Adobe Scout でアプリケーションをプロファイリングできます。Scout では、Flash Platform のどんなコンテンツでもプロファイリングできますが、詳細なテレメトリーを有効にすると、ActionScript 関数のタイミング、DisplayList、Stage3D レンダリングなどを詳しく把握できます。このフラグを使用するとパフォーマンスに少々影響があるため、製品版のアプリケーションには使用しないでください。

-hideAneLibSymbols：（iOS のみ、AIR 3.4 以降）アプリケーション開発者は、複数のソースの複数のネイティブエクステンションを使用できます。また、複数の ANE で同じシンボル名が共有されている場合、ADT では「オブジェクトファイルでシンボルが重複している」という内容のエラーが生成されます。場合によっては、このエラーにより、実行時にクラッシュが発生する可能性もあります。hideAneLibSymbols オプションを使用すると、ANE ライブラリのシンボルをそのライブラリのソースのみに表示するか（yes）グローバルに表示するか（no）を指定できます。

- **yes**：ANE シンボルを非表示にします。これにより、意図しないシンボル競合の問題を解決できます。
- **no**：（デフォルト）ANE シンボルを非表示にしません。これは、AIR 3.4 より前のバージョンの動作です。

-embedBitcode：（iOS のみ、AIR 25 以降）アプリケーション開発者は、embedBitcode オプションを使用して yes または no を指定することにより、ビットコードを iOS アプリケーションに埋め込むかどうかを指定できます。指定しない場合、このスイッチのデフォルト値は no です。

DEBUGGER_CONNECTION_OPTIONS：デバッガー接続オプションは、デバッグパッケージが他のコンピューターで動作しているリモートデバッガーに接続を試みるのか、リモートデバッガーからの接続を監視するのかを指定します。この一連のオプションはモバイルデバッグパッケージ（ターゲットが apk-debug および ipa-debug）でのみサポートされます。これらのオプションの説明は、179 ページの「デバッガー接続のオプション」に記載されています。

-airDownloadURL：AIR ランタイムをダウンロードして Android デバイスにインストールするための、代替 URL を指定します。代替 URL を指定しなかった場合、AIR ランタイムがまだインストールされていないと、AIR アプリケーションはユーザーを Android Market の AIR ランタイムにリダイレクトします。

アプリケーションが代替のマーケットプレイス（Google が運営する Android Market 以外）を介して配布される場合、そのマーケットから AIR ランタイムをダウンロードするための URL の指定が必要になることがあります。代替のマーケットによっては、そのマーケットの外部からのダウンロードをアプリケーションが要求できない場合があります。このオプションは Android パッケージのみでサポートされます。

NATIVE_SIGNING_OPTIONS：このネイティブ署名オプションは、ネイティブパッケージファイルへの署名に使用される証明書を特定します。これらの署名オプションは、AIR ランタイムではなく、ネイティブオペレーティングシステムで使用される署名を適用するために使用します。その他の点では、これらのオプションは AIR_SIGNING_OPTIONS と同じです。これらの署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。

ネイティブ署名は、Windows および Android でサポートされています。Windows では、AIR 署名オプションとネイティブ署名オプションの両方を指定する必要があります。Android では、ネイティブ署名オプションのみを指定できます。

多くの場合、同じコード署名証明書を使用して、AIR およびネイティブ署名の両方を適用することができます。ただし、このことはすべての場合について該当するわけではありません。例えば、Android Market に送信されたアプリケーションに関する Google のポリシーでは、すべてのアプリケーションは、少なくとも 2033 年まで有効な証明書で署名されている必要があると示されています。これは、AIR 署名を適用する際に推奨される既知の証明機関によって発行された証明書を、Android アプリケーションへの署名に使用してはいけないことを意味しています（25 年間といった長期の有効期間を持つコード署名証明書を発行する証明機関はありません）。

output：作成するパッケージファイルの名前です。ファイル拡張子は、必要に応じて指定できます。指定しない場合、-target 値と現在のオペレーティングシステムに適した拡張子が付加されます。

app_descriptor：アプリケーション記述ファイルのパスです。このパスには、現在のディレクトリの相対パスか、絶対パスを指定できます（アプリケーション記述ファイルの名前は、AIR ファイルの「**application.xml**」に変更されます）。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。

- Android：AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。
- iOS：AIR SDK はキャプティブ iOS SDK に付属しています。-platformsdk オプションを使用すると、アプリケーションを外部 SDK と共にパッケージ化できるので、キャプティブ iOS SDK 以外の SDK も使用できるようになります。例えば、最新の iOS SDK を使用して拡張を構築した場合は、アプリケーションをパッケージ化する際にその SDK を指定できます。また、iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

-arch：アプリケーション開発者は、この引数を使用して x86 プラットフォーム用の APK を作成できます。使用可能な値は次のとおりです。

- armv7：Android armv7 プラットフォーム用の ADT パッケージ APK です。
- x86：Android x86 プラットフォーム用の ADT パッケージ APK です。

値が指定されない場合は、armv7 がデフォルト値になります。

FILE_OPTIONS：パッケージに含めるアプリケーションファイルを特定します。ファイルオプションについては、178 ページの「ファイルとパスのオプション」で詳しく説明しています AIR または AIRI ファイルからネイティブパッケージを作成する場合は、ファイルオプションを指定しないでください。

input_airi：AIRI ファイルからネイティブパッケージを作成する場合に指定します。ターゲットが **air** である場合（またはターゲットが指定されていない場合）、AIR_SIGNING_OPTIONS が必須となります。

input_air：AIR ファイルからネイティブパッケージを作成する場合に指定します。AIR_SIGNING_OPTIONS は指定しないでください。

ANE_OPTIONS：ネイティブ拡張パッケージを作成するためのオプションとファイルを特定します。拡張パッケージのオプションについては、179 ページの「ネイティブ拡張のオプション」で詳しく説明しています。

ADT -package コマンドの例

SWF ベースの AIR アプリケーションの現在のディレクトリ内にある特定のアプリケーションファイルをパッケージ化します。

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf components.swc
```

HTML ベースの AIR アプリケーションの現在のディレクトリ内にある特定のアプリケーションファイルをパッケージ化します。

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html AIRAliases.js image.gif
```

現在の作業ディレクトリ内にあるすべてのファイルとサブディレクトリをパッケージ化します。

```
adt -package -storetype pkcs12 -keystore ../cert.p12 myApp.air myApp.xml .
```

注意：キーストアファイルには、アプリケーションへの署名に使用する秘密キーが格納されます。署名証明書は、AIR パッケージ内に含めないでください。ADT コマンドでワイルドカードを使用した場合は、キーストアファイルは、パッケージ内に含まれないよう別の場所に配置されます。この例では、キーストアファイル cert.p12 は親ディレクトリに存在します。

メインファイルと images サブディレクトリのみパッケージ化します。

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf images
```

HTML ベースのアプリケーションと HTML、scripts、images の各サブディレクトリ内のすべてのファイルをパッケージ化します。

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml index.html AIRALiases.js html scripts
images
```

作業ディレクトリ（release/bin）内にある application.xml ファイルとメイン SWF ファイルをパッケージ化します。

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml -C release/bin myApp.swf
```

ビルドファイルシステムの複数の場所のアセットをパッケージ化します。この例では、パッケージ化する前は、次のフォルダーにアプリケーションアセットが含まれています。

```
/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
```

/devRoot/myApp ディレクトリから次の ADT コマンドを実行します。

```
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
```

パッケージ構造は次のようになります。

```
/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
```

ADT を Java プログラムとしてシンプルな SWF ベースのアプリケーションに対して実行します（クラスパスは設定しません）。

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf
```

ADT を Java プログラムとしてシンプルな HTML ベースのアプリケーションに対して実行します（クラスパスは設定しません）。

```
java –jar {AIRSDK}/lib/ADT.jar –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html
AIRAliases.js
```

ADT を Java プログラムとして実行します（Java クラスパスを設定して、ADT.jar パッケージを含めます）。

```
java -com.adobe.air.ADT –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf
```

Apache Ant で ADT を Java タスクとして実行します（ただし、通常は、ADT コマンドを直接 Ant スクリプト内で使用することをお勧めします）。例に示してあるパスは、Windows 用です。

```
<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>
```

注意：一部のコンピューターシステムでは、ファイルシステムパス内の 2 バイト文字が誤って解釈される場合があります。この場合、ADT の実行に使用する JRE を、UTF-8 文字セットを使用するように設定してみてください。この設定は、Mac および Linux での ADT の実行に使用するスクリプトでは、デフォルトで行われます。Windows の adt.bat ファイルの場合、または Java から直接 ADT を実行する場合は、Java コマンドラインで -Dfile.encoding=UTF-8 オプションを指定します。

ADT prepare コマンド

-prepare コマンドは、署名されていない AIRI パッケージを作成します。AIRI パッケージは、そのままでは使用できません。-sign コマンドを使用して、AIRI ファイルを署名された AIR パッケージに変換するか、package コマンドを使用して、AIRI ファイルをネイティブパッケージに変換します。

-prepare コマンドでは、次のシンタックスを使用します。

```
adt -prepare output app_descriptor FILE_OPTIONS
```

output：作成する AIRI ファイルの名前です。

app_descriptor：アプリケーション記述ファイルのパスです。このパスには、現在のディレクトリの相対パスか、絶対パスを指定できます（アプリケーション記述ファイルの名前は、AIR ファイルの「**application.xml**」に変更されます）。

FILE_OPTIONS：パッケージに含めるアプリケーションファイルを特定します。ファイルオプションについては、178 ページの「ファイルとパスのオプション」で詳しく説明しています。

ADT sign コマンド

-sign コマンドは、AIRI および ANE ファイルに署名します。

-sign コマンドでは、次のシンタックスを使用します。

```
adt -sign AIR_SIGNING_OPTIONS input output
```

AIR_SIGNING_OPTIONS：この AIR 署名オプションは、パッケージファイルへの署名に使用される証明書を特定します。署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。

input：署名する AIRI または ANE ファイルの名前です。

output：作成する署名されたパッケージの名前です。

ANE ファイルが既に署名されている場合、古い署名は破棄されます（AIR ファイルは再署名できません。アプリケーションのアップデート用に新しい署名を使用するには、-migrate コマンドを使用してください）。

ADT migrate コマンド

-migrate コマンドは、AIR ファイルに移行署名を適用します。電子証明書を更新または変更したために、古い証明書で署名されたアプリケーションをアップデートする必要がある場合、移行署名を必ず使用してください。

移行署名を使用して AIR アプリケーションをパッケージ化する方法について詳しくは、195 ページの「AIR アプリケーションのアップデートバージョンの署名」を参照してください。

注意：移行証明書は、証明書の有効期限から 365 日以内に適用する必要があります。この猶予期間が経過すると、アプリケーションのアップデートに対して、移行署名による署名ができなくなります。ユーザーは、最初に、移行署名で署名されたアプリケーションのバージョンにアップデートしてから最新のアップデートをインストールするか、オリジナルのアプリケーションをアンインストールしてから新しい AIR パッケージをインストールすることができます。

移行署名を使用するには、最初に、新規または更新された証明書を使用して、AIR アプリケーションに署名します（-package または -sign コマンドを使用）。次に、古い証明書と -migrate コマンドを使用して、移行署名を適用します。

-migrate コマンドでは、次のシンタックスを使用します。

```
adt -migrate AIR_SIGNING_OPTIONS input output
```

AIR_SIGNING_OPTIONS：この AIR 署名オプションは、AIR アプリケーションの既存バージョンへの署名に使用したオリジナルの証明書を特定します。署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。

input：新しいアプリケーション証明書で既に署名されている AIR ファイルです。

output：新しい証明書と古い証明書の両方からの署名を持つ、最終的なパッケージの名前です。

入力 AIR ファイルと出力 AIR ファイルには、異なるファイル名を使用する必要があります。

注意：ADT migrate コマンドは、ネイティブ拡張を含む AIR デスクトップアプリケーションでは使用できません。これらのアプリケーションは、.air ファイルではなくネイティブインストーラーとしてパッケージ化されているからです。ネイティブ拡張を含む AIR デスクトップアプリケーション向けの証明書を変更するには、163 ページの「ADT package コマンド」を -migrate フラグ付きで使用してアプリケーションをパッケージ化してください。

ADT checkstore コマンド

-checkstore コマンドを使用して、キーストアの有効性を確認できます。このコマンドでは、次のシンタックスを使用します。

```
adt -checkstore SIGNING_OPTIONS
```

SIGNING_OPTIONS：この署名オプションは、有効性を確認するキーストアを特定します。署名オプションについては、176 ページの「ADT コード署名のオプション」で詳しく説明しています。

ADT certificate コマンド

-certificate コマンドを使用して、自己署名入りデジタルコード署名証明書を作成できます。このコマンドでは、次のシンタックスを使用します。

```
adt -certificate -cn name -ou orgUnit -o orgName -c country -validityPeriod years key-type output password
```

-cn：新しい証明書の共通名として割り当てる文字列です。

-ou：証明書を発行する組織単位として割り当てる文字列です（オプション）。

-o：証明書を発行する組織として割り当てる文字列です（オプション）。

-c：2 文字の ISO-3166 国コードです。無効なコードが指定された場合、証明書は生成されません（オプション）。

-validityPeriod：証明書が有効である年数です。指定しない場合、5 年の有効期間が割り当てられます（オプション）。

key_type：証明書に使用するキーのタイプは **2048-RSA** です。

output：生成される証明書ファイルのパスおよびファイル名です。

password：新しい証明書にアクセスするためのパスワードです。このパスワードは、この証明書を使用して AIR ファイルに署名するときに必要になります。

ADT installApp コマンド

-installApp コマンドは、デバイスまたはエミュレーターにアプリケーションをインストールします。

このコマンドを使用して再インストールする前に、既存のアプリケーションをアンインストールする必要があります。

このコマンドでは、次のシンタックスを使用します。

```
adt -installApp -platform platformName -platformsdk path-to-sdk -device deviceID -package fileName
```

-platform：デバイスのプラットフォームの名前です。**ios** または **android** を指定します。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです（オプション）。

- Android：AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。
- iOS：AIR SDK はキャプティブ iOS SDK に付属しています。-platformsdk オプションを使用すると、アプリケーションを外部 SDK と共にパッケージ化できるので、キャプティブ iOS SDK 以外の SDK も使用できるようになります。例えば、最新の iOS SDK を使用して拡張を構築した場合は、アプリケーションをパッケージ化する際にその SDK を指定できます。また、iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

-device：**ios_simulator**、接続されたデバイスのシリアル番号（Android）またはハンドル（iOS）を指定します。iOS では、このパラメーターは必須です。Android では、複数の Android デバイスまたはエミュレーターがコンピューターに接続および実行されている場合にのみこのパラメーターを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」（Android）または「無効なデバイスが指定されました」（iOS）を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

注意：IPA ファイルを直接 iOS デバイスにインストールすることは、AIR 3.4 以降でのみ可能です。その場合、iTunes 10.5.0 以降が必要です。

adt -devices コマンド（AIR 3.4 以降で利用可能）を使用して、接続されているデバイスのハンドルまたはシリアル番号を確認します。iOS では、デバイス UUID ではなくハンドルを使用します。詳しくは、175 ページの「ADT devices コマンド」を参照してください。

また、Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

-package：インストールするパッケージのファイル名です。iOS では、これは IPA ファイルである必要があります。Android では、これは APK パッケージにする必要があります。指定したパッケージが既にインストールされている場合、ADT は「エラーコード 14：デバイスエラー」を返します。

ADT appVersion コマンド

-appVersion コマンドは、デバイスまたはエミュレーターにインストールされているアプリケーションのバージョンを報告します。このコマンドでは、次のシンタックスを使用します。

```
adt -appVersion -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform：デバイスのプラットフォームの名前です。**ios** または **android** を指定します。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。

- Android：AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。
- iOS：AIR SDK はキャプティブ iOS SDK に付属しています。-platformsdk オプションを使用すると、アプリケーションを外部 SDK と共にパッケージ化できるので、キャプティブ iOS SDK 以外の SDK も使用できるようになります。例えば、最新の iOS SDK を使用して拡張を構築した場合は、アプリケーションをパッケージ化する際にその SDK を指定できます。また、iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

-device：**ios_simulator** またはデバイスのシリアル番号を指定します。複数の Android デバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

-appid：インストールされているアプリケーションの AIR アプリケーション ID です。指定した ID のアプリケーションがデバイスにインストールされていない場合、ADT は「終了コード 14：デバイスエラー」を返します。

ADT launchApp コマンド

-launchApp コマンドは、デバイスまたはエミュレーターにインストールされているアプリケーションを実行します。このコマンドでは、次のシンタックスを使用します。

```
adt -launchApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform：デバイスのプラットフォームの名前です。**ios** または **android** を指定します。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。

- Android：AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。
- iOS：AIR SDK はキャプティブ iOS SDK に付属しています。-platformsdk オプションを使用すると、アプリケーションを外部 SDK と共にパッケージ化できるので、キャプティブ iOS SDK 以外の SDK も使用できるようになります。例えば、最新の iOS SDK を使用して拡張を構築した場合は、アプリケーションをパッケージ化する際にその SDK を指定できます。また、iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

-device：**ios_simulator** またはデバイスのシリアル番号を指定します。複数の Android デバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

-appid：インストールされているアプリケーションの AIR アプリケーション ID です。指定した ID のアプリケーションがデバイスにインストールされていない場合、ADT は「終了コード 14：デバイスエラー」を返します。

ADT uninstallApp コマンド

-uninstallApp コマンドは、リモートデバイスまたはエミュレーターにインストールされているアプリケーションを完全に削除します。このコマンドでは、次のシンタックスを使用します。

```
adt -uninstallApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

-platform：デバイスのプラットフォームの名前です。**ios** または **android** を指定します。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。

- Android：AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。
- iOS：AIR SDK はキャプティブ iOS SDK に付属しています。-platformsdk オプションを使用すると、アプリケーションを外部 SDK と共にパッケージ化できるので、キャプティブ iOS SDK 以外の SDK も使用できるようになります。例えば、最新の iOS SDK を使用して拡張を構築した場合は、アプリケーションをパッケージ化する際にその SDK を指定できます。また、iOS シミュレーターで ADT を使用する場合は、iOS シミュレーター SDK のパスを指定する -platformsdk オプションを必ず含める必要があります。

-device：**ios_simulator** またはデバイスのシリアル番号を指定します。複数の Android デバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

-appid：インストールされているアプリケーションの AIR アプリケーション ID です。指定した ID のアプリケーションがデバイスにインストールされていない場合、ADT は「終了コード 14：デバイスエラー」を返します。

ADT installRuntime コマンド

-installRuntime コマンドは、デバイスに AIR ランタイムをインストールします。

このコマンドを使用して再インストールする前に、既存のバージョンの AIR ランタイムをアンインストールする必要があります。

このコマンドでは、次のシンタックスを使用します。

```
adt -installRuntime -platform platformName -platformsdk path_to_sdk -device deviceID -package fileName
```

-platform：デバイスのプラットフォームの名前です。現在、このコマンドは Android プラットフォームでのみサポートされています。名前には、**android** を使用してください。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。現在サポートされているプラットフォーム SDK は Android だけです。AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。

-device：デバイスのシリアル番号です。複数のデバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

-package：インストールするランタイムのファイル名です。Android では、これは APK パッケージにする必要があります。パッケージを指定しないと、AIR SDK の使用可能なランタイムから、デバイスまたはエミュレーターに適切なランタイムが選択されます。ランタイムが既にインストールされている場合、ADT は「エラーコード 14：デバイスエラー」を返します。

ADT runtimeVersion コマンド

-runtimeVersion コマンドは、デバイスまたはエミュレーターにインストールされている AIR ランタイムのバージョンを報告します。このコマンドでは、次のシンタックスを使用します。

```
adt -runtimeVersion -platform platformName -platformsdk path_to_sdk -device deviceID
```

-platform：デバイスのプラットフォームの名前です。現在、このコマンドは Android プラットフォームでのみサポートされています。名前には、**android** を使用してください。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。現在サポートされているプラットフォーム SDK は Android だけです。AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。

-device：デバイスのシリアル番号です。複数のデバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。ランタイムがインストールされていない場合、または指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

ADT uninstallRuntime コマンド

-uninstallRuntime コマンドは、デバイスまたはエミュレーターから AIR ランタイムを完全に削除します。このコマンドでは、次のシンタックスを使用します。

```
adt -uninstallRuntime -platform platformName -platformsdk path_to_sdk -device deviceID
```

-platform：デバイスのプラットフォームの名前です。現在、このコマンドは Android プラットフォームでのみサポートされています。名前には、**android** を使用してください。

-platformsdk：ターゲットデバイスのプラットフォーム SDK へのパスです。現在サポートされているプラットフォーム SDK は Android だけです。AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。また、AIR_ANDROID_SDK_HOME 環境変数が既に設定されている場合は、プラットフォーム SDK パスをコマンドラインで指定する必要はありません（両方とも設定されている場合、コマンドラインで指定したパスが使用されます）。

-device：デバイスのシリアル番号です。複数のデバイスまたはエミュレーターがコンピューターに接続され、実行されている場合にのみ、デバイスを指定する必要があります。指定したデバイスが接続されていない場合、ADT は「終了コード 14：デバイスエラー」を返します。複数のデバイスまたはエミュレーターが接続されているが、デバイスが指定されていない場合、ADT は「終了コード 2：使用方法エラー」を返します。

Android では、Android ADB ツールを次のように使用して、接続されているデバイスおよび実行しているエミュレーターのシリアル番号の一覧を表示します。

```
adb devices
```

ADT devices コマンド

ADT -help コマンドは、現在接続されているモバイルデバイスおよびエミュレーターのデバイス ID を表示します。

```
adt -devices -platform iOS|android
```

-platform：確認するプラットフォームの名前です。android または iOS を指定します。

注意：iOS では、このコマンドには iTunes 10.5.0 以降が必要です。

ADT help コマンド

ADT -help コマンドは、コマンドラインオプションの簡潔な説明を表示します。

```
adt -help
```

ヘルプの出力では、次の記号の規則を使用します。

- <> - 山かっこ内のアイテムは、指定する必要がある情報を表します。
- () - 丸かっこ内のアイテムは、オプションを表します。help コマンドの出力では、グループとして扱われます。
- すべて大文字 - 大文字で示されているアイテムは、個別に説明されるオプションのセットを表します。
- | - 「または」を表します。例えば、(A | B) はアイテム A またはアイテム B を意味します。
- ? - 0 個または 1 個を表します。アイテムの後にある疑問符は、アイテムがオプションであり、使用する場合はインスタンスを 1 つだけ指定できることを表します。
- * - 0 個またはそれ以上を表します。アイテムの後にあるアスタリスクは、アイテムがオプションであり、インスタンスをいくつでも指定できることを表します。
- + - 1 個またはそれ以上を表します。アイテムの後にあるプラス記号は、アイテムが必須であり、複数のインスタンスを指定できることを表します。
- 記号なし - アイテムの後に記号がない場合は、アイテムが必須であり、インスタンスを 1 つだけ指定できることを表します。

ADT オプションセット

ADT コマンドのいくつかは、共通のオプションセットを共有します。

ADT コード署名のオプション

ADT では、Java Cryptography Architecture（JCA）を利用して、AIR アプリケーションへの署名に使用する秘密キーと証明書にアクセスします。署名オプションによって、キーストアおよびそのキーストア内の秘密キーと証明書を特定します。

キーストアには、秘密キーおよび関連付けられた証明書チェーンが含まれている必要があります。署名証明書がコンピューター上の信頼できる証明書にチェーン化されると、証明書の共通名フィールドの内容が発行者名として AIR インストールダイアログに表示されます。

ADT では、証明書が x509v3 標準（RFC3280）に準拠し、コード署名の適切な値を持つ鍵用途拡張が証明書に含まれている必要があります。この証明書内で定義されている制約が優先され、AIR アプリケーションへの署名に一部の証明書が使用できなくなる場合があります。

注意：ADT では、インターネットリソースに接続して証明書の失効リストの確認およびタイムスタンプの取得を行うために、適切であれば、Java ランタイム環境のプロキシ設定を使用します。ADT の使用時にインターネットリソースへの接続の問題が発生し、特定のプロキシ設定がネットワークに必要な場合、JRE プロキシ設定の構成が必要になることがあります。

AIR 署名オプションのシンタックス

署名オプションは次のシンタックスを（1 つのコマンドラインで）使用します。

```
-alias aliasName
-storetype type
-keystore path
-storepass password1
-keypass password2
-providerName className
-tsa url
```

-alias：キーストア内にあるキーのエイリアスです。キーストアに含まれている証明書が 1 つだけの場合、エイリアスを指定する必要はありません。エイリアスを指定しない場合、ADT ではキーストアの最初のキーが使用されます。

キーストア管理アプリケーションによっては、証明書にエイリアスを割り当てることができません。例えば、Windows システムのキーストアを使用している場合は、証明書の識別名をエイリアスとして使用します。エイリアスを指定できるように、Java Keytool ユーティリティを使用して使用可能な証明書を一覧表示することができます。例えば、次のコマンドを実行します。

```
keytool -list -storetype Windows-MY
```

このコマンドでは、証明書に関する次のような出力が生成されます。

```
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
```

ADT コマンドラインでこの証明書を参照するには、エイリアスを次のように設定します。

```
CN=TestingCert,OU=QE,O=Adobe,C=US
```

Mac OS X では、Keychain での証明書のエイリアスは、Keychain Access アプリケーションに表示される名前です。

-storetype：キーストアのタイプです。キーストアの実装によって決まります。Java のほとんどのインストールに付属しているデフォルトのキーストア実装では、JKS タイプおよび PKCS12 タイプがサポートされています。Java 5.0 には、ハードウェアトークンのキーストアにアクセスするための PKCS11 タイプと、Mac OS X キーんチェーンにアクセスするための Keychain タイプのサポートが含まれています。Java 6.0 には、MSCAPI タイプ（Windows で使用）のサポートが含まれています。他の JCA プロバイダーがインストールされ、構成されている場合は、さらに別のキーストアタイプを使用できることがあります。キーストアタイプを指定しない場合、デフォルトの JCA プロバイダーのデフォルトタイプが使用されます。

ストアタイプ	キーストア形式	Java の必要最低バージョン
JKS	Java キーストアファイル（.keystore）	1.2
PKCS12	PKCS12 ファイル（.p12 または .pfx）	1.4
PKCS11	ハードウェアトークン	1.5
KeychainStore	Mac OS X キーチェーン	1.5
Windows-MY または Windows-ROOT	MSCAPI	1.6

-keystore：ファイルベースのストアタイプのキーストアファイルへのパスです。

-storepass：キーストアへのアクセスに必要なパスワードです。指定しない場合、パスワードの入力を求めるプロンプトが表示されます。

-keypass：AIR アプリケーションへの署名に使用する秘密キーへのアクセスに必要なパスワードです。指定しない場合、パスワードの入力を求めるプロンプトが表示されます。

注意：ADT コマンドの一部としてパスワードを入力する場合、パスワードの文字列はコマンドラインの履歴に保存されます。このため、証明書のセキュリティが重要となる場合には、-keypass または -storepass オプションの使用は推奨されません。また、パスワードオプションを省略した場合は、パスワードプロンプトで入力する文字列が表示されないことに注意してください（証明書のセキュリティを保護するため）。パスワードを入力し、Enter キーを押すだけです。

-providerName：指定したキーストアタイプの JCA プロバイダーです。指定しない場合、ADT ではそのキーストアタイプのデフォルトプロバイダーが使用されます。

-tsa：電子署名にタイムスタンプを付与する RFC3161 準拠のタイムスタンプサーバーの URL を指定します。URL を指定しない場合、GeoTrust が提供するデフォルトのタイムスタンプサーバーが使用されます。AIR アプリケーションの署名にタイムスタンプを付与すると、署名時にその証明書が有効であったことがタイムスタンプによって保証されるので、署名証明書の有効期限が切れた後でもアプリケーションをインストールできます。

ADT がタイムスタンプサーバーに接続できない場合、署名はキャンセルされ、パッケージは生成されません。タイムスタンプの付与を無効にするには、-tsa none を指定します。ただし、タイムスタンプなしでパッケージ化された AIR アプリケーションは、署名証明書の有効期限が切れるとインストールできなくなります。

注意：署名オプションの多くは、Java Keytool ユーティリティの同様のオプションに相当します。Keytool ユーティリティを使用すると、Windows でキーストアを調査することも、管理することもできます。Mac OS X での同じ目的のために、Apple® セキュリティユーティリティを使用することもできます。

-provisioning-profile：Apple iOS プロビジョニングファイルです（iOS アプリケーションをパッケージ化する場合のみ必須）。

署名オプションの例

.p12 ファイルを使用した署名

```
-storetype pkcs12 -keystore cert.p12
```

デフォルトの Java キーストアを使用した署名

```
-alias AIRcert -storetype jks
```

特定の Java キーストアを使用した署名

```
-alias AIRcert -storetype jks -keystore certStore.keystore
```

Mac OS X キーチェーンを使用した署名

```
-alias AIRcert -storetype KeychainStore -providerName Apple
```

Windows システムのキーストアを使用した署名

```
-alias cn=AIRCert -storeype Windows-MY
```

ハードウェアトークンを使用した署名（トークンを使用するように Java を構成する方法と正しい providerName 値については、トークンの製造メーカーの説明書を参照してください）

```
-alias AIRCert -storetype pkcs11 -providerName tokenProviderName
```

タイムスタンプの埋め込みなしでの署名

```
-storetype pkcs12 -keystore cert.p12 -tsa none
```

ファイルとパスのオプション

ファイルとパスのオプションによって、パッケージに含まれるすべてのファイルを指定します。ファイルとパスのオプションでは、次のシンタックスを使用します。

```
files_and_dirs -C dir files_and_dirs -e file_or_dir dir -extdir dir
```

files_and_dirs：AIR ファイルにパッケージ化するファイルとディレクトリです。空白文字で区切って、任意の数のファイルとディレクトリを指定できます。ディレクトリを指定すると、隠しファイルを除き、そのディレクトリ内のすべてのファイルとサブディレクトリがパッケージに追加されます（また、直接またはワイルドカードやディレクトリの展開によってアプリケーション記述ファイルが指定された場合は、無視され、パッケージにもう一度追加されることはありません）。指定するファイルとディレクトリは、現在のディレクトリまたはそのサブディレクトリのいずれかに含まれている必要があります。現在のディレクトリを変更するには、-C オプションを使用します。

重要：–C オプションに続く file_or_dir 引数では、ワイルドカードを使用できません（コマンドシェルでは引数を ADT に渡す前にワイルドカードを展開するので、ADT が誤った場所のファイルを検索することになります）。ただし、現在のディレクトリを表すドット文字（.）は使用できます。例えば、「-C assets .」と指定した場合は、サブディレクトリを含め、**assets** ディレクトリ内のすべてのコンテンツが、アプリケーションパッケージのルートレベルにコピーされます。

-C dir files_and_dirs：アプリケーションパッケージに追加された後続のファイルおよびディレクトリ（**files_and_dirs** で指定）を処理する前に、作業ディレクトリを **dir** の値に変更します。ファイルまたはディレクトリはアプリケーションパッケージのルートに追加されます。–C オプションは、ファイルシステムの複数の箇所からファイルを含めるために何度でも使用できます。dir に相対パスを指定した場合、そのパスは常に元の作業ディレクトリを基準に解決されます。

パッケージに含まれるファイルとディレクトリを ADT で処理するときに、現在のディレクトリとターゲットファイルの間の相対パスが格納されます。パッケージがインストールされるとき、これらのパスがアプリケーションのディレクトリ構造に展開されます。したがって、-C release/bin lib/feature.swf と指定すると、ファイル release/bin/lib/feature.swf は、ルートアプリケーションフォルダーの lib サブディレクトリに配置されます。

-e file_or_dir dir：指定したパッケージディレクトリにファイルやディレクトリを配置します。このオプションは、ANE ファイルをパッケージ化する場合は使用できません。

注意：アプリケーション記述ファイルの <content> エレメントで、アプリケーションパッケージのディレクトリツリー内における、メインアプリケーションファイルの最終的な場所を指定する必要があります。

-extdir dir：***dir*** の値は、ネイティブ拡張（ANE ファイル）を検索するためのディレクトリの名前です。絶対パス、または現在のディレクトリに対する相対パスを指定します。-extdir オプションは複数回指定できます。

指定したディレクトリには、アプリケーションで使用するネイティブ拡張用の ANE ファイルが含まれます。このディレクトリ内の ANE ファイルには、それぞれ .ane という拡張子が付きます。ただし、.ane という拡張子を除くファイル名は、アプリケーション記述ファイルの extensionID エレメントの値と一致している必要はありません。

例えば、-extdir ./extensions を使用した場合、extensions ディレクトリは次のようになります。

```
extensions/
    extension1.ane
    extension2.ane
```

注意：-extdir オプションの使用方法は、ADT ツールと ADL ツールで異なります。ADL では、このオプションは、サブディレクトリを含んだディレクトリを指定します。各サブディレクトリにはパッケージ化されていない ANE ファイルが格納されています。ADT では、このオプションは、ANE ファイルが格納されているディレクトリを指定します。

デバッガー接続のオプション

パッケージのターゲットが apk-debug、ipa-debug または ipa-debug-interpreter である場合は、デバッガー接続のオプションを使用して、アプリケーションがリモートデバッガーへの接続を試みる（通常、Wi-Fi デバッグの場合に使用）のか、リモートデバッガーからの着信接続を監視する（通常、USB デバッグの場合に使用）のかを指定できます。デバッガーに接続する場合は -connect オプションを使用し、また、USB 接続経由でデバッガーからの接続を受け入れる場合は -listen オプションを使用します。これらのオプションは相互に排他的です。つまり、これらを同時に使用することはできません。

-connect オプションでは、次のシンタックスを使用します。

```
-connect hostString
```

-connect：指定した場合、アプリケーションはリモートデバッガーへの接続を試行します。

hostString：Flash デバッグツール FDB を実行しているコンピューターを特定する文字列です。指定しない場合、アプリケーションは、パッケージが作成されるコンピューター上で実行されているデバッガーへ接続しようとします。ホスト文字列は、完全修飾のコンピュータードメイン名（**machinename.subgroup.example.com** など）、または IP アドレス（**192.168.4.122** など）にすることができます。指定した（またはデフォルトの）コンピューターが見つからない場合、ランタイムによって、有効なホスト名を要求するダイアログが表示されます。

-listen オプションでは、次のシンタックスを使用します。

```
-listen port
```

-listen：指定すると、ランタイムはリモートデバッガーからの接続を待ちます。

port：（オプション）監視するポートです。デフォルトでは、ランタイムはポート 7936 で監視します。-listen オプションの使用について詳しくは、105 ページの「FDB による USB 経由のリモートデバッグ」を参照してください。

Android アプリケーションのプロファイリングオプション

パッケージのターゲットが apk-profile である場合、プロファイラーオプションを使用して、パフォーマンスおよびメモリのプロファイリングに使用するためのプリロード済み SWF ファイルを指定できます。プロファイラーオプションでは、次のシンタックスを使用します。

```
-preloadSWFPath directory
```

-preloadSWFPath：指定した場合、アプリケーションは指定したディレクトリでプリロード SWF の検索を試みます。指定しない場合、ADT では AIR SDK のプリロード SWF ファイルが使用されます。

directory：プロファイラーのプリロード SWF ファイルを含むディレクトリです。

ネイティブ拡張のオプション

ネイティブ拡張のオプションでは、ネイティブ拡張用に ANE ファイルをパッケージ化するためのオプションとファイルを指定します。これらのオプションは、-target オプションに ane が指定されている ADT パッケージコマンドと共に使用します。

```
extension-descriptor -swc swcPath
   -platform platformName
  -platformoptions path/platform.xml
   FILE_OPTIONS
```

extension-descriptor：ネイティブ拡張の記述ファイルです。

-swc：ActionScript コードとネイティブ拡張のリソースを含む SWC ファイルです。

-platform：ANE ファイルがサポートするプラットフォームの名前です。複数の -platform オプションを含めることができ、各オプションには独自の FILE_OPTIONS を指定できます。

-platformoptions：プラットフォームオプション（platform.xml）ファイルへのパスです。拡張で使用する、デフォルト以外のリンカーオプション、共有ライブラリおよびサードパーティの静的ライブラリを指定するには、このファイルを使用します。詳細と例については、iOS のネイティブライブラリを参照してください。

FILE_OPTIONS：ネイティブ拡張パッケージに含める静的ライブラリなど、パッケージに含めるネイティブプラットフォームファイルを指定します。ファイルオプションについては、178 ページの「ファイルとパスのオプション」で詳しく説明しています（ANE ファイルをパッケージ化する際には、-e オプションは使用できません）。

関連項目

ネイティブ拡張のパッケージ化

ADT エラーメッセージ

次の表に、ADT プログラムによって報告される可能性があるエラーおよび考えられる原因を示します。

アプリケーション記述子の検証エラー

エラーコード	説明	注記
100	アプリケーション記述子を解析できません	閉じられていないタグなどの XML シンタックスエラーをアプリケーション記述ファイルで確認してください。
101	名前空間が見つかりません	見つからない名前空間を追加してください。
102	名前空間が無効です	名前空間のスペルを確認してください。
103	予期しないエレメントまたは属性です	問題のあるエレメントと属性を削除してください。記述ファイルではカスタム値は使用できません。 エレメント名と属性名のスペルを確認してください。 エレメントが正しい親エレメント内に配置され、属性が正しいエレメントと共に使用されていることを確認してください。
104	エレメントまたは属性が見つかりません	必要なエレメントまたは属性を追加してください。
105	エレメントまたは属性に無効な値が含まれています	問題のある値を修正してください。
106	ウィンドウ属性の組み合わせが無効です	ウィンドウ設定には、transparency = true と systemChrome = standard など、同時に指定できない組み合わせがあります。同時に指定できない設定のいずれか 1 つを変更してください。

エラーコード	説明	注記
107	ウィンドウの最小サイズがウィンドウの最大サイズを超えています	最小サイズまたは最大サイズの設定を変更してください。
108	属性が前のエレメントで既に使用されています	
109	重複するエレメントです。	重複するエレメントを削除してください。
110	指定したタイプの少なくとも 1 つのエレメントが必要です。	見つからないエレメントを追加してください。
111	アプリケーション記述子に示されているプロファイルはいずれも、ネイティブ拡張をサポートしません。	ネイティブ拡張をサポートする supportedProfies リストにプロファイルを追加してください
112	AIR のターゲットはネイティブ拡張をサポートしません。	ネイティブ拡張をサポートするターゲットを選択してください。
113	<nativeLibrary> と <initializer> は一緒に指定する必要があります。	ネイティブ拡張内のネイティブライブラリごとにイニシャライザー関数を指定する必要があります。
114	<nativeLibrary> を指定しない状態で <finalizer> が見つかりました。	プラットフォームがネイティブライブラリを使用しない限り、ファイナライザーを指定しないでください。
115	デフォルトのプラットフォームにネイティブ実装を含めることはできません。	デフォルトの platform エレメントにはネイティブライブラリを指定しないでください。
116	ブラウザーの呼び出しは、このターゲット用にはサポートされていません。	指定したパッケージターゲットに対しては、<allowBrowserInvocation> エレメントを true にすることはできません。
117	ネイティブ拡張をパッケージ化するには、このターゲットに少なくとも名前空間 n が必要です。	アプリケーション記述子の AIR 名前空間を、サポートされている値に変更してください。

名前空間、エレメント、属性およびそれらの有効な値について詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

アプリケーションアイコンのエラー

エラーコード	説明	注記
200	アイコンファイルを開けません	ファイルが指定されたパスに存在することを確認してください。 別のアプリケーションを使用して、ファイルを開けることを確認してください。
201	アイコンのサイズが間違っています	アイコンのサイズ（ピクセル）は、XML タグと一致している必要があります。例えば、 <image32x32>icon.png</image32x32> というアプリケーション記述エレメントが指定されている場合、icon.png 内の画像は 32 x 32 ピクセルである必要があります。
202	アイコンファイルにサポートされていない画像形式が含まれています	PNG 形式のみがサポートされています。アプリケーションをパッケージ化する前に、他の形式に画像を変換してください。

アプリケーションファイルのエラー

エラーコード	説明	注記
300	ファイルが見つからないか、開けません	コマンドラインで指定されたファイルが見つからないか、開けません。
301	アプリケーション記述ファイルが見つからないか、開けません	指定されたパスでアプリケーション記述ファイルが見つからないか、開けません。
302	パッケージにルートコンテンツファイルがありません	アプリケーション記述子の <content> エレメントで参照されている SWF ファイルまたは HTML ファイルを、ADT コマンドラインに指定するファイルリストに含めることでパッケージに追加する必要があります。
303	パッケージにアイコンファイルがありません	ADT コマンドラインに記載されているファイルに含めることで、アプリケーション記述子で指定されているアイコンファイルをパッケージに追加する必要があります。アイコンファイルは自動的に追加されません。
304	初期ウィンドウコンテンツが無効です	アプリケーション記述子の <content> エレメントで参照されているファイルは、有効な HTML ファイルまたは SWF ファイルとして認識されません。
305	初期ウィンドウコンテンツの SWF バージョンが名前空間よりも後のバージョンです	アプリケーション記述子の <content> エレメントで参照されているファイルの SWF バージョンは、記述子の名前空間で指定されている AIR のバージョンでサポートされていません。例えば、SWF10（Flash Player 10）ファイルを AIR 1.1 アプリケーションの初期コンテンツとしてパッケージ化しようとすると、このエラーが生成されます。
306	プロファイルがサポートされていません。	アプリケーション記述ファイルに指定したプロファイルは、サポートされていません。234 ページの「supportedProfiles」を参照してください。
307	名前空間は少なくとも **nnn** であることが必要です。	アプリケーションで使用される機能に対して適切な名前空間を使用してください（2.0 名前空間など）。

その他のエラーの終了コード

終了コード	説明	注記
2	使用法エラー	コマンドライン引数にエラーがないか確認してください。
5	不明なエラー	このエラーは、一般的なエラーの状態では説明できない状況を示します。考えられる根本的な原因には、ADT と Java ランタイム環境の非互換性、ADT または JRE インストールの破損、ADT 内のプログラミングエラーなどがあります。
6	出力ディレクトリに書き込めませんでした	指定された（または暗示された）出力ディレクトリにアクセスでき、コンテナとなっているドライブに十分なディスク領域があることを確認してください。

終了コード	説明	注記
7	証明書にアクセスできませんでした	キーストアへのパスが正しく指定されていることを確認してください。 キーストア内の証明書にアクセスできることを確認してください。Java 1.6 Keytool ユーティリティを使用すると、証明書のアクセスに関する問題をトラブルシューティングできます。
8	証明書が無効です	証明書ファイルの形式が不正か、変更、期限切れ、または失効しています。
9	AIR ファイルに署名できませんでした	ADT に渡された署名オプションを確認してください。
10	タイムスタンプを作成できませんでした	ADT はタイムスタンプサーバーへの接続を確立できませんでした。プロキシサーバー経由でインターネットに接続する場合、JRE プロキシ設定を構成する必要があります。
11	証明書の作成エラー	署名の作成に使用されるコマンドライン引数を確認してください。
12	入力が無効です	コマンドラインで ADT に渡されたファイルパスおよびその他の引数を確認してください。
13	デバイス SDK が見つかりません	デバイス SDK の構成を確認してください。ADT では、指定したコマンドの実行に必要なデバイス SDK を見つけることができません。
14	デバイスエラー	ADT では、デバイスの制限または問題が原因でコマンドを実行できません。例えば、実際にはインストールされていないアプリケーションをアンインストールしようとすると、この終了コードが発行されます。
15	デバイスがありません	デバイスが接続されオンになっているか、エミュレーターが実行されているかどうかを確認してください。
16	GPL コンポーネントが見つかりません	現在の AIR SDK に、要求された操作を実行するために必要なすべてのコンポーネントが含まれていません。
17	デバイスパッケージ化ツールが失敗しました。	必要なオペレーティングシステムコンポーネントが見つからないので、パッケージを作成できませんでした。

Android エラー

終了コード	説明	注記
400	現在の Android SDK のバージョンが属性をサポートしていません。	属性名のスペルが正しく、表示されるエレメントに対して有効な属性であることを確認してください。属性が Android 2.2 より後に導入されたものであると、場合によっては、ADT コマンドに -platformsdk フラグを設定する必要があります。
401	現在の Android SDK のバージョンが属性値をサポートしていません。	属性値のスペルが正しく、属性に対して有効な値であることを確認してください。属性値が Android 2.2 より後に導入されたものであると、場合によっては、ADT コマンドに -platformsdk フラグを設定する必要があります。
402	現在の Android SDK のバージョンが XML タグをサポートしていません。	XML タグ名のスペルが正しく、有効な Android マニフェストのドキュメントエレメントであることを確認してください。エレメントが Android 2.2 より後に導入されたものであると、場合によっては、ADT コマンドに -platformsdk フラグを設定する必要があります。
403	Android タグをオーバーライドすることはできません。	アプリケーションが、AIR で使用するために予約された Android マニフェストエレメントをオーバーライドしようとしました。75 ページの「Android の設定」を参照してください。
404	Android 属性をオーバーライドすることはできません。	アプリケーションが、AIR で使用するために予約された Android マニフェスト属性をオーバーライドしようとしました。75 ページの「Android の設定」を参照してください。
405	Android タグ %1 は、manifestAdditions タグの最初のエレメントにする必要があります。	指定したタグを必要な位置に移動してください。
406	Android タグ %2 の属性 %1 に無効な値 %3 が含まれています。	この属性に有効な値を指定してください。

ADT 環境変数

ADT は、次の環境変数の値を読み取ります（設定されている場合）。

AIR_ANDROID_SDK_HOME：Android SDK のルートディレクトリへのパスを指定します（このディレクトリには tools フォルダーが含まれています）。AIR 2.6 以降の SDK には、関連する ADT コマンドの実装に必要な Android SDK のツールが含まれています。異なるバージョンの Android SDK を使用する場合にのみ、この値を設定してください。この変数が設定されている場合、その値を必要とする ADT コマンドを実行するときに、-platformsdk オプションを指定する必要はありません。この変数とコマンドラインオプションの両方が設定されている場合、コマンドラインで指定したパスが使用されます。

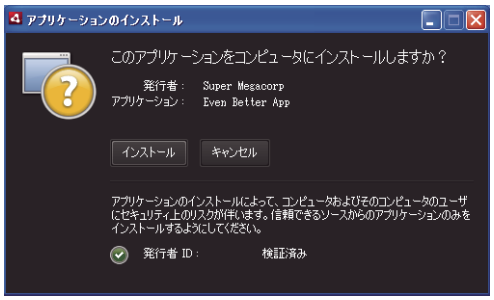
AIR_EXTENSION_PATH：アプリケーションで必要なネイティブ拡張を検索するためのディレクトリのリストを指定します。このディレクトリのリストは、ADT コマンドラインで指定されているネイティブ拡張ディレクトリの後に検索されます。ADL コマンドでも、この環境変数を使用します。

注意：一部のコンピューターシステムでは、この環境変数に格納されているファイルシステムパス内の 2 バイト文字が、誤って解釈される場合があります。この場合、ADT の実行に使用する JRE を、UTF-8 文字セットを使用するように設定してみてください。この設定は、Mac および Linux での ADT の実行に使用するスクリプトでは、デフォルトで行われます。Windows の adt.bat ファイルの場合、または Java から直接 ADT を実行する場合は、Java コマンドラインで -Dfile.encoding=UTF-8 オプションを指定します。

第 13 章：AIR アプリケーションへの署名

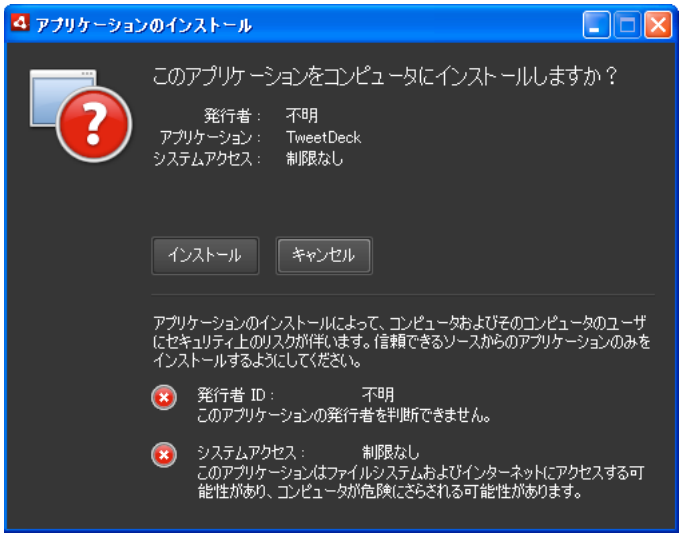
AIR ファイルへの電子署名

認定されている証明機関（CA）が発行した証明書を使用して AIR インストールファイルに電子署名すると、インストールしているアプリケーションが誤ってまたは悪意をもって変更されていないことをユーザーに確実に保証することができ、署名したユーザーは署名者（発行者）と見なされます。AIR のインストール中に発行者名が表示されるのは、信頼できる証明書、またはインストールされているコンピューターで信頼されている証明書にチェーン化されている証明書を使用して AIR アプリケーションに署名している場合です。



信頼できる証明書によって署名されたアプリケーションのインストールの確認ダイアログ

自己署名証明書（または信頼できる証明書にチェーン化されていない証明書）でアプリケーションに署名する場合、ユーザーは、より大きなセキュリティリスクがアプリケーションのインストールに伴うことを認識しておく必要があります。インストールダイアログにはこの追加のリスクが示されます。



自己署名証明書によって署名されたアプリケーションのインストールの確認ダイアログ

重要：悪意のあるエンティティは、なんらかの方法で署名されているキーストアファイルを取得するか、秘密キーを見つけ出し、署名者の ID で AIR ファイルを偽造する可能性があります。

コード署名証明書

コード署名証明書の使用に関連するセキュリティの保証、制限、および法的義務の概要については、証明機関運用規定（CPS）および発行側の証明機関が発行する利用規約に記載されています。AIR コードサイニング証明書を現在発行している証明機関の規約について詳しくは、次のサイトを参照してください。

ChosenSecurity（http://www.chosensecurity.com/products/tc_publisher_id_adobe_air.htm）

ChosenSecurity CPS（http://www.chosensecurity.com/resource_center/repository.htm）

GlobalSign（http://www.globalsign.com/code-signing/index.html）

GlobalSign CPS（http://www.globalsign.com/repository/index.htm）

Thawte CPS（http://www.thawte.com/cps/index.html）

日本ベリサイン CPS（VeriSign Japan CPS）（http://www.verisign.co.jp/repository/CPS/）

クライアント ID 利用規約（Digital ID 利用規約）（https://www.verisign.co.jp/repository/SUBAGR.html）

AIR コード署名について

AIR ファイルに署名が行われると、電子署名がインストールファイルに含まれます。署名には、署名後に AIR ファイルが変更されていないことを検証するために使用されるパッケージのダイジェストと、発行者 ID を検証するために使用する署名証明書に関する情報が含まれます。

AIR では、証明書が信頼できるかどうかを確認するために、オペレーティングシステムの証明書ストアを通じてサポートされる公開キーインフラストラクチャ（PKI）を使用します。AIR アプリケーションがインストールされているコンピューターでは、発行側の情報を検証するために、AIR アプリケーションの署名に使用されている証明書そのものを信頼するか、信頼できる証明機関の証明書にリンクしている証明書のチェーンを信頼する必要があります。

AIR ファイルが、信頼できるルート証明書（およびこれには通常すべての自己署名入り証明書が含まれる）のいずれにもチェーン化されていない証明書で署名されている場合、発行者側の情報は検証できません。AIR は、AIR パッケージが署名後に変更されていないか確認できますが、ファイルの実際の作成者および署名者を知ることはできません。

注意：ユーザーは自己署名入り証明書を信頼することができ、その証明書で署名された AIR アプリケーションには、発行者名として証明書の共通名フィールドの値が表示されます。AIR では、ユーザーが信頼済みの証明書を指定することはできません。証明書（秘密キーを含まない）はユーザーに個別に提供する必要があります。ユーザーはオペレーティングシステムが提供するメカニズムのいずれかを使用するか、適切なツールを使用してシステムの証明書ストア内の適切な場所に証明書を読み込む必要があります。

AIR 発行者 ID について

重要：AIR 1.5.3 で発行者 ID は非推奨となり、コードサイニング証明書に基づいて算出されなくなりました。新しいアプリケーションに発行者 ID は不要なので、使用しないでください。既存のアプリケーションを更新する場合は、アプリケーション記述ファイルに元の発行者 ID を指定する必要があります。

AIR 1.5.3 よりも前のバージョンでは、AIR ファイルのインストール時に、AIR アプリケーションのインストーラーによって発行者 ID が生成されました。これは、AIR ファイルの署名に使用される証明書に固有の ID です。複数の AIR アプリケーションに対して同じ証明書を再利用する場合は、それらのアプリケーションの発行者 ID は同じになります。別の証明書や場合によっては元の証明書の更新されたインスタンスを使用してアプリケーションのアップデートに署名すると、発行者 ID が変更されます。

AIR 1.5.3 以降では、AIR によって発行者 ID が割り当てられることはありません。AIR 1.5.3 を使用して発行するアプリケーションでは、アプリケーション記述子に発行者 ID を指定できます。AIR 1.5.3 よりも前のバージョンで発行されているアプリケーションに対してアップデートを発行するときにのみ、発行者 ID を指定します。アプリケーション記述子に元の ID を指定しないと、新しい AIR パッケージは既存のアプリケーションに対するアップデートとして扱われません。

元の発行者 ID を確認するには、META-INF/AIR サブディレクトリにある publisherid ファイルを探します。このディレクトリには、元のアプリケーションがインストールされています。このファイル内のストリングが 発行者 ID です。発行者 ID を手動で指定するには、アプリケーション記述ファイル内で、アプリケーション記述子の名前空間宣言に AIR 1.5.3（またはそれ以降の）ランタイムが指定されている必要があります。

発行者 ID が指定されている場合は、次の目的で使用します。

- 暗号化されたローカルストアの暗号化キーの一部として
- アプリケーション記憶域ディレクトリのパスの一部として
- ローカル接続の接続ストリングの一部として
- AIR ブラウザー API でアプリケーションを呼び出すときに使用する識別ストリングの一部として
- OSID（カスタムインストール / アンインストールプログラムの作成時に使用）の一部として

発行者 ID が変更されると、ID に依存している AIR の機能の動作も変わります。例えば、既存の暗号化されたローカルストアにあるデータにはアクセスできなくなります。また、Flash または AIR インスタンスでアプリケーションへのローカル接続を確立するには、接続ストリングに新しい ID を使用する必要があります。インストール済みアプリケーションの発行者 ID は、AIR 1.5.3 以降では変更できません。AIR パッケージの発行時に別の発行者 ID を使用すると、インストーラーは新しいパッケージをアップデートではなく別のアプリケーションとして扱います。

証明書の形式について

AIR 署名ツールは、Java Cryptography Architecture（JCA）を通じてアクセスできるすべてのキーストアを受け入れます。これには、PKCS12 形式のファイル（通常 .pfx または .p12 ファイル拡張子が付いている）などのファイルベースのキーストア、Java .keystore ファイル、PKCS11 ハードウェアキーストア、およびシステムキーストアが含まれます。ADT がアクセスできるキーストア形式は、ADT の実行に使用される Java ランタイムのバージョンと設定によって異なります。PKCS11 ハードウェアトークンなど一部の種類のキーストアにアクセスするには、追加のソフトウェアドライバーや JCA プラグインのインストールと設定が必要になる場合があります。

AIR ファイルに署名する際は、既にあるコードサイニング証明書をほとんどの場合使用できます。また、AIR アプリケーションの署名用であることを明示して発行された新しい証明書を取得することもできます。例えば、VeriSign、Thawte、GlobalSign または ChosenSecurity の次の種類の証明書はどれでも使用できます。

- ChosenSecurity
  - Adobe AIR 用の TC Publisher ID
- GlobalSign
  - ObjectSign Code Signing Certificate
- Thawte：
  - AIR Developer Certificate
  - Apple Developer Certificate
  - JavaSoft Developer Certificate
  - Microsoft Authenticode Certificate
- VeriSign：
  - Adobe AIR Digital ID
  - Microsoft Authenticode Digital ID
  - Sun Java Signing Digital ID

注意：証明書は、コード署名用に作成されたものである必要があります。SSL やその他の種類の証明書を使用して AIR ファイルに署名することはできません。

タイムスタンプ

AIR ファイルに署名するときは、パッケージングツールで、タイムスタンプ局のサーバーを照会して、単独で検証可能な署名の日時を取得します。取得したタイムスタンプは AIR ファイルに埋め込まれます。署名した証明書が署名時に有効である限り、AIR ファイルは証明書の期限が切れた後もインストールできます。一方、タイムスタンプを取得していない場合は、証明書の期限が切れたり証明書が取り消されたりすると、AIR ファイルはインストールできなくなります。

デフォルトでは、AIR パッケージングツールでタイムスタンプを取得します。ただし、タイムスタンプサービスが使用できないときにアプリケーションをパッケージ化できるようにするには、タイムスタンプ機能をオフにします。公開配布されているすべての AIR ファイルにはタイムスタンプを含めることをお勧めします。

AIR パッケージングツールで使用されるデフォルトのタイムスタンプ局は、Geotrust です。

証明書の取得

証明書を取得するには、通常は証明機関の Web サイトにアクセスし、その会社の取得プロセスに従います。AIR ツールに必要なキーストアファイルを生成するツールは、購入した証明書の種類、受信側コンピューターでの証明書の格納方法、および場合によっては証明書の取得に使用するブラウザーによって異なります。例えば、Adobe Developer Certificate を Thawte から取得して書き出すには、Mozilla Firefox を使用する必要があります。これにより、Firefox ユーザーインターフェイスから証明書を .p12 または .pfx ファイルとして直接書き出せます。

注意：Java バージョン 1.5 以上では、PKCS12 証明書ファイルを保護するパスワードに high-ASCII（拡張 ASCII）文字を使用できません。AIR 開発ツールでは、署名された AIR パッケージの作成に Java を使用します。証明書を .p12 または .pfx ファイルとして書き出す場合、パスワードには正規の ASCII 文字のみを使用してください。

AIR インストールファイルのパッケージ化に使用する AIR 開発ツール（ADT）を使用して自己署名入り証明書を生成できます。一部のサードパーティツールも使用できます。

自己署名入り証明書の生成手順、および AIR ファイルの署名手順については、162 ページの「AIR 開発ツール（ADT）」を参照してください。また、Flash Builder、Dreamweaver、および AIR update for Flash を使用して AIR ファイルを書き出し、署名することもできます。

次の例で、Thawte 証明機関から AIR Developer Certificate を取得し、それを ADT で使用できるように準備する方法を説明します。

例：Thawte からの AIR Developer Certificate の取得

注意：コードサイニング証明書を取得して準備する方法は他にも多数あり、この例は単にその 1 つに過ぎません。ポリシーや手続きは証明機関ごとに異なります。

AIR Developer Certificate を購入するには、Thawte Web サイトへのアクセスに Mozilla Firefox ブラウザーを使用する必要があります。証明書の秘密キーは、ブラウザーのキーストアに格納されます。Firefox キーストアがマスタパスワードで保護されていること、およびコンピューターそのものが物理的に安全であることを確認してください（取得プロセスが完了すると、ブラウザーのキーストアから証明書と秘密キーを書き出して削除できます）。

証明書登録プロセス中に、秘密キーと公開キーのペアが生成されます。秘密キーは自動的に Firefox キーストア内に格納されます。Thawte の Web サイトに証明書を要求する場合も証明書を取得する場合にも同じコンピューターとブラウザーを使用する必要があります。

1. Thawte の Web サイトにアクセスし、コードサイニング証明書の製品ページを参照してください。
2. コードサイニング証明書の一覧から、「Adobe AIR Developer Certificate」を選択します。
3. 3 つのステップの登録プロセスを完了します。組織および連絡先の情報を指定する必要があります。これで、Thawte は、ID 検証プロセスを実行し、追加の情報を要求できます。検証の完了後、Thawte から証明書の取得手順に関する電子メールが送られてきます。

注意： 必要なドキュメントの種類について詳しくは、https://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/enroll_codesign_eng.pdf を参照してください。

4 発行された証明書を Thawte サイトから取得します。証明書は、自動的に Firefox キーストアに保存されます。

5 次の手順を実行して、秘密キーと証明書を含むキーストアファイルを Firefox キーストアから書き出します。

注意：Firefox から秘密キーと証明書を書き出す場合、そのファイルは、ADT、Flex、Flash および Dreamweaver で使用できる .p12（pfx）形式で書き出されます。

a Firefox の証明書マネージャーダイアログを開きます。

b Windows の場合は、ツール／オプション／詳細／暗号化／証明書を表示を開きます。

c Mac OS の場合は、Firefox ／環境設定／詳細／暗号化／証明書を表示を開きます。

d Linux の場合は、編集／設定／詳細／暗号化／証明書を表示を開きます。

e 証明書の一覧から「Adobe AIR Code Signing Certificate」を選択し、「**バックアップ**」ボタンをクリックします。

f キーストアファイルを書き出すファイル名と場所を入力し、「**保存**」をクリックします。

g Firefox のマスタパスワードを使用している場合は、ファイルを書き出すためにソフトウェアセキュリティデバイスのパスワードの入力が要求されます（このパスワードは Firefox のみで使用されます）。

h 証明書バックアップパスワードの選択ダイアログボックスで、キーストアファイルのパスワードを作成します。

重要：このパスワードによってキーストアファイルは保護されます。このパスワードは、AIR アプリケーションへの署名にファイルが使用される際に必要になります。安全なパスワードを選択してください。

i 「OK」をクリックします。パスワードのバックアップが成功したことを示すメッセージが表示されます。秘密キーを含むキーストアファイルは、.p12 ファイル拡張子（PKCS12 形式）で保存されます。

6 書き出したキーストアファイルを ADT、Flash Builder、Flash Professional または Dreamweaver で使用します。ファイルに対して作成されたパスワードは、AIR アプリケーションが署名されるたびに必要になります。

重要：秘密キーと証明書は Firefox キーストアにも格納されます。これにより証明書ファイルの追加のコピーを書き出せますが、証明書と秘密キーのセキュリティを維持するために保護する必要がある別のアクセスポイントも提供されます。

証明書の変更

場合によっては、AIR アプリケーションのアップデートの署名に使用する証明書を変更する必要があります。これが必要になるのは、次のような場合です。

- 元の署名証明書を更新する
- 自己署名入り証明書を証明機関から発行された証明書にアップグレードする
- 有効期限が近い自己署名入り証明書を別の証明書に変更する
- ある商用証明書を別の商用証明書に変更する（企業の ID が変更された場合など）

AIR で AIR ファイルをアップデートとして認識できるようにするには、元の AIR ファイルとアップデートの AIR ファイルに同じ証明書を使用して署名するか、アップデートに証明書の移行署名を適用します。移行署名は、アップデートの AIR パッケージに適用される 2 つ目の署名です。これには元の証明書が使用されます。移行署名では元の証明書を使用して、署名者がアプリケーションの元の発行者であることを証明します。

移行署名が適用された AIR ファイルがインストールされると、新しい証明書がプライマリ証明書になります。後続のアップデートに移行署名を適用する必要はありません。ただし、移行署名はできるだけ長い期間にわたって適用し、アップデートを実行していないユーザーに対応する必要があります。

重要：証明書の期限が切れる前に、証明書を変更し、元の証明書を使用してアップデートに移行署名を適用する必要があります。これを行わない場合は、新しいバージョンをインストールする前に、アプリケーションの既存のバージョンをアンインストールする必要があります。AIR 1.5.3 以降では、期限切れ後 365 日の猶予期間以内であれば、期限切れの証明書を使用して移行署名を適用できます。ただし、期限切れの証明書を使用してメインアプリケーションに署名を適用することはできません。

証明書を変更するには：

1 アプリケーションのアップデートを作成します。

2 アップデートの AIR ファイルをパッケージ化し、**新しい**証明書で署名します。

3 **元の**証明書を使用して（ADT の -migrate コマンドを使用して）AIR ファイルにもう一度署名します。

移行署名を含む AIR ファイルは、別の意味では通常の AIR ファイルです。元のバージョンがないシステムにアプリケーションをインストールした場合、AIR では通常の方法で新しいバージョンをインストールします。

注意：AIR 1.5.3 よりも前のバージョンでは、更新された証明書を使用して AIR アプリケーションに署名する場合、必ずしも移行署名は必要ありませんでした。AIR 1.5.3 以降、更新された証明書には必ず移行署名が必要です。

移行署名を適用するには、170 ページの「ADT migrate コマンド」を使用します。詳しくは、195 ページの「AIR アプリケーションのアップデートバージョンの署名」を参照してください。

注意：ADT migrate コマンドは、ネイティブ拡張を含む AIR デスクトップアプリケーションでは使用できません。これらのアプリケーションは、.air ファイルではなくネイティブインストーラーとしてパッケージ化されているからです。ネイティブ拡張を含む AIR デスクトップアプリケーション向けの証明書を変更するには、163 ページの「ADT package コマンド」を -migrate フラグ付きで使用してアプリケーションをパッケージ化してください。

アプリケーション ID の変更

AIR 1.5.3 よりも前のバージョンでは、移行署名を使用して署名されたアップデートがインストールされると、AIR アプリケーションの ID が変更されていました。アプリケーションの ID の変更には、次のような影響があります。

- 新しいバージョンのアプリケーションでは、既存の暗号化されたローカルストアにあるデータにアクセスできません。
- アプリケーション記憶域ディレクトリの場所が変更されます。以前の場所にあるデータが新しいディレクトリにコピーされません（ただし、新しいアプリケーションは、以前の発行者 ID に基づく元のディレクトリには配置できません）。
- アプリケーションは、以前の発行者 ID を使用してローカル接続を開くことができなくなります。
- Web ページからのアプリケーションへのアクセスに使用する識別ストリングが変更されます。
- アプリケーションの OSID が変更されます（OSID はカスタムインストール / アンインストールプログラムの記述時に使用します）。

AIR 1.5.3 以降を使用してアップデートを発行する場合、アプリケーション ID は変更できません。元のアプリケーションと発行者 ID は、アップデートの AIR ファイルのアプリケーション記述子に指定されている必要があります。指定されていない場合、新しいパッケージはアップデートとして認識されません。

注意：AIR 1.5.3 以降を使用して新しい AIR アプリケーションを発行する場合、発行者 ID は指定しないでください。

用語集

ここでは、公開配布するアプリケーションへの署名方法を決定する際に理解しておく必要があるいくつかの用語を示します。

用語	説明
証明機関（CA）	信頼できるサードパーティの役割を果たし、最終的に公開キーの所有者の ID を認定する公開キーインフラストラクチャネットワーク内のエンティティ。CA は一般的に、所有している秘密キーで署名した電子証明書を発行し、証明書の所有者の ID が検証済みであることを証明します。
Certificate Practice Statement（CPS）	証明書を発行および検証する証明機関の運用とポリシーを規定します。CPS は、CA とその加入者およびその関係者間で結ばれる契約の一部です。また、証明機関が提供する証明書の ID 検証のポリシーおよび保証のレベルについても記載されています。
Certificate Revocation List（CRL）	発行済み証明書の中で、取り消され、信頼されなくなった証明書のリスト。AIR は、AIR アプリケーションの署名時に CRL をチェックし、タイムスタンプがない場合は、もう一度アプリケーションをインストールします。
証明書チェーン	証明書チェーンは一連の証明書のことで、チェーン内の各証明書はその次の証明書で署名されています。
電子証明書	所有者の ID、所有者の公開キー、および証明書自体の ID で構成されている電子ドキュメント。証明機関で発行された証明書自体は、発行側の CA に属する証明書で署名されます。
電子署名	公開キーと秘密キーのペアの公開キーでのみ解読できる暗号化されたメッセージまたはダイジェスト。PKI では、電子署名には、最終的に証明機関で追跡可能な 1 つ以上の電子証明書が含まれます。電子署名は、署名後に（使用されている暗号化アルゴリズムが提供する保証の制限内で）改ざんされていないメッセージ（またはコンピューターファイル）の検証に使用でき、発行側の証明機関、署名者の ID が信頼できることが前提になります。
キーストア	電子証明書、および関連の秘密キーが含まれる場合もあるデータベース。
Java Cryptography Architecture（JCA）	キーストアを管理したり、キーストアにアクセスするための拡張可能なアーキテクチャ。詳しくは、『Java Cryptography Architecture Reference Guide』（英語）を参照してください。
PKCS #11	RSA Laboratories が策定した暗号化トークンインターフェイス標準。ハードウェアトークンベースのキーストアです。
PKCS #12	RSA Laboratories が策定した個人情報交換シンタックス標準。通常秘密キーと関連の電子証明書を含むファイルベースのキーストアです。
秘密キー	2 つの部分から構成される公開キーと秘密キーの非対称暗号化システムの秘密の部分。秘密キーは、秘密を保持する必要があり、ネットワーク上で転送されません。電子署名されたメッセージは、署名者によって秘密キーで暗号化されます。
公開キー	2 つの部分から構成される公開キーと秘密キーの非対称暗号化システムの公開の部分。公開キーは自由に利用でき、秘密キーで暗号化されたメッセージの解読に使用されます。
公開キーインフラストラクチャ（PKI）	証明機関が公開キーの所有者の ID を証明するための信頼性のシステム。ネットワークのクライアントは、信頼できる CA が発行した電子証明書を信頼して、電子メッセージ（またはファイル）の署名者の ID を検証します。
タイムスタンプ	イベントの発生日時を含む電子署名されたデータ。ADT には、AIR パッケージ内の RFC 3161 準拠のタイムサーバーからのタイムスタンプを含めることができます。タイムスタンプがある場合、AIR はそのタイムスタンプを使用して、署名時に証明書の有効性を立証します。これにより、AIR アプリケーションは、署名された証明書の期限が切れてもインストールできます。
タイムスタンプ局	タイムスタンプを発行する局。AIR でタイムスタンプを認識するために、タイムスタンプは RFC 3161 に準拠し、タイムスタンプ署名はインストールマシンの信頼できるルート証明書にチェーン化されている必要があります。

iOS 証明書

Apple が発行するコードサイニング証明書は、Adobe AIR を使用して開発されたアプリケーションなどの iOS アプリケーションの署名に使用されます。テストデバイスにアプリケーションをインストールするには、Apple 開発用証明書を使用して署名を適用する必要があります。完成したアプリケーションを配布するには、配布用証明書を使用して署名を適用する必要があります。

アプリケーションに署名するには、ADT がコードサイニング証明書と関連する秘密キーの両方にアクセスできる必要があります。証明書ファイル自体には秘密キーは含まれません。証明書と秘密キーの両方を含む Personal Information Exchange ファイル（.p12 または .pfx）の形式でキーストアを作成する必要があります。193 ページの「P12 キーストアファイルへの開発用証明書の変換」を参照してください。

証明書署名要求の生成

開発用証明書を取得するには、証明書署名要求を生成して、Apple iOS Provisioning Portal に送信する必要があります。

証明書署名要求の処理によって、公開キーと秘密キーのペアが生成されます。秘密キーはユーザーのコンピューター上に残ります。証明機関の役割を担う Apple に、公開キーとユーザーの識別情報を含む署名要求を送信します。Apple は独自の World Wide Developer Relations 証明書を使用して、ユーザーの証明書に署名します。

Mac OS での証明書署名要求の生成

Mac OS では、キーチェーンアクセスアプリケーションを使用して、コード署名要求を作成できます。キーチェーンアクセスアプリケーションは、アプリケーションディレクトリのユーティリティサブディレクトリあります。証明書署名要求の生成方法の説明については、Apple iOS Provisioning Portal を参照してください。

Windows での証明書署名要求の生成

Windows での開発者にとっては、Mac コンピューターで iPhone 開発用証明書を入手するほうが簡単です。しかし、Windows コンピューターで証明書を入手することもできます。最初に、OpenSSL を使用して証明書の署名リクエスト（CSR ファイル）を作成します。

1. Windows コンピューターに OpenSSL をインストールします（http://www.openssl.org/related/binaries.html に移動します）。

   また、Visual C++ 2008 再頒布可能ファイルもインストールする必要があります。このファイルは Open SSL のダウンロードページにリストされています（コンピューターに Visual C++ をインストールする必要はありません）。

2. Windows コマンドセッションを開き、OpenSSL の bin ディレクトリ（c:¥OpenSSL¥bin¥ など）に移動します。

3. コマンドラインで次のように入力して秘密鍵を作成します。

   ```
   openssl genrsa -out mykey.key 2048
   ```

   この秘密鍵ファイルを保存します。このファイルは後で使用します。

   OpenSSL を使用するときは、エラーメッセージを無視しないようにしてください。OpenSSL でエラーメッセージが生成されても、引き続きファイルが出力される場合があります。ただし、それらのファイルは使用できません。エラーが表示されたら、シンタックスを確認して、コマンドを再実行してください。

4. コマンドラインで次のように入力して CSR ファイルを作成します。

   ```
   openssl req -new -key mykey.key -out CertificateSigningRequest.certSigningRequest  -subj
   "/emailAddress=yourAddress@example.com, CN=John Doe, C=US"
   ```

   電子メールアドレス、CN（証明書名）、C（国）を、ユーザー独自の値に置き換えます。

5. iPhone デベロッパーサイトで、Apple に CSR ファイルをアップロードします（「iPhone 開発用証明書の申請およびプロビジョニングプロファイルの作成」を参照してください）。

P12 キーストアファイルへの開発用証明書の変換

P12 キーストアファイルを作成するには、Apple 開発用証明書および関連する秘密キーを、単一のファイルに組み合わせる必要があります。キーストアファイルの作成手順は、元の証明書署名要求の生成方法および秘密キーの保存先によって異なります。

Mac OS での iPhone 開発用証明書の P12 ファイルへの変換

Apple iPhone 証明書を Apple からダウンロードしたら、P12 キーストア形式に書き出します。Mac® OS でこれを行う手順は次のとおりです。

1. アプリケーション / ユーティリティフォルダー内にあるキーチェーンアクセスアプリケーションを開きます。
2. 証明書をキーチェーンにまだ追加していない場合は、ファイル／読み込みを選択します。次に、Apple から取得した証明書ファイル（.cer ファイル）に移動します。
3. キーチェーンアクセスの「分類」で「鍵」を選択します。
4. 使用している iPhone 開発用証明書に関連付けられている秘密鍵を選択します。

   秘密鍵は、ペアになっている iPhone Developer: <First Name> <Last Name> 公開証明書によって識別されます。
5. iPhone Developer 証明書を Command キーを押しながらクリックし、「**iPhone Developer: Name...**」の書き出しを選択します。
6. Personal Information Exchange（.p12）ファイル形式でキーストアを保存します。
7. パスワードを作成するようメッセージが表示されます。このパスワードは、キーストアを使用してアプリケーションを署名するときや、キーストア内のキーと証明書を他のキーストアに転送するときに使用します。

Windows における P12 ファイルへの Apple 開発用証明書の変換

AIR for iOS アプリケーションを開発するには、P12 証明書ファイルを使用する必要があります。この証明書は、Apple から入手する Apple iPhone 開発用証明書ファイルに基づいて生成します。

1. Apple から入手した開発用証明書ファイルを PEM 証明書ファイルに変換します。OpenSSL の bin ディレクトリから、次のコマンドラインステートメントを実行します。

   ```
   openssl x509 -in developer_identity.cer -inform DER -out developer_identity.pem -outform PEM
   ```
2. Mac コンピューターで、キーチェーンから秘密鍵を使用している場合は、秘密鍵を PEM キーに変換します。

   ```
   openssl pkcs12 -nocerts -in mykey.p12 -out mykey.pem
   ```
3. これで、キーと PEM 版の iPhone 開発用証明書に基づいて、有効な P12 ファイルを生成できます。

   ```
   openssl pkcs12 -export -inkey mykey.key -in developer_identity.pem -out iphone_dev.p12
   ```

   Mac OS キーチェーンからキーを使用している場合は、前の手順で生成した PEM 版を使用します。それ以外の場合は、前に（Windows で）作成した OpenSSL キーを使用します。

ADT を使用した未署名の AIR 中間ファイルの作成

-prepare コマンドを使用すると、未署名の AIR 中間ファイルを作成できます。有効な AIR インストールファイルを生成するには、ADT の -sign コマンドで AIR 中間ファイルに署名する必要があります。

-prepare コマンドは、-package コマンドと同じフラグおよびパラメーターを受け取ります（署名オプションを除く）。唯一の違いは、出力ファイルが未署名である点です。中間ファイルは、airi というファイル名拡張子で生成されます。

AIR 中間ファイルに署名するには、ADT の -sign コマンドを使用します（169 ページの「ADT prepare コマンド」を参照）。

ADT -prepare コマンドの例

```
adt -prepare unsignedMyApp.airi myApp.xml myApp.swf components.swc
```

ADT を使用した AIR 中間ファイルへの署名

ADT で AIR 中間ファイルに署名するには、-sign コマンドを使用します。sign コマンドは、AIR 中間ファイル（拡張子 airi）に対してのみ有効です。AIR ファイルに 2 回署名することはできません。

AIR 中間ファイルを作成するには、adt -prepare コマンドを使用します（169 ページの「ADT prepare コマンド」を参照）。

AIRI ファイルへの署名

❖ ADT の -sign コマンドを次のシンタックスで使用します。

```
adt -sign SIGNING_OPTIONS airi_file air_file
```

SIGNING_OPTIONS この署名オプションは、AIR ファイルへの署名に使用する秘密キーと証明書を指定します。これらのオプションについては、176 ページの「ADT コード署名のオプション」で説明しています。

airi_file 署名する未署名の AIR 中間ファイルのパスです。

air_file 作成する AIR ファイルの名前です。

ADT -sign コマンドの例

```
adt -sign -storetype pkcs12 -keystore cert.p12 unsignedMyApp.airi myApp.air
```

詳しくは、169 ページの「ADT sign コマンド」を参照してください。

AIR アプリケーションのアップデートバージョンの署名

既存の AIR アプリケーションのアップデートバージョンを作成するたびに、アップデートされたアプリケーションに署名します。最良のケースでは、以前のバージョンへの署名に使用した証明書で、アップデートバージョンに署名できます。その場合、署名の方法は、最初にアプリケーションに署名したときと全く同様です。

以前のバージョンのアプリケーションへの署名で使用した証明書が期限切れになり、その証明書を更新したか置き換えた場合は、その更新後の証明書または新規（置き換え後）の証明書を使用して、アップデートバージョンに署名できます。そのためには、新しい証明書を使用してアプリケーションに署名し、さらに元の証明書を使用して移行署名を適用します。移行署名によって、元の証明書所有者がアップデートを発行したことが検証されます。

移行署名の適用前に、次のポイントについて考慮してください。

- 移行署名を適用するには、元の証明書がまだ有効であるか、有効期限が切れてから 365 日以内であることが必要です。この期間は「猶予期間」と呼ばれ、期間の長さは将来変更される可能性があります。

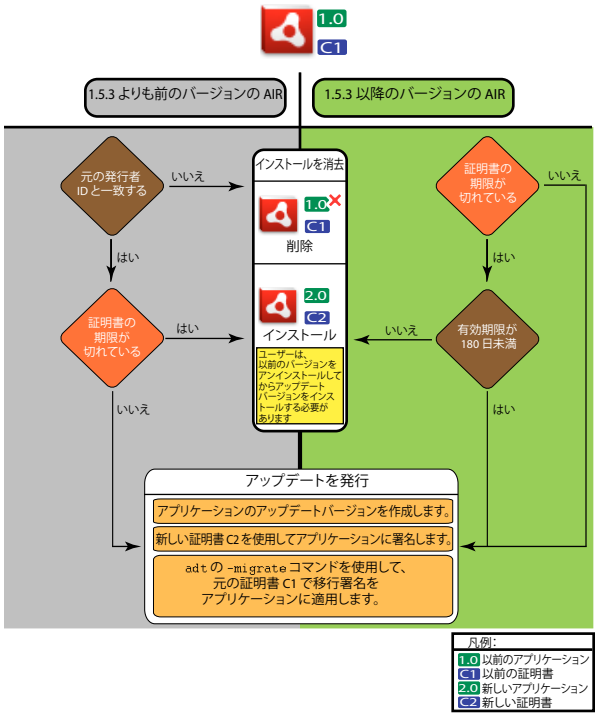
  注意：AIR 2.6 以前では、猶予期間は 180 日です。

- 証明書の期限が切れており、かつ 365 日の猶予期間が過ぎた後は、移行署名を適用することはできません。その場合、アップデートバージョンをインストールするためには、既存のバージョンをアンインストールしておく必要があります。
- 365 日の猶予期間が適用されるのは、アプリケーション記述子の名前空間に AIR のバージョン 1.5.3 以上が指定されているアプリケーションのみです。

重要：期限の切れた証明書の移行署名を使用してアップデートに署名する方法は、一時的な解決策です。包括的な解決策を実施するために、アプリケーションアップデートのデプロイを管理する署名ワークフローを標準化してください。例えば、最新の証明書を使用して各アップデートに署名し、（適用できる場合は）以前のアップデートへの署名に使用した証明書を使用して移行証明書を適用します。その後、各アップデートを、ユーザーがアプリケーションをダウンロードするための個別の URL にアップロードします。詳しくは、256 ページの「アプリケーションアップデートの署名ワークフロー」を参照してください。

以下の表と図に、移行署名のワークフローをまとめます。

シナリオ	元の証明書の状態	開発者の対応	ユーザーの対応
Adobe AIR ランタイムバージョン 1.5.3 以降に基づいたアプリケーション	有効	AIR アプリケーションの最新バージョンを発行します。	対応は必要ありません。 アプリケーションが自動的にアップグレードを実行します。
	期限切れだが 365 日の猶予期間以内	新しい証明書を使用してアプリケーションに署名します。期限切れの証明書を使用して移行署名を適用します。	対応は必要ありません。 アプリケーションが自動的にアップグレードを実行します。
	期限切れ、猶予期間外	AIR アプリケーションのアップデートに移行署名を適用できません。 代わりに、新しい証明書を使用して AIR アプリケーションの異なるバージョンを発行する必要があります。ユーザーは、AIR アプリケーションの既存バージョンのアンインストール後に新しいバージョンをインストールできます。	AIR アプリケーションの現在のバージョンをアンインストールし、最新バージョンをインストールします。
• Adobe AIR ランタイムバージョン 1.5.2 以前に基づいたアプリケーション • アップデートのアプリケーション記述子内にある発行者 ID は以前のバージョンの発行者 ID と一致します。	有効	AIR アプリケーションの最新バージョンを発行します。	対応は必要ありません。 アプリケーションが自動的にアップグレードを実行します。
	期限切れ、猶予期間外	AIR アプリケーションのアップデートに移行署名を適用できません。 代わりに、新しい証明書を使用して AIR アプリケーションの異なるバージョンを発行する必要があります。ユーザーは、AIR アプリケーションの既存バージョンのアンインストール後に新しいバージョンをインストールできます。	AIR アプリケーションの現在のバージョンをアンインストールし、最新バージョンをインストールします。
• Adobe AIR ランタイムバージョン 1.5.2 以前に基づいたアプリケーション • アップデートのアプリケーション記述子内にある発行者 ID は以前のバージョンの発行者 ID と一致**しません**。	任意	有効な証明書を使用して AIR アプリケーションに署名し、その AIR アプリケーションの最新バージョンを発行します。	AIR アプリケーションの現在のバージョンをアンインストールし、最新バージョンをインストールします。



アップデートに対する署名ワークフロー

新しい証明書を使用するための AIR アプリケーションの移行

AIR アプリケーションのアップデート時にそのアプリケーションを最新の証明書に移行するには：

1 アプリケーションのアップデートを作成します。

2 アップデートの AIR ファイルをパッケージ化し、**新しい**証明書で署名します。

3 **元の**証明書を使用して -migrate コマンドで AIR ファイルにもう一度署名します。

-migrate コマンドを使用して署名した AIR ファイルは、古い証明書を使用して署名した以前のバージョンをアップデートするために使用するほか、そのアプリケーションの新しいバージョンをインストールするためにも使用できます。

注意：1.5.3 よりも前のバージョンの AIR で発行されたアプリケーションをアップデートする場合は、アプリケーション記述子に元の発行者 ID を指定します。元の発行者 ID を指定しない場合、アプリケーションのユーザーは、アップデートをインストールする前に、以前のバージョンをアンインストールする必要があります。

ADT の -migrate コマンドを次のシンタックスで使用します。

```
adt -migrate SIGNING_OPTIONS air_file_in air_file_out
```

- **SIGNING_OPTIONS** この署名オプションは、AIR ファイルへの署名に使用する秘密キーと証明書を特定します。このオプションでは、**元の**署名証明書を特定する必要があります。詳しくは、176 ページの「ADT コード署名のオプション」を参照してください。
- **air_file_in 新しい**証明書で署名されたアップデートの AIR ファイルです。
- **air_file_out** 作成する AIR ファイルです。

注意：入力 AIR ファイルと出力 AIR ファイルには、異なるファイル名を使用する必要があります。

以下の例に、ADT を -migrate フラグ付きで呼び出して、移行署名を AIR アプリケーションのアップデートバージョンに適用する方法を示します。

```
adt -migrate -storetype pkcs12 -keystore cert.p12 myAppIn.air myApp.air
```

注意：-migrate コマンドは、AIR 1.1 リリースで ADT に追加されました。

新しい証明書を使用するための AIR ネイティブインストーラーアプリケーションの移行

ネイティブインストーラーとして発行された AIR アプリケーション（例えば、ネイティブ拡張 API を使用するアプリケーション）は、ADT -migrate コマンドを使用して署名することはできません。これは、このアプリケーションは .air ファイルではなく、プラットフォーム固有のネイティブアプリケーションであるからです。そうではなく、ネイティブ拡張として発行された AIR アプリケーションを新しい証明書に移行するには、以下の手順を実行します。

1. アプリケーションのアップデートを作成します。
2. アプリケーション記述ファイル（app.xml）で <supportedProfiles> タグにデスクトッププロファイルと extendedDesktop プロファイルの両方が含まれていることを確認します（または、アプリケーション記述子から <supportedProfiles> タグを削除します）。
3. ADT -package コマンドを使用し、**新しい**証明書を指定して、アップデートアプリケーションを **.air ファイルとして**パッケージ化して署名します。
4. ADT -migrate コマンドを使用し、**元の**証明書を指定して、その .air ファイルに移行証明書を適用します（詳しくは、前述の 197 ページの「新しい証明書を使用するための AIR アプリケーションの移行」を参照してください）。
5. ADT -package コマンドを -target native フラグ付きで使用して、その .air ファイルをネイティブインストーラーにパッケージ化します。アプリケーションは既に署名済みであるので、この手順では署名証明書は指定しません。

以下の例に、このプロセスの手順 3 ～ 5 について示します。このコードは ADT -package コマンドを呼び出し、ADT -migrate コマンドを呼び出し、さらにもう一度 ADT -package コマンドを呼び出して、AIR アプリケーションのアップデートバージョンをネイティブインストーラーとしてパッケージ化します。

```
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
```

新しい証明書を使用するための、ネイティブ拡張を使用する AIR アプリケーションの移行

ネイティブ拡張を使用する AIR アプリケーションに、ADT -migrate コマンドを使用して署名することはできません。また、AIR ネイティブインストーラーアプリケーションの移行手順に従って移行することもできません。AIR ネイティブインストーラーアプリケーションを中間 .air ファイルとして発行することができないからです。そうではなく、ネイティブ拡張を使用する AIR アプリケーションを新しい証明書に移行するには、以下の手順を実行します。

1. アプリケーションのアップデートを作成します。
2. ADT -package コマンドを使用して、アップデートネイティブインストーラーをパッケージ化して署名します。**新しい**証明書を使用してアプリケーションをパッケージ化しますが、その際に、-migrate フラグで**元の**証明書を指定します。

次のシンタックスを使用して、ADT -package コマンドを migrate フラグ付きで呼び出します。

```
adt -package AIR_SIGNING_OPTIONS -migrate MIGRATION_SIGNING_OPTIONS -target package_type
NATIVE_SIGNING_OPTIONS output app_descriptor FILE_OPTIONS
```

- **AIR_SIGNING_OPTIONS** この署名オプションは、AIR ファイルへの署名に使用する秘密キーと証明書を指定します。このオプションでは、**新しい**署名証明書を指定します。詳しくは、176 ページの「ADT コード署名のオプション」を参照してください。

- **MIGRATION_SIGNING_OPTIONS** この署名オプションは、AIR ファイルへの署名に使用する秘密キーと証明書を指定します。このオプションでは、**元の**署名証明書を指定する必要があります。詳しくは、176 ページの「ADT コード署名のオプション」を参照してください。
- その他のオプションは、AIR ネイティブインストーラーアプリケーションのパッケージ化で使用するオプションと同じものです。詳しくは、163 ページの「ADT package コマンド」を参照してください。

以下の例では、ADT -package コマンドを -migrate フラグ付きで呼び出して、ネイティブ拡張を使用する AIR アプリケーションのアップデートバージョンをパッケージ化し、移行署名をそのアップデートに適用しています。

```
adt -package -storetype pkcs12 -keystore new_cert.p12 -migrate -storetype pkcs12 -keystore original_cert.p12
-target native myApp.exe myApp.xml myApp.swf
```

注意：-package コマンドの -migrate フラグは、AIR 3.6 以降の ADT で使用できます。

ADT を使用した自己署名入り証明書の作成

自己署名入り証明書を使用して、有効な AIR インストールファイルを作成できます。ただし、自己署名入り証明書によってユーザーに提供されるセキュリティの保証は限られます。自己署名入り証明書の信頼性は検証できません。自己署名した AIR ファイルがインストールされるとき、発行者側の情報は不明としてユーザーに表示されます。ADT で生成された証明書は 5 年間有効です。

自己署名入り証明書で署名した AIR アプリケーションのアップデートを作成する場合、元の AIR ファイルとアップデート AIR ファイルへの署名には同じ証明書を使用する必要があります。ADT で生成される証明書は、同じパラメーターを使用しても常に一意になります。そのため、ADT で生成された証明書を使用してアップデートに自己署名する場合は、元の証明書を安全な場所に保管しておいてください。また、ADT で生成された元の証明書の有効期限が切れると、アップデートした AIR ファイルを作成できなくなります（別の証明書を使用して新しいアプリケーションを発行することはできますが、同じアプリケーションの新しいバージョンは発行できません）。

重要：自己署名入り証明書のこれらの制限から、一般に公開する AIR アプリケーションへの署名には、信頼できる証明機関から発行された商用証明書を使用することを強くお勧めします。

ADT で生成された証明書および関連付けられている秘密キーは、PKCS12 タイプのキーストアファイルに格納されます。指定したパスワードは、キーストアではなくキー自体に設定されます。

証明書生成の例

```
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
```

これらの証明書を使用して AIR ファイルに署名するには、ADT の -package コマンドまたは -prepare コマンドで次の署名オプションを使用します。

```
-storetype pkcs12 -keystore newcert.p12 -storepass 39#wnetx3tl
-storetype pkcs12 -keystore SigningCert.p12 -storepass 39#wnetx3tl
```

注意：Java バージョン 1.5 以上では、PKCS12 証明書ファイルを保護するパスワードに high-ASCII（拡張 ASCII）文字を使用できません。パスワードには正規の ASCII 文字のみを使用してください。

第 14 章：AIR アプリケーション記述ファイル

すべての AIR アプリケーションにはアプリケーション記述ファイルが必要です。アプリケーション記述ファイルは、アプリケーションの基本プロパティを定義する XML ドキュメントです。

AIR をサポートする多くの開発環境では、プロジェクトを作成するとアプリケーション記述子が自動的に生成されます。生成されない場合は、記述子ファイルを手動で作成する必要があります。AIR SDK および Flex SDK の samples ディレクトリには、サンプルの記述子ファイル descriptor-sample.xml が用意されています。

アプリケーション記述ファイルには任意の名前を付けることができます。アプリケーションをパッケージ化すると、アプリケーション記述ファイルは application.xml という名前に変更されて、AIR パッケージ内の特別なディレクトリに格納されます。

アプリケーション記述子の例

次のアプリケーション記述ドキュメントでは、ほとんどの AIR アプリケーションで使用される基本的なプロパティが設定されます。

```
<?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>
```

SWF ファイルではなく HTML ファイルをアプリケーションのルートコンテンツとして使用する場合は、<content> エレメントのみが次のように異なります。

```
<content>
    HelloWorld.html
</content>
```

アプリケーション記述子の変更点

AIR アプリケーション記述子は、次の AIR リリースで変更されました。

AIR 1.1 アプリケーション記述子の変更点

アプリケーションの name エレメントと description エレメントを、text エレメントを使用してローカライズできるようになりました。

AIR 1.5 アプリケーション記述子の変更点

contentType は fileType の必須の子になりました。

AIR 1.5.3 アプリケーション記述子の変更点

アプリケーションで発行者の ID 値を指定できる publisherID エレメントが追加されました。

AIR 2.0 アプリケーション記述子の変更点

次のエレメントが追加されました。

- aspectRatio
- autoOrients
- fullScreen
- image29x29（imageNxN を参照）
- image57x57
- image72x72
- image512x512
- iPhone
- renderMode
- supportedProfiles

AIR 2.5 アプリケーション記述子の変更点

次のエレメントが削除されました。version

次のエレメントが追加されました。

- android
- extensionID
- extensions
- image36x36（imageNxN を参照）
- manifestAdditions
- versionLabel
- versionNumber

AIR 2.6 アプリケーション記述子の変更点

次のエレメントが追加されました。

- image114x114（imageNxN を参照）

- requestedDisplayResolution
- softKeyboardBehavior

AIR 3.0 アプリケーション記述子の変更点

次のエレメントが追加されました。

- colorDepth
- renderMode の有効な値として、direct
- renderMode はデスクトッププラットフォームで無視されない
- Android <uses-sdk> エレメントを指定可能（以前のバージョンでは指定不可）

AIR 3.1 アプリケーション記述子の変更点

次のエレメントが追加されました。

- 214 ページの「Entitlements」

AIR 3.2 アプリケーション記述子の変更点

次のエレメントが追加されました。

- depthAndStencil
- supportedLanguages

AIR 3.3 アプリケーション記述子の変更点

次のエレメントが追加されました。

- aspectRatio に ANY オプションを追加

AIR 3.4 アプリケーション記述子の変更点

次のエレメントが追加されました。

- image50x50（222 ページの「imageNxN」を参照）
- image58x58（222 ページの「imageNxN」を参照）
- image100x100（222 ページの「imageNxN」を参照）
- image1024x1024（222 ページの「imageNxN」を参照）

AIR 3.6 アプリケーション記述子の変更点

次のエレメントが追加されました。

- 226 ページの「iPhone」エレメントの 232 ページの「requestedDisplayResolution」に、excludeDevices 属性が追加されました。この属性では、高解像度または標準解像度を使用する iOS ターゲットを指定できます。
- 223 ページの「initialWindow」の新規の 232 ページの「requestedDisplayResolution」エレメントでは、Mac などの高解像度ディスプレイを搭載したデスクトッププラットフォームで、高解像度を使用するか標準解像度を使用するかを指定します。

AIR 3.7 アプリケーション記述子の変更点

次のエレメントが追加されました。

- 226 ページの「iPhone」エレメントは 215 ページの「externalSwfs」エレメントを提供するようになりました。これにより、ランタイムでロードされる SWF のリストを指定できます。
- 226 ページの「iPhone」エレメントは、219 ページの「forceCPURenderModeForDevices」エレメントを提供するようになりました。これにより、指定したデバイスのセットに CPU レンダリングモードを強制的に適用できます。

アプリケーション記述ファイルの構造

アプリケーション記述ファイルは次の構造を持つ XML ドキュメントです。

```
<application xmlns="http://ns.adobe.com/air/application/3.0">
    <allowBrowserInvocation>...<allowBrowserInvocation>
    <android>
        <colorDepth>...</colorDepth>
        <manifestAdditions
                <manifest>...</manifest>
            ]]>
        </manifestAdditions
    </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>
    <234 ページの「supportedLanguages」>...</234 ページの「supportedLanguages」>
    <supportedProfiles>...</supportedProfiles>
    <versionNumber>...</versionNumber>
    <versionLabel>...</versionLabel>
</application>
```

AIR アプリケーション記述エレメント

以下のエレメントの辞書では、AIR アプリケーション記述ファイルの有効なエレメントについて説明します。

allowBrowserInvocation

Adobe AIR 1.0 以降 - 省略可能

AIR ブラウザー API を使用して、アプリケーションを検出および起動できます。

この値を true に設定する場合は、セキュリティ上の影響を考慮してください。セキュリティ上の影響については、「ブラウザーからの AIR アプリケーションの呼び出し」（ActionScript 開発者用）および「ブラウザーからの AIR アプリケーションの呼び出し」（HTML 開発者用）で説明しています。

詳しくは、252 ページの「インストール済み AIR アプリケーションのブラウザーからの起動」を参照してください。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

true または false（デフォルト）

例

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

android

Adobe AIR 2.5 以降 - 省略可能

エレメントを Android マニフェストファイルに追加できます。AIR によって、各 APK パッケージの AndroidManifest.xml ファイルが作成されます。AIR アプリケーション記述子で android エレメントを使用して新規アイテムを追加できます。Android 以外のすべてのプラットフォームでは無視されます。

親エレメント：205 ページの「application」

子エレメント：

- 209 ページの「colorDepth」
- 227 ページの「manifestAdditions」

コンテンツ
Android アプリケーションマニフェストに追加する Android 固有のプロパティを定義するエレメント。

例

```
<android>
    <manifestAdditions>
        ...
    </manifestAdditions>
</android>
```

関連項目

75 ページの「Android の設定」

AndroidManifest.xml ファイル

application

Adobe AIR 1.0 以降 — 必須

AIR アプリケーション記述ドキュメントのルートエレメント。

親エレメント：なし

子エレメント：

- 204 ページの「allowBrowserInvocation」
- 205 ページの「android」
- 211 ページの「copyright」
- 211 ページの「customUpdateUI」
- 212 ページの「description」
- 215 ページの「extensions」
- 216 ページの「filename」
- 217 ページの「fileTypes」
- 220 ページの「icon」
- 221 ページの「id」
- 223 ページの「initialWindow」

- 225 ページの「installFolder」
- 226 ページの「iPhone」
- 229 ページの「name」
- 231 ページの「programMenuFolder」
- 231 ページの「publisherID」
- 234 ページの「supportedLanguages」
- 234 ページの「supportedProfiles」
- 237 ページの「version」
- 237 ページの「versionLabel」
- 238 ページの「versionNumber」

属性

minimumPatchLevel — このアプリケーションで必須の AIR ランタイムの最小パッチレベル。

xmlns — XML 名前空間属性によって、アプリケーションの必須 AIR ランタイムバージョンが決まります。

この名前空間は、AIR のメジャーリリースごとに異なります（マイナーパッチには影響されません）。名前空間の最後のセグメント（「3.0」など）は、アプリケーションに必要なランタイムバージョンを示します。

AIR メジャーリリースの xmlns 値は次のとおりです。

```
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"
```

SWF ベースのアプリケーションの場合、アプリケーション記述子で指定された AIR ランタイムのバージョンによって、アプリケーションの初期コンテンツとして読み込むことができる SWF の最大バージョンが決定されます。AIR 1.0 または AIR 1.1 を指定するアプリケーションは、AIR 2 ランタイムを使用している場合でも、初期コンテンツとして使用できるのは SWF9（Flash Player 9）ファイルのみです。AIR 1.5（またはそれ以降）を指定するアプリケーションは、初期コンテンツとして SWF9 または SWF10（Flash Player 10）ファイルを使用できます。

SWF バージョンによって、使用できる AIR および Flash Player API のバージョンが決定されます。AIR 1.5 アプリケーションの初期コンテンツとして SWF9 ファイルが使用されている場合、そのアプリケーションは AIR 1.1 および Flash Player 9 API にのみアクセスできます。また、AIR 2.0 または Flash Player 10.1 内の既存の API に対して行われた動作の変更は有効になりません（ただし、API へのセキュリティに関する重要な変更は例外であり、ランタイムの現在または将来のパッチでさかのぼって適用することができます）。

HTML ベースのアプリケーションの場合、アプリケーション記述子で指定されたランタイムのバージョンによって、アプリケーションで使用できる AIR および Flash Player API のバージョンが決定されます。HTML、CSS、および JavaScript の動作は、アプリケーション記述子ではなく、インストールされた AIR ランタイムで使用されている Webkit のバージョンによって決定されます。

AIR アプリケーションが SWF コンテンツを読み込む場合、そのコンテンツで使用できる AIR および Flash Player API のバージョンは、コンテンツの読み込み方法によって異なります。実際のバージョンは、アプリケーション記述子の名前空間によって決定される場合や、読み込むコンテンツのバージョンによって決定される場合、または読み込まれているコンテンツのバージョンによって決定される場合があります。次の表に、読み込み方法に基づいて API バージョンがどのように決定されるかを示します。

コンテンツの読み込み方法	**API バージョンがどのように決定されるか**
初期コンテンツ、SWF ベースのアプリケーション	**loaded** ファイルの SWF バージョン
初期コンテンツ、HTML ベースのアプリケーション	アプリケーション記述子の名前空間
SWF コンテンツによって読み込まれる SWF	**loading** コンテンツのバージョン
<script> タグを使用した HTML コンテンツによって読み込まれる SWF ライブラリ	アプリケーション記述子の名前空間
AIR または Flash Player API（flash.display.Loader など）を使用している HTML コンテンツによって読み込まれる SWF	アプリケーション記述子の名前空間
<object> タグまたは <embed> タグ（または同等の JavaScript API）を使用している HTML コンテンツによって読み込まれる SWF	**loaded** ファイルの SWF バージョン

読み込まれるコンテンツとは異なるバージョンの SWF ファイルを読み込む場合、次の 2 つの問題が発生する場合があります。

- 古いバージョンの SWF によって新しいバージョンの SWF を読み込む場合 — 読み込まれたコンテンツ内の、新しいバージョンの AIR と Flash Player で追加された API への参照が解決されません。
- 新しいバージョンの SWF によって古いバージョンの SWF を読み込む場合 — 新しいバージョンの AIR と Flash Player で追加された API が、読み込まれたコンテンツの予期しない動作をする可能性があります。

コンテンツ

アプリケーションエレメントには、AIR アプリケーションのプロパティを定義する子エレメントが含まれます。

例

```
<?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>
        <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 以降、iOS および Android — 省略可能

アプリケーションの縦横比を指定します。

指定しない場合、アプリケーションはデバイス「本来の」縦横比および向きで開きます。本来の向きは、デバイスによって異なります。通常、電話などの小さな画面のデバイスでは、縦長の縦横比となります。iPad タブレットなどの一部のデバイスでは、アプリケーションは現在の向きで開きます。AIR 3.3 以降では、最初の表示だけでなく、アプリケーション全体にこのエレメントが適用されます。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
portrait、landscape または any

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

autoOrients

Adobe AIR 2.0 以降、iOS および Android — 省略可能

デバイス自体の物理的な向きが変わったときに、アプリケーションのコンテンツの向きを自動的に変えるかどうかを指定します。詳しくは、「ステージの方向」を参照してください。

自動回転を使用する場合は、Stage の align プロパティと scaleMode プロパティを次のように設定することを検討してください。

```
stage.align = StageAlign.TOP_LEFT;
stage.scaleMode = StageScaleMode.NO_SCALE;
```

これらの設定によって、アプリケーションは左上隅を軸に回転できるようになり、また、アプリケーションコンテンツが自動的に拡大／縮小されなくなります。他の拡大／縮小モードを使用すると、回転されるステージのサイズに合うようにコンテンツが調整されますが、同時にコンテンツのクリップ、変形、過度な縮小も行われます。ほとんどの場合、独自にコンテンツの再描画や再レイアウトを実行することで、さらに良い結果が得られます。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true または false（デフォルト）

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

colorDepth

Adobe AIR 3 以降 - 省略可能

16 bit カラーと 32 bit カラーのどちらを使用するかを指定します。

16 bit カラーを使用すると、レンダリングのパフォーマンスを高めることができますが、色の再現性は低下します。AIR 3 よりも前のバージョンの場合、Android では常に 16 bit カラーが使用されます。AIR 3 では、デフォルトで 32 bit カラーが使用されます。

注意：StageVideo クラスを使用するアプリケーションでは、32 bit カラーを使用する必要があります。

親エレメント：205 ページの「android」

子エレメント：なし

コンテンツ
次の値のいずれかを使用します。

- 16 bit
- 32 bit

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

containsVideo

アプリケーションにビデオコンテンツか含まれるかどうかを指定します。

親エレメント：205 ページの「android」

子エレメント：なし

コンテンツ
次の値のいずれかを使用します。

- true
- false

例
```
<android>
    <containsVideo>true</containsVideo>
    <manifestAdditions>...</manifestAdditions>
</android>
```

content

Adobe AIR 1.0 以降 — 必須

content エレメントには、アプリケーションのメインコンテンツファイルの URL を指定します。このファイルは SWF ファイルまたは HTML ファイルのいずれかです。URL は、アプリケーションのインストールフォルダーのルートを基準に指定します（ADL を使用して AIR アプリケーションを実行する場合は、アプリケーション記述ファイルを含むフォルダーを基準とする URL を指定します。ADL の root-dir パラメーターを使用すると、別のルートディレクトリを指定できます）。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
アプリケーションディレクトリへの相対ディレクトリ。content エレメントの値は URL として扱われるので、コンテンツファイル名の文字は、RFC 1738 で定義されたルールに従って URL エンコードする必要があります。例えば、空白文字は、%20 としてエンコードする必要があります。

例
```
<content>TravelPlanner.swf</content>
```

contentType

Adobe AIR 1.0 ～ 1.1 — 省略可能、AIR 1.5 以降 — 必須

contentType は、AIR 1.5 以降では必須です（AIR 1.0 および 1.1 では省略可能でした）。一部のオペレーティングシステムでこのプロパティを使用すると、ファイルを開く最適なファイルを特定できます。この値はファイルコンテンツの MIME タイプである必要があります。Linux では、当該ファイルタイプが既に登録済みで MIME タイプに割り当てられている場合、この値は無視されます。

親エレメント：217 ページの「fileType」

子エレメント：なし

コンテンツ
MIME タイプとサブタイプ。MIME タイプについて詳しくは、RFC2045 を参照してください。

例

```
<contentType>text/plain</contentType>
```

copyright

Adobe AIR 1.0 以降 - 省略可能

AIR アプリケーションの著作権情報です。Mac OS では、インストールしたアプリケーションのバージョン情報ダイアログボックスに著作権情報が表示されます。Mac OS では、アプリケーションの Info.plist ファイルの「NSHumanReadableCopyright」フィールドにも著作権情報が表示されます。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
アプリケーションの著作権情報を含むストリング。

例

```
<copyright>© 2010, Examples, Inc.All rights reserved.</copyright>
```

customUpdateUI

Adobe AIR 1.0 以降 - 省略可能

アプリケーションが独自のアップデートダイアログを表示するかどうかを指定します。false の場合、標準のアップデートダイアログがユーザーに表示されます。組み込みの AIR アップデートシステムを使用できるのは、AIR ファイルとして配布されているアプリケーションのみです。

インストールされているバージョンのアプリケーションで customUpdateUI エレメントが true に設定されているときに、ユーザーが新しいバージョンの AIR ファイルをダブルクリックするか、シームレスインストール機能を使用してアプリケーションのアップデートをインストールすると、インストールされているバージョンのアプリケーションがランタイムによって起動されます。ランタイムでは、デフォルトの AIR アプリケーションインストーラーは起動されません。その後、アプリケーションロジックでアップデート操作の続行方法を決定できます（アップグレードを続行するには、AIR ファイルとインストールされているアプリケーションのアプリケーション ID および発行者 ID が一致している必要があります）。

注意：customUpdateUI のメカニズムが機能するのは、アプリケーションがインストール済みで、ユーザーがアップデートを含む AIR インストールファイルをダブルクリックするか、シームレスインストール機能を使用してアプリケーションのアップデートをインストールする場合だけです。アップデートのダウンロードおよびインストールは、customUpdateUI が true かどうかにかかわらず、独自のアプリケーションロジックとカスタム UI （必要な場合）を使用して行うことができます。

詳しくは、254 ページの「AIR アプリケーションのアップデート」を参照してください。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
true または false（デフォルト）

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

depthAndStencil

Adobe AIR 3.2 以降 - 省略可能

アプリケーションで深度またはステンシルバッファーを使用する必要があることを示します。通常、これらのバッファーは 3D コンテンツを操作する際に使用します。デフォルトでは、このエレメントの値は false であり、深度およびステンシルバッファーは無効化されています。これらのバッファーは、コンテンツの読み込み前のアプリケーション起動時に割り当てられる必要があるので、このエレメントが必要になります。

このエレメントの設定は、Context3D.configureBackBuffer() メソッドの enableDepthAndStencil 引数に渡される値と一致する必要があります。値が一致しない場合は、AIR でエラーが発生します。

このエレメントは、renderMode = direct の場合にのみ適用されます。renderMode が direct でない場合は、ADT により次のようなエラー 118 がスローされます。

```
<depthAndStencil> element unexpected for render mode cpu.  It requires "direct" render mode.
```

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true または false（デフォルト）

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

description

Adobe AIR 1.0 以降 - 省略可能

AIR アプリケーションインストーラーに表示される、アプリケーションの説明です。

1 つのテキストノードを指定した場合（複数の text エレメントを指定しない場合）、システム言語に関係なく、この説明が AIR アプリケーションインストーラーで使用されます。複数のテキストノードを指定した場合、AIR アプリケーションインストーラーでは、オペレーティングシステムのユーザーインターフェイス言語に最も近い説明が使用されます。例えば、アプリケーション記述ファイルの description エレメントの値として en（英語）が含まれているとします。オペレーティングシステムのユーザーインターフェイス言語が en（英語）になっている場合は、AIR アプリケーションインストーラーで en

記述が使用されます。システムのユーザーインターフェイス言語が en-US（アメリカ英語）の場合にも、en 記述が使用されます。ただし、システムのユーザーインターフェイス言語が en-US で、アプリケーション記述ファイルに en-US と en-GB の両方が定義されている場合は、AIR アプリケーションインストーラーで en-US が使用されます。システムのユーザーインターフェイス言語と一致する記述がアプリケーションに定義されていない場合、AIR アプリケーションインストーラーではアプリケーション記述ファイルに定義された最初の description の値が使用されます。

複数言語アプリケーションの開発について詳しくは、289 ページの「AIR アプリケーションのローカライズ」を参照してください。

親エレメント：205 ページの「application」

子エレメント：236 ページの「text」

コンテンツ

AIR 1.0 アプリケーション記述スキーマでは、1 つの単純なテキストノードのみを名前として定義できます。複数の text エレメントを定義することはできません。

AIR 1.1（またはそれ以降）では、複数の言語を description エレメントに指定できます。各テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

例

単純な text ノードを使用する記述：

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

英語、フランス語、スペイン語にローカライズされた text エレメントを使用する記述（AIR 1.1 以降で有効）：

```
<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 以降 — 必須

ファイルタイプ記述は、オペレーティングシステムのユーザーに表示されます。ファイルタイプ記述はローカライズできません。

参照：212 ページの「description」（application エレメントの子として）

親エレメント：217 ページの「fileType」

子エレメント：なし

コンテンツ

ファイルコンテンツを説明するストリング。

例

```
<description>PNG image</description>
```

embedFonts

AIR アプリケーション内の StageText でカスタムフォントを使用できるようにします。このエレメントはオプションです。

親エレメント：205 ページの「application」

子エレメント：218 ページの「font」

コンテンツ
embedFonts エレメントには、font エレメントをいくつでも含めることができます。

例
```
<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 以降 - iOS のみ、省略可能

iOS では、Entitlements というプロパティを使用して、追加のリソースおよび機能へのアクセスをアプリケーションに提供します。Entitlements エレメントを使用して、モバイル iOS アプリケーションでこの情報を指定します。

親エレメント：226 ページの「iPhone」

子エレメント：iOS Entitlements.plist エレメント

コンテンツ
アプリケーションの Entitlements.plist 設定として使用するキーと値のペアを指定する、子エレメントが含まれます。Entitlements エレメントのコンテンツは CDATA ブロックで囲む必要があります。詳しくは、iOS Developer Library の「Entitlement Key Reference」を参照してください。

例
```
<iphone>
...
   <Entitlements>
      <![CDATA[
         <key>aps-environment</key>
         <string>development</string>
      ]]>
   </Entitlements>
</iphone>
```

extension

Adobe AIR 1.0 以降 — 必須

ファイルタイプの拡張ストリング。

親エレメント：217 ページの「fileType」

子エレメント：なし

コンテンツ
ファイル拡張子文字を指定するストリング（ドット（.）を除きます）。

例
```
<extension>png</extension>
```

extensionID

Adobe AIR 2.5 およびそれ以降

アプリケーションで使用される ActionScript 拡張の ID を指定します。ID は拡張記述ドキュメントに定義されます。

親エレメント：215 ページの「extensions」

子エレメント：なし

コンテンツ
ActionScript 拡張 ID を示すストリング。

例
```
<extensionID>com.example.extendedFeature</extensionID>
```

extensions

Adobe AIR 2.5 以降 - 省略可能

アプリケーションで使用される ActionScript 拡張を指定します。

親エレメント：205 ページの「application」

子エレメント：215 ページの「extensionID」

コンテンツ
拡張記述ファイルの ActionScript 拡張 ID を含む子の extensionID エレメント。

例
```
<extensions>
    <extensionID>extension.first</extensionID>
    <extensionID>extension.next</extensionID>
    <extensionID>extension.last</extensionID>
</extensions>
```

externalSwfs

Adobe AIR 3.7 以降、iOS のみ — 省略可能

リモートでホストするために、ADT によって設定される SWF のリストを含むテキストファイルの名前を指定します。アプリケーションによって使用される SWF のサブセットをパッケージ化し、Loader.load() メソッドを使用してランタイムでその他の（アセットのみの）外部 SWF をロードすることで、最初のアプリケーションのダウンロードサイズを最小限に抑えることができます。この機能を使用するには、ADT がアセットのみを含む SWF ファイルを残し、外部でロードされた

SWF ファイルからメインアプリケーションの SWF にすべての ActionScript バイトコード（ABC）を移動するアプリケーションをパッケージ化する必要があります。これは、アプリケーションがインストールされた後にコードをダウンロードすることを禁じる Apple Store のルールに従うためです。

詳しくは、85 ページの「外部の、アセットのみの SWF をロードすることでダウンロードサイズを最小限にする」を参照してください。

親エレメント：226 ページの「iPhone」、223 ページの「initialWindow」

子エレメント：なし

コンテンツ
リモートでホストされる SWF の行区切りリストを含むテキストファイルの名前

属性：
なし。

例
iOS：

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

filename

Adobe AIR 1.0 以降 — 必須

アプリケーションのインストール時に、アプリケーションのファイル名（拡張子なし）として使用するストリングです。ランタイムでは、アプリケーションファイルから AIR アプリケーションを起動します。name の値を指定しない場合、filename がインストールフォルダーの名前としても使用されます。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
filename プロパティには、任意の Unicode（UTF-8）文字を含めることができますが、次の文字は使用できません。これらの文字は、各種のファイルシステムで使用が禁止されています。

文字	16 進コード
各種文字	0x00 ～ x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E

文字	16 進コード
?	x3F
\	x5C
|	x7C

filename の値の末尾にピリオドを付けることはできません。

例

```
<filename>MyApplication</filename>
```

fileType

Adobe AIR 1.0 以降 - 省略可能

アプリケーションが登録できる単一のファイルタイプを指定します。

親エレメント：217 ページの「fileTypes」

子エレメント：

- 211 ページの「contentType」
- 213 ページの「description」
- 214 ページの「extension」
- 220 ページの「icon」
- 230 ページの「name」

コンテンツ

ファイルタイプを指定するエレメント。

例

```
<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</imge48x48>
    <icon>
</fileType>
```

fileTypes

Adobe AIR 1.0 以降 - 省略可能

fileTypes エレメントを使用すると、AIR アプリケーションを関連付けることができるファイルの種類を宣言できます。

AIR アプリケーションをインストールすると、宣言済みのファイルの種類がすべてオペレーティングシステムに登録されます。これらのファイルの種類が別のアプリケーションに関連付けられていない場合は、AIR アプリケーションに関連付けられます。ファイルの種類と他のアプリケーションとの関連付けをオーバーライドするには、実行時に NativeApplication.setAsDefaultApplication() メソッドを使用します（ユーザーの権限で実行することをお勧めします）。

注意：ランタイムメソッドで管理できるのは、アプリケーション記述子で宣言されたファイルの種類の関連付けだけです。

fileTypes プロパティはオプションです。

親エレメント：205 ページの「application」

子エレメント：217 ページの「fileType」

コンテンツ

fileTypes エレメントには、fileType エレメントをいくつでも含めることができます。

例

```
<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

AIR アプリケーション内で使用できる単一のカスタムフォントを表します。

親エレメント：213 ページの「embedFonts」

子エレメント：218 ページの「fontName」、219 ページの「fontPath」

コンテンツ

カスタムフォントの名前とパスを指定するエレメント。

例

```
<font>
   <fontPath>ttf/space age.ttf</fontPath>
   <fontName>space age</fontName>
</font>
```

fontName

カスタムフォントの名前を指定します。

親エレメント：218 ページの「font」

子エレメント：なし

コンテンツ

StageText.fontFamily に指定されるカスタムフォントの名前。

例

```
<fontName>space age</fontName>
```

fontPath

カスタムフォントファイルの場所を示します。

親エレメント：218 ページの「font」

子エレメント：なし

コンテンツ
カスタムフォントファイルのパス（ソースを基準として指定）。

例

```
<fontPath>ttf/space age.ttf</fontPath>
```

forceCPURenderModeForDevices

Adobe AIR 3.7 以降、iOS のみ — 省略可能

指定したデバイスセットに対して CPU レンダリングモードを強制的に適用します。この機能を使用すると、効率的にその他の iOS デバイスに対して選択的に GPU レンダリングモードを有効にすることができます。

このタグは iPhone タグの子として追加し、デバイスのモデル名のリストをスペース区切りで指定します。有効なデバイスモデル名には以下が含まれます。

iPad 1,1	iPhone1,1	iPod1,1
iPad 2,1	iPhone1,2	iPod2,1
iPad 2,2	iPhone2,1	iPod3,3
iPad2,3	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		

親エレメント：226 ページの「iPhone」、223 ページの「initialWindow」

子エレメント：なし

コンテンツ
デバイスモデル名のスペース区切りのリスト

属性：
なし。

例
iOS：

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

fullScreen

Adobe AIR 2.0 以降、iOS および Android — 省略可能

アプリケーションをフルスクリーンモードで起動するかどうかを指定します。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true または false（デフォルト）

例

```
<fullscreen>true</fullscreen>
```

height

Adobe AIR 1.0 以降 - 省略可能

アプリケーション起動時のメインウィンドウの高さ。

高さを設定しない場合、ルート SWF ファイルの設定によって決まります。また、HTML ベースの AIR アプリケーションの場合は、オペレーティングシステムによって決まります。

AIR 2 では、ウィンドウの最高の高さは 2,048 ピクセルから 4,096 ピクセルに変更されました。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
最大値が 4,095 の正の整数。

例

```
<height>4095</height>
```

icon

Adobe AIR 1.0 以降 - 省略可能

icon プロパティでは、アプリケーションで使用するアイコンファイルを指定します。アイコンの指定は必須ではありません。icon プロパティを指定しない場合、オペレーティングシステムによってデフォルトのアイコンが表示されます。

アイコンファイルのパスは、アプリケーションのルートディレクトリを基準に指定します。アイコンファイルは PNG 形式である必要があります。次のすべてのアイコンサイズを指定できます。

特定のサイズに対応するエレメントが存在する場合、ファイル内のイメージは指定どおりのサイズになります。一部のサイズしか存在しない場合、アイコンの使用に合わせて、最も近いサイズのイメージがオペレーティングシステムによって拡大または縮小されます。

注意：指定したアイコンは、AIR パッケージに自動的には追加されません。アプリケーションをパッケージ化する際には、アイコンファイルが適切な相対場所に格納されている必要があります。

最適な結果を得るためには、使用可能なサイズごとにイメージを用意します。また、アイコンが 16 bit および 32 bit のカラーモードで適切に表示されることを確認します。

親エレメント：205 ページの「application」

子エレメント：222 ページの「imageNxN」

コンテンツ

目的の各アイコンサイズの imageNxN エレメント。

例

```
<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 以降 — 必須

アプリケーション ID と呼ばれる、アプリケーションの識別ストリングです。リバース DNS スタイル ID がよく使用されますが、必須ではありません。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

ID 値には、次の文字のみを使用できます。

- 0 ～ 9
- a ～ z
- A ～ Z
- .（ドット）
- -（ハイフン）

値の長さは 1 ～ 212 文字です。このエレメントは必須です。

例

```
<id>org.example.application</id>
```

imageNxN

Adobe AIR 1.0 以降 - 省略可能

アイコンへのパスを定義します（アプリケーションディレクトリに対して相対的なパス）。

次のアイコン画像を使用し、それぞれに異なるアイコンサイズを指定できます

- image16x16
- image29x29（AIR 2 以降）
- image32x32
- image36x36（AIR 2.5 以降）
- image48x48
- image50x50（AIR 3.4 以降）
- image57x57（AIR 2 以降）
- image58x58（AIR 3.4 以降）
- image72x72（AIR 2 以降）
- image100x100（AIR 3.4 以降）
- image114x114（AIR 2.6 以降）
- image128x128
- image144x144（AIR 3.4 以降）
- image512x512（AIR 2 以降）
- image1024x1024（AIR 3.4 以降）

アイコンは、image エレメントに指定されたとおりのサイズの PNG グラフィックにする必要があります。アイコンファイルはアプリケーションパッケージに含める必要があります。アプリケーション記述ドキュメントで参照されるアイコンは、自動的には取り込まれません。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

アイコンへのファイルパスには、任意の Unicode（UTF-8）文字を含めることができますが、次の文字は使用できません。これらの文字は、各種のファイルシステムで使用が禁止されています。

文字	16 進コード
各種文字	0x00 ～ x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E

文字	16 進コード
?	x3F
\	x5C
|	x7C

例

```
<image32x32>icons/icon32.png</image32x32>
```

InfoAdditions

Adobe AIR 1.0 以降 - 省略可能

iOS アプリケーションの追加プロパティを指定できます。

親エレメント：226 ページの「iPhone」

子エレメント：iOS Info.plist エレメント

コンテンツ

アプリケーションの Info.plist 設定として使用するキーと値のペアを指定する、子エレメントが含まれます。InfoAdditions エレメントのコンテンツは CDATA ブロックで囲む必要があります。

キーと値のペアおよびそれらを XML で表現する方法については、Apple iPhone Reference Library の「Information Property List Key Reference」を参照してください。

例

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

関連項目

80 ページの「iOS の設定」

initialWindow

Adobe AIR 1.0 以降 — 必須

メインコンテンツファイルとアプリケーションの外観の初期設定を定義します。

親エレメント：205 ページの「application」

子エレメント：次のエレメントのすべてを initialWindow エレメントの子として表示できます。ただし、AIR がプラットフォームでウィンドウをサポートしているかどうかによって、一部のエレメントは無視されます。

エレメント	デスクトップ	モバイル
208 ページの「aspectRatio」	無視	使用
209 ページの「autoOrients」	無視	使用
210 ページの「content」	使用	使用
212 ページの「depthAndStencil」	使用	使用
220 ページの「fullScreen」	無視	使用
220 ページの「height」	使用	無視
228 ページの「maximizable」	使用	無視
228 ページの「maxSize」	使用	無視
229 ページの「minimizable」	使用	無視
229 ページの「minSize」	使用	無視
231 ページの「renderMode」	使用（AIR 3.0 以降）	使用
232 ページの「requestedDisplayResolution」	使用（AIR 3.6 以降）	無視
233 ページの「resizable」	使用	無視
233 ページの「softKeyboardBehavior」	無視	使用
235 ページの「systemChrome」	使用	無視
236 ページの「title」	使用	無視
237 ページの「transparent」	使用	無視
238 ページの「visible」	使用	無視
239 ページの「width」	使用	無視
239 ページの「x」	使用	無視
239 ページの「y」	使用	無視

コンテンツ
アプリケーションの外観と動作を定義する子エレメント。

例

```
<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 以降 - 省略可能

デフォルトのインストールディレクトリのサブディレクトリを指定します。

Windows では、デフォルトのインストールサブディレクトリは Program Files ディレクトリです。Mac OS では、/ アプリケーションディレクトリです。Linux では、/opt/ です。例えば、installFolder プロパティが「Acme」に設定されていて、アプリケーションの名前が「ExampleApp」である場合、アプリケーションは、Windows では C:¥Program Files¥Acme¥ExampleApp に、Ma OS では / アプリケーション /Acme Example.app に、Linux では /opt/Acme/ExampleApp にそれぞれインストールされます。

installFolder プロパティはオプションです。installFolder プロパティを指定しない場合、アプリケーションは、name プロパティに基づいて、デフォルトのインストールディレクトリのサブディレクトリにインストールされます。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

installFolder プロパティには、任意の Unicode（UTF-8）文字を含めることができます。ただし、各種のファイルシステムでフォルダー名として使用できない文字は除きます（使用できない文字の一覧については、filename プロパティを参照してください）。

ネストされたサブディレクトリを指定する場合は、スラッシュ（/）をディレクトリの区切り文字として使用します。

例

```
<installFolder>utilities/toolA</installFolder>
```

iPhone

Adobe AIR 2.0、iOS のみ — 省略可能

iOS 固有のアプリケーションプロパティを定義します。

親エレメント：205 ページの「application」

子エレメント：

- 214 ページの「Entitlements」
- 215 ページの「externalSwfs」
- 219 ページの「forceCPURenderModeForDevices」
- 223 ページの「InfoAdditions」
- 232 ページの「requestedDisplayResolution」

関連項目

80 ページの「iOS の設定」

manifest

Adobe AIR 2.5 以降、Android のみ — 省略可能

アプリケーションの Android マニフェストファイルに追加する情報を指定します。

親エレメント：227 ページの「manifestAdditions」

子エレメント：Android SDK によって定義

コンテンツ

技術的には、manifest エレメントは AIR アプリケーションの記述スキーマの一部ではありません。Android マニフェスト XML ドキュメントのルートです。manifest エレメントに配置するコンテンツは、AndroidManifest.xml スキーマに従う必要があります。AIR ツールで APK ファイルを生成すると、manifest エレメント内の情報が、アプリケーションの生成された AndroidManifest.xml 内の対応する部分にコピーされます。

AIR で直接サポートされる SDK バージョンよりも新しいバージョンでのみ利用可能な Android マニフェストの値を指定する場合は、アプリケーションをパッケージ化する際に、ADT に -platformsdk フラグを設定する必要があります。このフラグを、追加する値をサポートしているバージョンの Android SDK へのファイルシステムパスに設定してください。

manifest エレメント自体を、AIR アプリケーション記述子内の CDATA ブロックに含める必要があります。

例

```
<![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>
]]>
```

関連項目

75 ページの「Android の設定」

AndroidManifest.xml ファイル

manifestAdditions

Adobe AIR 2.5 以降、Android のみ

Android マニフェストファイルに追加する情報を指定します。

各 Android アプリケーションには、基本的なアプリケーションプロパティを定義するマニフェストが含まれています。Android マニフェストは、AIR アプリケーション記述子と概念が似ています。AIR for Android アプリケーションには、アプリケーション記述子と自動的に生成された Android マニフェストファイルの両方があります。AIR for Android をパッケージ化すると、この manifestAdditions エレメントの情報は、Android マニフェストドキュメントの対応する部分に追加されます。

親エレメント：205 ページの「android」

子エレメント：226 ページの「manifest」

コンテンツ

manifestAdditions エレメントの情報は、AndroidManifest XML ドキュメントの対応する部分に追加されます。

AIR は、生成された Android マニフェストドキュメントに複数のマニフェストエントリを設定して、アプリケーションとランタイム機能が正しく動作するようにします。次の設定をオーバーライドすることはできません。

manifest エレメントの次の属性を設定することはできません。

- package
- android:versionCode
- android:versionName

メインアクティビティエレメントの次の属性を設定することはできません。

- android:label
- android:icon

application エレメントの次の属性を設定することはできません。

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

例

```
<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>
```

関連項目

75 ページの「Android の設定」

AndroidManifest.xml ファイル

maximizable

Adobe AIR 1.0 以降 - 省略可能

ウィンドウを最大化できるかどうかを指定します。

注意：Mac OS X などのオペレーティングシステムでは、ウィンドウの最大化がサイズ変更操作として行われるため、ウィンドウのズームまたはサイズ変更を防ぐために、maximizable と resizable の両方を false に設定する必要があります。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ

true（デフォルト）または false

例

```
<maximizable>false</maximizable>
```

maxSize

Adobe AIR 1.0 以降 - 省略可能

ウィンドウの最大サイズ。最大サイズを設定しない場合、オペレーティングシステムの制限に従います。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ

空白で区切られた、最大の幅と高さを表す 2 つの整数。

注意：AIR がサポートする最大ウィンドウサイズは、AIR 2 で 2,048 × 2,048 ピクセルから 4,096 × 4,096 ピクセルに拡大されました（画面の座標はゼロベースなので、幅または高さに使用できる最大値は 4,095 です）。

例

```
<maxSize>1024 360</maxSize>
```

minimizable

Adobe AIR 1.0 以降 - 省略可能

ウィンドウを最小化できるかどうかを指定します。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true（デフォルト）または false

例
```
<minimizable>false</minimizable>
```

minSize

Adobe AIR 1.0 以降 - 省略可能

ウィンドウに使用できる最小サイズを指定します。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
空白で区切られた、最小の幅と高さを表す 2 つの整数。オペレーティングシステムによる最大サイズ制限の方が、アプリケーション記述子で設定した値よりも優先されます。

例
```
<minSize>120 60</minSize>
```

name

Adobe AIR 1.0 以降 - 省略可能

AIR アプリケーションインストーラーで表示されるアプリケーションタイトル。

name エレメントが指定されていない場合、AIR アプリケーションインストーラーではアプリケーション名として filename が表示されます。

親エレメント：205 ページの「application」

子エレメント：236 ページの「text」

コンテンツ
1 つのテキストノードを指定した場合（複数の <text> エレメントを指定しない場合）、システム言語に関係なく、この名前が AIR アプリケーションインストーラーで使用されます。

AIR 1.0 アプリケーション記述スキーマでは、1 つの単純なテキストノードのみを名前として定義できます。複数の text エレメントを定義することはできません。AIR 1.1（またはそれ以降）では、複数の言語を name エレメントに指定できます。

各テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

AIR アプリケーションインストーラーでは、ユーザーのオペレーティングシステムのユーザーインターフェイス言語と最も近い名前が使用されます。例えば、アプリケーション記述ファイルの name エレメントの値として en（英語）が含まれているとします。オペレーティングシステムのユーザーインターフェイス言語が en（英語）になっている場合は、AIR アプリケーションインストーラーで名前 en が使用されます。システムのユーザーインターフェイス言語が en-US（アメリカ英語）の場合にも名前 en が使用されます。ただし、ユーザーインターフェイス言語が en-US で、アプリケーション記述ファイルに en-US と en-GB の両方が定義されている場合は、AIR アプリケーションインストーラーで en-US が使用されます。システムのユーザーインターフェイス言語と一致する名前がアプリケーションに定義されていない場合、AIR アプリケーションインストーラーではアプリケーション記述ファイルに定義された最初の name の値が使用されます。

name エレメントは、AIR アプリケーションインストーラーで使用されるアプリケーションのタイトルを定義するだけです。AIR アプリケーションインストーラーは、複数の言語（繁体字中国語、簡体字中国語、チェコ語、オランダ語、英語、フランス語、ドイツ語、イタリア語、日本語、韓国語、ポーランド語、ブラジルポルトガル語、ロシア語、スペイン語、スウェーデン語およびトルコ語）をサポートしています。AIR アプリケーションでは、システムのユーザーインターフェイス言語に基づいて、アプリケーションとタイトルおよび説明以外のテキスト用として、表示された言語が選択されます。この言語の選択は、アプリケーション記述ファイルの設定とは関係なく行われます。

name エレメントでは、実行中のインストール済みアプリケーションの言語は定義しません。複数言語アプリケーションの開発について詳しくは、289 ページの「AIR アプリケーションのローカライズ」を参照してください。

例

次に、簡単な text ノードを使用して名前を定義する例を示します。

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

次の例は AIR 1.1 以降で有効です。<text> エレメントノードを使用して、3 つの言語（英語、フランス語、およびスペイン語）で名前を指定します。

```
<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 以降 — 必須

ファイルタイプの名前を指定します。

親エレメント：217 ページの「fileType」

子エレメント：なし

コンテンツ

ファイルタイプの名前を表す文字列。

例

```
<name>adobe.VideoFile</name>
```

programMenuFolder

Adobe AIR 1.0 以降 - 省略可能

Windows オペレーティングシステムのすべてのプログラムメニューまたは Linux のアプリケーションメニューに表示する、アプリケーションのショートカットの位置を指定します（現在、この設定は他のオペレーティングシステムでは無視されます）。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

programMenuFolder 値には、任意の Unicode（UTF-8）文字を含めることができます。ただし、各種のファイルシステムでフォルダー名として使用できない文字は除きます（使用できない文字の一覧については、filename エレメントを参照してください）。この値の末尾文字としてスラッシュ（/）は使用しないでください。

例

```
<programMenuFolder>Example Company/Sample Application</programMenuFolder>
```

publisherID

Adobe AIR 1.5.3 以降 - 省略可能

AIR バージョン 1.5.2 以前で作成された AIR アプリケーションの更新に使用する発行者 ID を指定します。

アプリケーションのアップデートを作成するときにのみ、発行者 ID を指定します。publisherID エレメントの値は、前のバージョンのアプリケーション用に AIR で生成された発行者 ID と一致する必要があります。インストール済みのアプリケーションの場合、発行者 ID はアプリケーションのインストールフォルダーの META-INF/AIR/publisherid ファイルで見つけることができます。

AIR 1.5.3 以降で作成された新しいアプリケーションでは、発行者 ID を指定しないでください。

詳しくは、187 ページの「AIR 発行者 ID について」を参照してください。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

発行者 ID のストリング。

例

```
<publisherID>B146A943FBD637B68C334022D304CEA226D129B4.1</publisherID>
```

renderMode

Adobe AIR 2.0 以降 - 省略可能

グラフィック処理装置（GPU）のアクセラレーションが現在のコンピューターデバイスでサポートされる場合、そのアクセラレーションを使用するかどうかを指定します。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
次の値のいずれかを使用します。

- auto（デフォルト）— 現在は CPU モードに戻ります。
- cpu — ハードウェアアクセラレーションは使用されません。
- direct — コンポジションのレンダリングは CPU で行われ、ブリッティングには GPU が使用されます。AIR 3 以降で使用できます。

  注意：Flash コンテンツの GPU アクセラレーションを、モバイルプラットフォーム向けの AIR で利用するには、renderMode="gpu" ではなく renderMode="direct"（Stage3D）を使用することをお勧めします。Adobe で公式にサポートおよび推奨している Stage3D ベースのフレームワークは、Starling（2D）および Away3D（3D）です。Stage3D および Starling/Away3D について詳しくは、http://gaming.adobe.com/getstarted/ を参照してください。
- gpu — ハードウェアアクセラレーションが使用可能な場合は使用されます。

  重要：Flex アプリケーションでは GPU レンダリングモードを使用しないでください。

例

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

requestedDisplayResolution

Adobe AIR 2.6 以降（iOS のみ）、Adobe AIR 3.6 以降（Mac OS X）— オプション

アプリケーションが、高解像度画面を搭載するデバイスまたはコンピューター画面で標準解像度を使用するか高解像度を使用するかを指定します。デフォルトの **standard** に設定すると、画面はアプリケーションに対して標準解像度の画面として表示されます。**high** に設定すると、アプリケーションは高解像度の各ピクセルを解決できます。

例えば、640x960 の高解像度 iPhone 画面で、設定を **standard** にした場合、フルスクリーンのステージサイズは 320x480 となり、アプリケーションの各ピクセルは画面の 4 ピクセルを使用してレンダリングされます。設定が **high** の場合は、フルスクリーンのステージサイズは 640x960 になります。

標準解像度の画面を搭載するデバイスでは、どちらの設定が使用されていても、ステージサイズは画面サイズに一致します。

requestedDisplayResolution エレメントが iPhone エレメント内でネストされている場合は、この設定は iOS デバイスに適用されます。その場合、excludeDevices 属性を使用して、この設定が適用されないデバイスを指定できます。

requestedDisplayResolution エレメントが initialWindow エレメント内でネストされている場合は、この設定は高解像度ディスプレイをサポートする MacBook Pro コンピューター上のデスクトップ AIR アプリケーションに適用されます。指定された値は、アプリケーションで使用されるすべてのネイティブウィンドウに適用されます。initialWindow 内での requestedDisplayResolution エレメントのネストは、AIR 3.6 以降でサポートされます。

親エレメント：226 ページの「iPhone」、223 ページの「initialWindow」

子エレメント：なし

コンテンツ
standard（デフォルト）、または **high**。

属性：
excludeDevices — iOS モデル名またはモデル名の接頭辞をスペースで区切ったリスト。この指定によって、開発者は高解像度を使用するデバイスと標準解像度を使用するデバイスを区別して指定できます。この属性は、iOS でのみ使用できます（requestedDisplayResolution エレメントは iPhone エレメント内でネストされます）。excludeDevices 属性は、AIR 3.6 以降で使用できます。

この属性内でモデル名が指定されているすべてのデバイスで、requestedDisplayResolution の値は、この指定された値の逆になります。つまり、requestedDisplayResolution の値が **high** の場合、除外されるデバイスでは標準解像度が使用されます。requestedDisplayResolution の値が **standard** の場合は、除外されるデバイスでは高解像度が使用されます。

この値は、iOS デバイスモデル名かモデル名の接頭辞のいずれかです。例えば、iPad3,1 という値は、Wi-Fi モデルの第 3 世代 iPad を示します（ただし、GSM モデルまたは CDMA モデルの第 3 世代 iPad ではありません）。また、iPad3 という値は任意の第 3 世代 iPad を示します。iOS モデル名の非公式のリストは、iPhone Wiki のモデル名のページにあります。

例
デスクトップ：

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

iOS：

```
<iPhone>
    <requestedDisplayResolution excludeDevices="iPad3 iPad4">high</requestedDisplayResolution>
</iPhone>
```

resizable

Adobe AIR 1.0 以降 - 省略可能

ウィンドウをサイズ変更できるかどうかを指定します。

注意：Mac OS X などのオペレーティングシステムでは、ウィンドウの最大化がサイズ変更操作として行われるため、ウィンドウのズームまたはサイズ変更を防ぐために、maximizable と resizable の両方を false に設定する必要があります。

親エレメント：223 ページの「initialWindow」

子エレメント：

コンテンツ
true（デフォルト）または false

例

```
<resizable>false</resizable>
```

softKeyboardBehavior

Adobe AIR 2.6 以降、モバイルプロファイル - 省略可能

仮想キーボードが表示されたときのアプリケーションのデフォルトの動作を指定します。デフォルトの動作としては、アプリケーションを上方にパンします。ランタイムにより、フォーカスの当たっているテキストフィールドまたはインタラクティブオブジェクトが画面上に維持されます。アプリケーションで独自のキーボード処理ロジックを実装していない場合は、**pan** オプションを使用します。

また、softKeyboardBehavior エレメントを **none** に設定することによって、この自動動作をオフにできます。この場合、ソフトキーボードが開いたときにテキストフィールドおよびインタラクティブオブジェクトから SoftKeyboardEvent が送出されますが、ランタイムによるアプリケーションのパンまたはサイズ変更は行われません。アプリケーション側で、テキスト入力領域の表示を維持する必要があります。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
none または **pan**。デフォルト値は **pan** です。

例
```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

関連項目
SoftKeyboardEvent

supportedLanguages

Adobe AIR 3.2 以降 - 省略可能

アプリケーションのサポート対象言語を指定します。このエレメントは、iOS、Mac キャプティブランタイムおよび Android アプリケーションでのみ使用されます。その他の種類のアプリケーションでは、このエレメントは無視されます。

このエレメントを指定しない場合、Packager によって、アプリケーションの種類に応じてデフォルトで次の処理が実行されます。

- iOS - アプリケーションのサポート対象言語として、AIR ランタイムのサポート対象言語のすべてが、iOS App Store で表示されます。
- Mac キャプティブランタイム - キャプティブバンドルとともにパッケージ化されたアプリケーションには、ローカライズ情報は含まれません。
- Android — アプリケーションバンドルに、AIR ランタイムのサポート対象言語のすべてに対応するリソースが含まれます。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
空白文字で区切られた、サポート対象言語のリスト。AIR ランタイムのサポート対象言語に対応する ISO 639-1 値で指定します。有効な言語の値は、en、de、es、fr、it、ja、ko、pt、ru、cs、nl、pl、sv、tr、zh、da、nb、iw です。

<supportedLanguages> エレメントの値が空の場合は、Packager でエラーが発生します。

注意：<supportedLanguages> タグを使用している場合に、このタグに含まれていない言語の値は、ローカライズ対象のタグ（name タグなど）で無視されます。<supportedLanguages> タグで指定されていない言語のリソースがネイティブ拡張に含まれている場合は、警告が表示され、その言語のリソースは無視されます。

例
```
<supportedLanguages>en ja fr es</supportedLanguages>
```

supportedProfiles

Adobe AIR 2.0 以降 - 省略可能

アプリケーションでサポートされるプロファイルを指定します。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ

次に示す値を任意に組み合わせて supportedProfiles エレメントに指定できます。

- desktop - デスクトッププロファイル。AIR ファイルを使用してデスクトップコンピューター上にインストールされる AIR アプリケーション用です。この種類のアプリケーションは、NativeProcess クラス（ネイティブアプリケーションとの通信を実現するクラス）にはアクセスできません。
- extendedDesktop - 拡張デスクトッププロファイル。ネイティブアプリケーションインストーラーを使用してデスクトップコンピューター上にインストールされる AIR アプリケーション用です。この種類のアプリケーションは、NativeProcess クラス（ネイティブアプリケーションとの通信を実現するクラス）にアクセスできます。
- mobileDevice - モバイルデバイスプロファイル。モバイルアプリケーション用です。
- extendedMobileDevice - 拡張モバイルデバイスプロファイルは現在使用されていません。

supportedProfiles プロパティはオプションです。このエレメントをアプリケーション記述ファイルに含めない場合、そのアプリケーションは任意のプロファイル用にコンパイルおよびデプロイできます。

複数のプロファイルを指定する場合は、プロファイル間を 1 個の空白文字で区切ります。例えば、次の設定は、そのアプリケーションをデスクトップおよび拡張プロファイルでのみ提供できることを指定します。

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

注意：ADL を使用してアプリケーションを実行するとき、ADL の -profile オプションの値を指定しないと、アプリケーション記述子の最初のプロファイルが使用されます（アプリケーション記述子にプロファイルが指定されていない場合、デスクトッププロファイルが使用されます）。

例

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

関連項目

241 ページの「デバイスプロファイル」

74 ページの「サポートされるプロファイル」

systemChrome

Adobe AIR 1.0 以降 - 省略可能

オペレーティングシステムに用意されている標準のタイトルバー、ボーダーおよびコントロールを使用して、アプリケーションの初期ウィンドウを作成するかどうかを指定します。

ウィンドウのシステムクロムの設定を実行時に変更することはできません。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ

次の値のいずれかを使用します。

- none — システムクロムを提供しません。アプリケーション（または Flex などのアプリケーションフレームワーク）には、ウィンドウクロムを表示する役割があります。
- standard（デフォルト）— システムクロムはオペレーティングシステムから提供されます。

例

```
<systemChrome>standard</systemChrome>
```

text

Adobe AIR 1.1 以降 - 省略可能

ローカライズされたストリングを指定します。

テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

AIR アプリケーションインストーラーでは、オペレーティングシステムのユーザーインターフェイス言語に最も近い xml:lang 属性値を持つ text エレメントが使用されます。

例えば、text エレメントに en（英語）ロケールの値が含まれているインストールを行うとします。オペレーティングシステムのユーザーインターフェイス言語が en（英語）になっている場合は、AIR アプリケーションインストーラーで名前 en が使用されます。システムのユーザーインターフェイス言語が en-US（アメリカ英語）の場合にも名前 en が使用されます。ただし、ユーザーインターフェイス言語が en-US で、アプリケーション記述ファイルに en-US と en-GB の両方が定義されている場合は、AIR アプリケーションインストーラーで en-US が使用されます。

システムのユーザーインターフェイス言語に一致する text エレメントがアプリケーションに定義されていない場合、AIR アプリケーションインストーラーではアプリケーション記述ファイルに定義された最初の name の値が使用されます。

親エレメント：

- 229 ページの「name」
- 212 ページの「description」

子エレメント：なし

コンテンツ

ロケールとローカライズされたテキストのストリングを示す xml:lang 属性。

例

```
<text xml:lang="fr">Bonjour AIR</text>
```

title

Adobe AIR 1.0 以降 - 省略可能

アプリケーションの初期ウィンドウのタイトルバーに表示されるタイトルを指定します。

タイトルが表示されるのは、systemChrome エレメントが standard に設定されている場合のみです。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ

ウィンドウタイトルを含むストリング。

例

```
<title>Example Window Title</title>
```

transparent

Adobe AIR 1.0 以降 - 省略可能

アプリケーションの初期ウィンドウをデスクトップとアルファブレンドするかどうかを指定します。

透明なウィンドウを有効にすると、描画に時間がかかり、より多くのメモリを必要とする場合があります。実行時に透明設定を変更することはできません。

重要：transparent を true に設定できるのは、systemChrome が none の場合に限られます。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true または false（デフォルト）

例
```
<transparent>true</transparent>
```

version

Adobe AIR 1.0 ～ 2.0 — 必須、AIR 2.5 以降では使用できません

アプリケーションのバージョン情報を指定します。

version ストリングは、アプリケーション定義の指示子です。AIR は、version ストリングを解釈しません。したがって、バージョン「3.0」はバージョン「2.0」より新しいとは見なされません。例えば、「1.0」、「.4」、「0.5」、4.9」、「1.3.4a」のように指定します。

AIR 2.5 以降では、version エレメントは versionNumber エレメントと versionLabel エレメントに置き換えられました。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
アプリケーションのバージョンを含むストリング

例
```
<version>0.1 Alpha</version>
```

versionLabel

Adobe AIR 2.5 以降 - 省略可能

人間が読み取りできるバージョンストリングを指定します。

バージョンラベルの値は、versionNumber エレメントの値の代わりにインストールダイアログに表示されます。versionLabel が使用されない場合、両方に versionNumber が使用されます。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
公開されているバージョンテキストを含むストリング。

例
```
<versionLabel>0.9 Beta</versionlabel>
```

versionNumber

Adobe AIR 2.5 以降 — 必須

アプリケーションのバージョン番号。

親エレメント：205 ページの「application」

子エレメント：なし

コンテンツ
バージョン番号には、3 個以下の連結した整数をピリオドで区切って含めることができます。各整数には、0 以上 999 以下の数を指定する必要があります。

例
```
<versionNumber>1.0.657</versionNumber>

<versionNumber>10</versionNumber>

<versionNumber>0.01</versionNumber>
```

visible

Adobe AIR 1.0 以降 - 省略可能

アプリケーションの初期ウィンドウの作成後すぐに表示可能にするかどうかを指定します。

初期ウィンドウを含む AIR ウィンドウは、デフォルトでは非表示状態で作成されます。ウィンドウを表示するには、NativeWindow オブジェクトの activate() メソッドを呼び出すか、visible プロパティを true に設定します。最初はメインウィンドウを非表示のままにしておくことで、ウィンドウの位置やサイズ、およびウィンドウ内のコンテンツのレイアウトに対する変更が表示されないようにすることができます。

Flex mx:WindowedApplication コンポーネントは、applicationComplete イベントが送出される直前にウィンドウを自動的に表示してアクティブにします。ただし、MXML 定義で visible 属性が false に設定されている場合は除きます。

ウィンドウをサポートしないモバイルプロファイルのデバイスでは、visible 設定は無視されます。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
true または false（デフォルト）

例
```
<visible>true</visible>
```

width

Adobe AIR 1.0 以降 - 省略可能

アプリケーション起動時のメインウィンドウの幅。

幅を設定しない場合、ルート SWF ファイルの設定によって決まります。また、HTML ベースの AIR アプリケーションの場合は、オペレーティングシステムによって決まります。

AIR 2 では、ウィンドウの最高の幅は 2,048 から 4,096 ピクセルに変更されました。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
最大値が 4,095 の正の整数。

例

```
<width>1024</width>
```

x

Adobe AIR 1.0 以降 - 省略可能

アプリケーションの初期ウィンドウの水平位置。

ほとんどの場合、固定値を割り当てるのではなく、オペレーティングシステムによるウィンドウ初期位置の決定に従うことをお勧めします。

画面座標系の始点（0,0）は、メインデスクトップ画面の左上隅です（オペレーティングシステムによって決まります）。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
整数値。

例

```
<x>120</x>
```

y

Adobe AIR 1.0 以降 - 省略可能

アプリケーション初期ウィンドウの垂直位置。

ほとんどの場合、固定値を割り当てるのではなく、オペレーティングシステムによるウィンドウ初期位置の決定に従うことをお勧めします。

画面座標系の始点（0,0）は、メインデスクトップ画面の左上隅です（オペレーティングシステムによって決まります）。

親エレメント：223 ページの「initialWindow」

子エレメント：なし

コンテンツ
整数値。

例
```
<y>250</y>
```

第 15 章：デバイスプロファイル

Adobe AIR 2 以降

プロファイルは、アプリケーションが動作するコンピューティングデバイスのクラスを定義するメカニズムです。プロファイルでは、特定のデバイスのクラスで通常サポートされる API と機能のセットを定義します。次のプロファイルを使用できます。

- desktop
- extendedDesktop
- mobileDevice
- extendedMobileDevice

アプリケーション記述子でアプリケーションのプロファイルを定義できます。対象となるプロファイル内のコンピューターとデバイスのユーザーは、アプリケーションをインストールできます。他のコンピューターやデバイスのユーザーはインストールできません。例えば、アプリケーション記述子にデスクトッププロファイルのみを含めると、ユーザーはデスクトップコンピューターでのみアプリケーションをインストールおよび実行できます。

実際にはアプリケーションがサポートしていないプロファイルを含めると、そのような環境ではユーザーの操作感が低下します。アプリケーション記述子にプロファイルを一切指定しない場合、AIR ではアプリケーションは制限されません。サポートされるフォーマットでアプリケーションをパッケージ化できます。また、プロファイルのデバイスを使用するユーザーは、アプリケーションをインストールできますが、実行時に適切に動作しない場合があります。

可能であれば、アプリケーションをパッケージ化するときにプロファイルを制限することをお勧めします。例えば、extendedDesktop プロファイルのみを含める場合、アプリケーションを AIR ファイルとしてはパッケージ化できません。ネイティブインストーラーとしてのみパッケージ化できます。同様に、mobileDevice プロファイルを含めない場合、アプリケーションを Android APK としてパッケージ化できません。

単一のコンピューティングデバイスで、複数のプロファイルをサポートできます。例えば、デスクトップコンピューターの AIR は、デスクトップおよび extendedDesktop プロファイル両方のアプリケーションをサポートします。ただし、拡張デスクトッププロファイルアプリケーションは、ネイティブプロセスと通信できます。また、ネイティブインストーラー（exe、dmg、deb、または rpm）としてパッケージ化する必要があります。一方、デスクトッププロファイルアプリケーションはネイティブプロセスと通信できません。デスクトッププロファイルアプリケーションは AIR ファイルまたはネイティブインストーラーとしてパッケージ化できます。

プロファイルに機能を含めることは、プロファイルの定義の対象となるデバイスのクラスで、その機能のサポートが一般的であることを示します。ただし、プロファイルのすべてのデバイスがすべての機能をサポートするという意味ではありません。例えば、（すべてではなく）ほとんどの携帯電話には加速度センサーが含まれています。通常、ユニバーサルサポートがないクラスおよび機能には、機能を使用する前にチェックできるブールプロパティがあります。例えば加速度センサーの場合、静的プロパティ Accelerometer.isSupported を使用して、現在のデバイスにサポートされる加速度センサーがあるかどうかを判断します。

アプリケーション記述子で supportedProfiles エレメントを使用して AIR アプリケーションに割り当てることができるプロファイルは次のとおりです。

デスクトップ デスクトッププロファイルは、AIR ファイルとしてデスクトップコンピューターにインストールされた AIR アプリケーションの機能のセットを定義します。これらのアプリケーションは、サポートされているデスクトッププラットフォーム（Mac OS、Windows および Linux）にインストールされ、そのプラットフォームで実行されます。AIR 2 より前のバージョンの AIR で開発された AIR アプリケーションは、デスクトッププロファイルと見なすことができます。API の中にはこのプロファイルでは機能しないものがあります。例えば、デスクトップアプリケーションは、ネイティブプロセスとは通信できません。

拡張デスクトップ 拡張デスクトッププロファイルは、ネイティブインストーラーにパッケージ化され、ネイティブインストーラーによってインストールされる AIR アプリケーションの機能のセットを定義します。これらのネイティブインストーラーは、Windows では EXE ファイル、Mac OS では DMG ファイル、Linux では BIN、DEB または RPM ファイルです。拡張デスクトップアプリケーションは、デスクトッププロファイルアプリケーションでは使用できない追加の機能を備えています。詳しくは、56 ページの「デスクトップネイティブインストーラーのパッケージ化」を参照してください。

モバイルデバイス モバイルデバイスのプロファイルは、携帯電話やタブレットなどのモバイルデバイス上にインストールされるアプリケーションの機能のセットを定義します。これらの合うプリケーションは、Android、Blackberry Tablet OS および iOS などのサポート対象のモバイルプラットフォームにインストールされて実行されます。

拡張モバイルデバイス 拡張モバイルデバイスプロファイルは、モバイルデバイスにインストールされるアプリケーションの機能の拡張セットを定義します。現時点では、このプロファイルをサポートするデバイスはありません。

アプリケーション記述ファイルでのターゲットプロファイルの制限

Adobe AIR 2 以降

AIR 2 以降では、アプリケーション記述ファイルには supportedProfiles エレメントが含まれています。このエレメントにより、ターゲットプロファイルを制限できます。例えば、次の設定は、そのアプリケーションをデスクトッププロファイルでのみ提供できることを指定します。

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

このエレメントが設定されている場合、アプリケーションは、リストするプロファイルだけにパッケージ化できます。次の値を使用します。

- desktop - デスクトッププロファイル
- extendedDesktop - 拡張デスクトッププロファイル
- mobileDevice - モバイルデバイスプロファイル

supportedProfiles エレメントはオプションです。このエレメントをアプリケーション記述ファイルに含めない場合、そのアプリケーションは、どのプロファイル用にでもパッケージ化およびデプロイできます。

supportedProfiles エレメントで複数のプロファイルを指定するには、次のように、各プロファイルを空白文字で区切ります。

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

様々なプロファイルの機能

Adobe AIR 2 以降

次の表に、プロファイルによってサポートの有無があるクラスと機能の一覧を示します。

クラスまたは機能	desktop	extendedDesktop	mobileDevice
加速度センサー（Accelerometer.isSupported）	不可	不可	チェック
アクセシビリティ（Capabilities.hasAccessibility）	可	可	不可
音声エコー除去機能（Microphone.getEnhancedMicrophone()）	可	可	不可
ActionScript 2	可	可	不可
CacheAsBitmap マトリクス	不可	不可	可
カメラ（Camera.isSupported）	可	可	可
CameraRoll	不可	不可	可
CameraUI（CameraUI.isSupported）	不可	不可	可
キャプティブランタイムバンドル	可	可	可
ContextMenu（ContextMenu.isSupported）	可	可	不可
DatagramSocket（DatagramSocket.isSupported）	可	可	可
DockIcon（NativeApplication.supportsDockIcon）	チェック	チェック	不可
ドラッグ＆ドロップ（NativeDragManager.isSupported）	可	可	チェック
EncyptedLocalStore（EncyptedLocalStore.isSupported）	可	可	可
Flash Access（DRMManager.isSupported）	可	可	不可
GameInput（GameInput.isSupported）	不可	不可	不可
Geolocation（Geolocation.isSupported）	不可	不可	チェック
HTMLLoader（HTMLLoader.isSupported）	可	可	不可
IME（IME.isSupported）	可	可	チェック
LocalConnection（LocalConnection.isSupported）	可	可	不可
マイク（Microphone.isSupported）	可	可	チェック
マルチチャンネルオーディオ（Capabilities.hasMultiChannelAudio()）	不可	不可	不可
ネイティブ拡張	不可	可	可
NativeMenu（NativeMenu.isSupported）	可	可	不可
NativeProcess（NativeProcess.isSupported）	不可	可	不可

クラスまたは機能	desktop	extendedDesktop	mobileDevice
NativeWindow（NativeWindow.isSupported）	可	可	不可
NetworkInfo（NetworkInfo.isSupported）	可	可	チェック
デフォルトアプリケーションでファイルを開く	制限付き	可	不可
PrintJob（PrintJob.isSupported）	可	可	不可
SecureSocket（SecureSocket.isSupported）	可	可	チェック
ServerSocket（ServerSocket.isSupported）	可	可	可
シェーダー	可	可	制限付き
Stage3D（Stage.stage3Ds.length）	可	可	可
ステージの方向（Stage.supportsOrientationChange）	不可	不可	可
StageVideo	不可	不可	チェック
StageWebView（StageWebView.isSupported）	可	可	可
ログイン時のアプリケーションの起動（NativeApplication.supportsStartAtLogin）	可	可	不可
StorageVolumeInfo（StorageVolumeInfo.isSupported）	可	可	不可
システムアイドルモード	不可	不可	可
SystemTrayIcon（NativeApplication.supportsSystemTrayIcon）	チェック	チェック	不可
Text Layout Framework の入力	可	可	不可
アップデーター（Updater.isSupported）	可	不可	不可
XMLSignatureValidator（XMLSignatureValidator.isSupported）	可	可	不可

表の各エントリは次の意味を表します。

- チェック - 機能はプロファイルの一部のデバイスではサポートされますが、すべてのデバイスではサポートされません。機能を使用する前に、ランタイムを調べ、機能がサポートされているかどうかを確認してください。
- 制限付き - 機能はサポートされますが、重要な制限があります。詳しくは、関連するドキュメントを参照してください。
- 不可 - 機能はプロファイルでサポートされません。
- 可 - 機能はプロファイルでサポートされます。個々のコンピューティングデバイスによっては、機能に必要となるハードウェアを搭載していない可能性があります。例えば、カメラを搭載していない電話もあります。

ADL を使用したデバッグ時のプロファイルの指定

Adobe AIR 2 以降

ADL では、アプリケーション記述ファイルにある supportedProfiles エレメントでサポート対象のプロファイルが指定されているかどうかが確認されます。指定した場合、デフォルトでは、プロファイルとして最初に示されているサポート対象のプロファイルがデバッグ時に使用されます。

ADL デバッグセッション用にプロファイルを指定するには、-profile コマンドライン引数を使用します（157 ページの「AIR Debug Launcher（ADL）」を参照してください）。この引数は、アプリケーション記述ファイルにある supportedProfiles エレメントでプロファイルを指定するかどうかに関係なく使用できます。ただし、supportedProfiles エレメントを指定する場合は、指定するプロファイルがコマンドラインに含まれていなければなりません。含まれていない場合、ADL でエラーが生成されます。

第 16 章：AIR.SWF ブラウザー API

シームレスインストールの badge.swf のカスタマイズ

SDK に付属している badge.swf ファイルの使用に加え、ブラウザーページで使用できる独自の SWF ファイルを作成できます。カスタム SWF ファイルは、次の方法でランタイムと対話できます。

- AIR アプリケーションをインストールできます。251 ページの「ブラウザーからの AIR アプリケーションのインストール」を参照してください。
- 特定の AIR アプリケーションがインストールされているかどうかを確認できます。250 ページの「AIR アプリケーションがインストールされているかどうかの Web ページからの確認」を参照してください。
- ランタイムがインストールされているかどうかを確認できます。250 ページの「ランタイムがインストールされているかどうかの確認」を参照してください。
- ユーザーのシステムでインストール済みの AIR アプリケーションを起動できます。252 ページの「インストール済み AIR アプリケーションのブラウザーからの起動」を参照してください。

これらの機能はすべて、adobe.com: air.swf にホストされている SWF ファイル内の API を呼び出して使用できます。badge.swf をカスタマイズして、独自の SWF ファイルから air.swf API を呼び出すことができます。

また、ブラウザーで実行中の SWF ファイルは、LocalConnection クラスを使用して実行されている AIR アプリケーションと通信できます。詳しくは、「Flash Player および AIR の他のインスタンスとの通信」（ActionScript 開発者用）、または「Flash Player および AIR の他のインスタンスとの通信」（HTML 開発者用）を参照してください。

重要：Windows または Mac OS では、この節で説明する機能（および air.swf ファイルの API）を使用するには、エンドユーザーが Adobe® Flash® Player 9 アップデート 3 以降を Web ブラウザーでインストールしている必要があります。Linux では、シームレスインストール機能の実行に Flash Player 10（バージョン 10.0.12.36 以降）が必要です。インストールされている Flash Player のバージョンを確認し、Flash Player の必要なバージョンがインストールされていない場合は、ユーザーに代替のインターフェイスを提供するコードを作成できます。例えば、古いバージョンの Flash Player がインストールされている場合は、（badge.swf ファイルまたは air.swf API を使用してアプリケーションをインストールするのではなく）AIR ファイルのダウンロードバージョンへのリンクを提供します。

badge.swf ファイルを使用した AIR アプリケーションのインストール

AIR SDK および Flex SDK には、badge.swf ファイルが含まれています。このファイルを使用すると、シームレスインストール機能を簡単に使用できます。badge.swf では、ランタイムと AIR アプリケーションを Web ページのリンクからインストールできます。Web サイトに配布するための badge.swf ファイルとそのソースコードが提供されます。

Web ページへの badge.swf ファイルの埋め込み

1. AIR SDK または Flex SDK の samples/badge ディレクトリにある次のファイルを探し、Web サーバーに追加します。
   - badge.swf
   - default_badge.html
   - AC_RunActiveContent.js
2. default_badge.html ページをテキストエディターで開きます。

3 default_badge.html ページの AC_FL_RunContent() JavaScript 関数で、FlashVars パラメーター定義を次のように調整します。

パラメーター	説明
appname	ランタイムがインストールされていないときに、SWF ファイルよって表示されるアプリケーションの名前。
appurl	（必須）ダウンロードする AIR ファイルの URL。相対 URL ではなく、絶対 URL を使用する必要があります。
airversion	（必須）ラインタイム 1.0 バージョンの場合は、このパラメーターに 1.0 を設定します。
imageurl	badge に表示される画像（オプション）の URL。
buttoncolor	ダウンロードボタンの色（FFCC00 のように、16 進数で指定します）。
messagecolor	ランタイムがインストールされていないときにボタンの下に表示されるテキストメッセージの色（FFCC00 のように、16 進数で指定します）。

4 badge.swf ファイルの最小サイズは、217 ピクセル（幅）x 180 ピクセル（高さ）です。AC_FL_RunContent() 関数の width パラメーターと height パラメーターの値を、ニーズに合わせて調整します。

5 default_badge.html ファイルの名前を変更し、ニーズに合わせてそのコードを調整します（または別の HTML ページにコードを含めます）。

注意：badge.swf ファイルを読み込む HTML embed タグに対しては、wmode 属性は設定せず、デフォルト（"window"）のままにしておいてください。デフォルト以外の wmode 設定を使用すると、システムによってはインストールが失敗します。また、他の wmode 設定を使用するとエラー「エラー #2044: ハンドルされていない ErrorEvent : text=Error #2074: ステージが小さすぎるため、ダウンロード UI を格納できません」が発生します。

また、badge.swf ファイルを編集および再コンパイルすることもできます。詳しくは、248 ページの「badge.swf ファイルの変更」を参照してください。

Web ページのシームレスインストールリンクからの AIR アプリケーションのインストール

ページにシームレスインストールリンクを追加すると、SWF ファイル内のリンクをクリックして AIR アプリケーションをインストールできます。

1 Flash Player（Windows および Mac OS ではバージョン 9 アップデート 3 以降、Linux ではバージョン 10）がインストールされている Web ブラウザーで HTML ページへ移動します。

2 Web ページで、badge.swf ファイル内のリンクをクリックします。

- ランタイムを既にインストールしている場合は、次の手順にスキップします。
- ランタイムをインストールしていない場合は、インストールするかどうかを確認するダイアログボックスが表示されます。ランタイムをインストールして（3 ページの「Adobe AIR のインストール」を参照）、次の手順に進みます。

3 インストールウィンドウで、デフォルトの設定を選択した状態のまま、「続行」をクリックします。

Windows コンピューターでは、AIR が次の動作を自動的に行います。

- c:¥Program Files¥ にアプリケーションをインストールします。
- アプリケーションのデスクトップショートカットを作成します。
- スタートメニューショートカットを作成します。
- 「プログラムの追加と削除」コントロールパネルにアプリケーションのエントリを追加します。

Mac OS では、インストーラーによってアプリケーションがアプリケーションディレクトリ（Mac OS の / アプリケーションディレクトリなど）に追加されます。

Linux コンピューターでは、AIR が次の動作を自動的に行います。

- アプリケーションを /opt にインストールします。
- アプリケーションのデスクトップショートカットを作成します。
- スタートメニューショートカットを作成します。
- アプリケーションのエントリをシステムパッケージマネージャーに追加します。

4 必要なオプションを選択し、「インストール」ボタンをクリックします。

5 インストールが完了したら、「完了」をクリックします。

badge.swf ファイルの変更

Flex SDK および AIR SDK には badge.swf ファイル用のソースファイルが用意されています。SDK の samples/badge フォルダーには、次のファイルが含まれています。

ソースファイル	説明
badge.fla	badge.swf ファイルのコンパイルに使用するソースの Flash ファイル。badge.fla ファイルは、SWF 9 ファイル（Flash Player に読み込み可能）にコンパイルされます。
AIRBadge.as	basdge.fla ファイルで使用される基本クラスを定義する ActionScript 3.0 クラス。

Flash Professional を使用して、badge.fla ファイルの仮想インターフェイスを再デザインできます。

AIRBadge クラスで定義される AIRBadge() コンストラクター関数は、http://airdownload.adobe.com/air/browserapi/air.swf にホストされている air.swf ファイルを読み込みます。air.swf ファイルには、シームレスインストール機能を使用するためのコードが含まれています。

onInit() メソッド（AIRBadge クラス内）は、air.swf ファイルが正常に読み込まれると呼び出されます。

```
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;
    }
}
```

コードでは、グローバルな _air 変数が、読み込んだ air.swf ファイルのメインクラスに設定されます。このクラスには、badge.swf ファイルがシームレスインストール機能を呼び出す際にアクセスする次のパブリックメソッドが含まれています。

メソッド	説明
getStatus()	コンピューターにランタイムがインストールされているか（またはインストール可能か）どうかを確認します。詳しくは、250 ページの「ランタイムがインストールされているかどうかの確認」を参照してください。 • runtimeVersion - インストールするアプリケーションに必要なランタイムのバージョンを示すストリング（「1.0.M6」など）。
installApplication()	指定されたアプリケーションをユーザーのマシンにインストールします。詳しくは、251 ページの「ブラウザーからの AIR アプリケーションのインストール」を参照してください。 • url - URL を定義するストリング。URL のパスには、相対 URL ではなく、絶対 URL を使用する必要があります。 • runtimeVersion - インストールするアプリケーションに必要なランタイムのバージョンを示すストリング（「2.5.」など）。 • arguments - インストール時にアプリケーションが起動する場合にアプリケーションに渡される引数。アプリケーション記述ファイルで allowBrowserInvocation エレメントが true に設定されている場合、アプリケーションはインストール時に起動します（アプリケーション記述ファイルについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください）。ブラウザーからのシームレスインストールによりアプリケーションが起動している場合（インストール時に起動することをユーザーが選択している場合）、アプリケーションの NativeApplication オブジェクトは、引数が渡されている場合にのみ BrowserInvokeEvent オブジェクトを送出します。アプリケーションに渡すデータのセキュリティ上の影響を考慮します。詳しくは、252 ページの「インストール済み AIR アプリケーションのブラウザーからの起動」を参照してください。

url および runtimeVersion の設定は、コンテナの HTML ページの FlashVars 設定を通じて SWF ファイルに渡されます。

インストール時にアプリケーションが自動的に起動する場合は、LocalConnection 通信を使用して、インストールされたアプリケーションの呼び出し時に badge.swf ファイルと通信させることができます。詳しくは、「Flash Player および AIR の他のインスタンスとの通信」（ActionScript 開発者用）、または「Flash Player および AIR の他のインスタンスとの通信」（HTML 開発者用）を参照してください。

また、air.swf ファイルの getApplicationVersion() メソッドを呼び出して、アプリケーションがインストールされているかどうかを確認することもできます。このメソッドは、アプリケーションのインストールプロセスの前、またはインストールの開始後に呼び出すことができます。詳しくは、250 ページの「AIR アプリケーションがインストールされているかどうかの Web ページからの確認」を参照してください。

air.swf ファイルの読み込み

air.swf ファイルの API を使用する独自の SWF ファイルを作成し、ブラウザー内の Web ページからランタイムおよび AIR アプリケーションを操作できます。air.swf ファイルは、http://airdownload.adobe.com/air/browserapi/air.swf にホストされています。SWF ファイルから air.swf API を参照するには、air.swf ファイルを SWF ファイルと同じアプリケーションドメインに読み込みます。次のコードは、読み込み中の SWF ファイルのアプリケーションドメインに air.swf ファイルを読み込む例を示しています。

```
 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;
}
```

air.swf ファイルが読み込まれたら（Loader オブジェクトの contentLoaderInfo オブジェクトが init イベントを送出した後）、以下の節で説明している air.swf API のいずれかを呼び出すことができます。

注意：AIR SDK および Flex SDK に付属している badge.swf ファイルでは、air.swf ファイルを自動的に読み込みます。246 ページの「badge.swf ファイルを使用した AIR アプリケーションのインストール」を参照してください。この節に示す手順は、air.swf ファイルを読み込む独自の SWF ファイルの作成に適用されます。

ランタイムがインストールされているかどうかの確認

SWF ファイルでは、http://airdownload.adobe.com/air/browserapi/air.swf から読み込んだ air.swf ファイルで getStatus() メソッドを呼び出してランタイムがインストールされているかどうかを確認できます。詳しくは、249 ページの「air.swf ファイルの読み込み」を参照してください。

air.swf ファイルが読み込まれると、SWF ファイルでは air.swf ファイルの getStatus() メソッドを呼び出すことができます。

```
var status:String = airSWF.getStatus();
```

getStatus() メソッドは、コンピューター上のランタイムのステータスに基づいて、次のストリング値のいずれか 1 つを返します。

ストリング値	説明
"available"	ランタイムはコンピューターにインストールできますが、現在インストールされていません。
"unavailable"	ランタイムはこのコンピューターにインストールできません。
"installed"	ランタイムはこのコンピューターにインストールされています。

getStatus() メソッドでは、必要なバージョンの Flash Player（Windows および Mac OS ではバージョン 9 アップデート 3 以降、Linux ではバージョン 10）がブラウザーにインストールされていない場合、エラーがスローされます。

AIR アプリケーションがインストールされているかどうかの Web ページからの確認

SWF ファイルでは、http://airdownload.adobe.com/air/browserapi/air.swf から読み込んだ air.swf ファイルの getApplicationVersion() メソッドを呼び出して、AIR アプリケーション（対応するアプリケーション ID と発行者 ID を持つ）がインストールされているかどうかを確認できます。詳しくは、249 ページの「air.swf ファイルの読み込み」を参照してください。

air.swf ファイルが読み込まれると、SWF ファイルでは air.swf ファイルの getApplicationVersion() メソッドを呼び出すことができます。

```
 var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.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.
    }
}
```

getApplicationVersion() メソッドには、次のパラメーターがあります。

パラメーター	説明
appID	アプリケーションのアプリケーション ID。詳しくは、221 ページの「id」を参照してください。
pubID	アプリケーションの発行者 ID。詳しくは、231 ページの「publisherID」を参照してください。対象のアプリケーションに発行者 ID がない場合は、pubID パラメーターを空のストリング（""）に設定します。
callback	ハンドラー関数として処理されるコールバック関数。getApplicationVersion() メソッドは非同期に動作し、インストールされているバージョンを検出するとき（またはインストールされているバージョンがないことを検出するとき）に、このコールバックメソッドが呼び出されます。コールバックメソッドの定義では、1 つのパラメーター（インストールされているアプリケーションのバージョンストリングに設定されているストリング）を含める必要があります。アプリケーションがインストールされていない場合は、前のコード例に示したように null 値が関数に渡されます。

getApplicationVersion() メソッドでは、必要なバージョンの Flash Player（Windows および Mac OS ではバージョン 9 アップデート 3 以降、Linux ではバージョン 10）がブラウザーにインストールされていない場合、エラーがスローされます。

注意：AIR 1.5.3 では、発行者 ID の使用は推奨されません。発行者 ID はアプリケーションに自動的に割り当てられなくなりました。下位互換性を維持するため、アプリケーションでは引き続き発行者 ID を指定することができます。

ブラウザーからの AIR アプリケーションのインストール

SWF ファイルでは、http://airdownload.adobe.com/air/browserapi/air.swf から読み込んだ air.swf ファイルで installApplication() メソッドを呼び出して AIR アプリケーションをインストールできます。詳しくは、249 ページの「air.swf ファイルの読み込み」を参照してください。

air.swf ファイルが読み込まれると、SWF ファイルでは、次のコードに示すように、air.swf ファイルの installApplication() メソッドを呼び出すことができます。

```
var url:String = "http://www.example.com/myApplication.air";
var runtimeVersion:String = "1.0";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.installApplication(url, runtimeVersion, arguments);
```

installApplication() メソッドは、指定されたアプリケーションをユーザーのマシンにインストールします。このメソッドのパラメーターは次のとおりです。

パラメーター	説明
url	インストールする AIR ファイルの URL を定義するストリング。URL のパスには、相対 URL ではなく、絶対 URL を使用する必要があります。
runtimeVersion	インストールするアプリケーションに必要なランタイムのバージョンを示すストリング（「1.0」など）。
arguments	インストール時にアプリケーションが起動する場合にアプリケーションに渡される引数の配列。引数内で認識される文字は半角英数のみです。その他の値を渡す必要がある場合は、何らかのエンコード方式を使用してください。 アプリケーション記述ファイルで allowBrowserInvocation エレメントが true に設定されている場合、アプリケーションはインストール時に起動します（アプリケーション記述ファイルについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください）。ブラウザーからのシームレスインストールによりアプリケーションが起動している場合（インストール時に起動することをユーザーが選択している場合）、アプリケーションの NativeApplication オブジェクトは、引数が渡されている場合にのみ BrowserInvokeEvent オブジェクトを送出します。詳しくは、252 ページの「インストール済み AIR アプリケーションのブラウザーからの起動」を参照してください。

installApplication() メソッドは、マウスのクリックなどのユーザーイベントのイベントハンドラーで呼び出されたときにのみ動作できます。

installApplication() メソッドでは、必要なバージョンの Flash Player（Windows および Mac OS ではバージョン 9 アップデート 3 以降、Linux ではバージョン 10）がブラウザーにインストールされていない場合、エラーがスローされます。

Mac OS でアプリケーションのアップデートバージョンをインストールするには、アプリケーションディレクトリにインストールするための適切なシステム権限（アプリケーションでランタイムをアップデートする場合は管理権限も）が必要です。Windows では、管理権限が必要です。

また、air.swf ファイルの getApplicationVersion() メソッドを呼び出して、アプリケーションが既にインストールされているかどうかを確認することもできます。このメソッドは、アプリケーションのインストールプロセスの開始前、またはインストールの開始後に呼び出すことができます。詳しくは、250 ページの「AIR アプリケーションがインストールされているかどうかの Web ページからの確認」を参照してください。実行中のアプリケーションは、LocalConnection クラスを使用してブラウザーで SWF コンテンツと通信できます。詳しくは、「Flash Player および AIR の他のインスタンスとの通信」（ActionScript 開発者用）、または「Flash Player および AIR の他のインスタンスとの通信」（HTML 開発者用）を参照してください。

インストール済み AIR アプリケーションのブラウザーからの起動

ブラウザー呼び出し機能を使用するには（ブラウザーから起動できるようにするには）、対象アプリケーションのアプリケーション記述ファイルに次の設定を含める必要があります。

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

アプリケーション記述ファイルについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

ブラウザーの SWF ファイルでは、http://airdownload.adobe.com/air/browserapi/air.swf から読み込んだ air.swf ファイルで launchApplication() メソッドを呼び出して AIR アプリケーションを起動できます。詳しくは、249 ページの「air.swf ファイルの読み込み」を参照してください。

air.swf ファイルが読み込まれると、SWF ファイルでは、次のコードに示すように、air.swf ファイルの launchApplication() メソッドを呼び出すことができます。

```
var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.launchApplication(appID, pubID, arguments);
```

launchApplication() メソッドは、air.swf ファイルのトップレベルで定義されます（ユーザーインターフェイス SWF ファイルのアプリケーションドメインに読み込まれます）。このメソッドを呼び出すと、AIR で指定されたアプリケーションが起動します（アプリケーションがインストール済みで、アプリケーション記述ファイルの allowBrowserInvocation 設定でブラウザー呼び出しが許可されている場合）。このメソッドのパラメーターは次のとおりです。

パラメーター	説明
appID	起動するアプリケーションのアプリケーション ID。詳しくは、221 ページの「id」を参照してください。
pubID	起動するアプリケーションの発行者 ID。詳しくは、231 ページの「publisherID」を参照してください。対象のアプリケーションに発行者 ID がない場合は、pubID パラメーターを空のストリング（""）に設定します。
arguments	アプリケーションに渡す引数の配列。アプリケーションの NativeApplication オブジェクトは、引数のプロパティがこの配列に設定されている BrowserInvokeEvent イベントを送出します。引数内で認識される文字は半角英数のみです。その他の値を渡す必要がある場合は、何らかのエンコード方式を使用してください。

launchApplication() メソッドは、マウスのクリックなどのユーザーイベントのイベントハンドラーで呼び出されたときにのみ動作できます。

launchApplication() メソッドでは、必要なバージョンの Flash Player（Windows および Mac OS ではバージョン 9 アップデート 3 以降、Linux ではバージョン 10）がブラウザーにインストールされていない場合、エラーがスローされます。

アプリケーション記述ファイルで allowBrowserInvocation エレメントが false に設定されている場合、launchApplication() メソッドを呼び出してもアプリケーションは起動されません。

アプリケーションを起動するためのユーザーインターフェイスを示す前に、air.swf ファイルで getApplicationVersion() メソッドを呼び出す必要があります。詳しくは、250 ページの「AIR アプリケーションがインストールされているかどうかの Web ページからの確認」を参照してください。

ブラウザー呼び出し機能によりアプリケーションが呼び出されると、アプリケーションの NativeApplication オブジェクトは BrowserInvokeEvent オブジェクトを送出します。詳しくは、「ブラウザーからの AIR アプリケーションの呼び出し」（ActionScript 開発者用）または「ブラウザーからの AIR アプリケーションの呼び出し」（HTML 開発者用）を参照してください。

ブラウザー呼び出し機能を使用する場合は、セキュリティ上の影響を考慮してください。セキュリティ上の影響については、「ブラウザーからの AIR アプリケーションの呼び出し」（ActionScript 開発者用）および「ブラウザーからの AIR アプリケーションの呼び出し」（HTML 開発者用）で説明しています。

実行中のアプリケーションは、LocalConnection クラスを使用してブラウザーで SWF コンテンツと通信できます。詳しくは、「Flash Player および AIR の他のインスタンスとの通信」（ActionScript 開発者用）、または「Flash Player および AIR の他のインスタンスとの通信」（HTML 開発者用）を参照してください。

注意： AIR 1.5.3 では、発行者 ID の使用は推奨されません。発行者 ID はアプリケーションに自動的に割り当てられなくなりました。下位互換性を維持するため、アプリケーションでは引き続き発行者 ID を指定することができます。

第 17 章：AIR アプリケーションのアップデート

ユーザーは、コンピューター上の AIR ファイルをダブルクリックするかブラウザーから（シームレスインストール機能を使用して）AIR アプリケーションをインストールまたはアップデートできます。Adobe® AIR® インストーラーアプリケーションによってインストールが管理され、既に存在するアプリケーションをアップデートする場合はユーザーに警告が表示されます

一方、Updater クラスを使用して、インストールされているアプリケーションの新しいバージョンへのアップデートをアプリケーション自体で行うこともできます（インストールされているアプリケーションで、新しいバージョンをダウンロードしてインストールできるようになったことを検出できます）。Updater クラスに含まれる update() メソッドを使用することにより、ユーザーのコンピューター上の AIR ファイルを参照してそのバージョンにアップデートすることができます。Updater クラスを使用するには、アプリケーションを AIR ファイルにパッケージ化する必要があります。ネイティブの実行可能ファイルまたはパッケージとしてパッケージ化されたアプリケーションでは、ネイティブプラットフォームが提供するアップデート機能を使用する必要があります。

アップデート AIR ファイルのアプリケーション ID と 発行者 ID の両方が、アップデートするアプリケーションと一致する必要があります。発行者 ID は署名者の証明書から取得します。アップデートおよびアップデートされるアプリケーションの両方を、同じ証明書を使用して署名する必要があります。

AIR 1.5.3 以降では、アプリケーション記述ファイルには <publisherID> エレメントが含まれています。AIR 1.5.2 以前を使用して開発されたバージョンのアプリケーションがある場合は、このエレメントを使用する必要があります。詳しくは、231 ページの「publisherID」を参照してください。

AIR 1.1 以降では、アプリケーションを移行して新しいコード署名証明書を使用できます。アプリケーションを移行して新しい署名を使用するには、新しい証明書と元の証明書の両方を使用してアップデート AIR ファイルに署名する必要があります。証明書の移行は一方的なプロセスです。移行後は、新しい証明書（または両方の証明書）で署名した AIR ファイルのみが既存のインストールへのアップデートとして認識されます。

アプリケーションのアップデートの管理は複雑になることがあります。AIR 1.5 には、新しいアップデートフレームワーク（**AdobeAIR** アプリケーション向け）が含まれています。このフレームワークでは、開発者が AIR アプリケーションに優れたアップデート機能を実装するのを支援する API が提供されています。

証明書の移行により、自己署名証明書から商用コード署名証明書に切り替えたり、自己署名証明書間または商用コード署名証明書間で切り替えたりすることができます。証明書を移行しないと、既存のユーザーは現在のバージョンのアプリケーションを削除してから新しいバージョンをインストールする必要があります。詳しくは、190 ページの「証明書の変更」を参照してください。

アプリケーションにはアップデートのメカニズムを含めることをお勧めします。新しいバージョンのアプリケーションを作成する場合、アップデートのメカニズムを使用すると、新しいバージョンをインストールするようにユーザーに求めることができます。

AIR アプリケーションがインストール、アップデートまたは削除されるとき、AIR アプリケーションインストーラーによってログファイルが作成されます。これらのログの内容は、インストールの問題の原因を突き止めるのに役立ちます。インストールのログに関する説明を参照してください。

注意：Adobe AIR ランタイムの新しいバージョンには、WebKit の更新されたバージョンが含まれている場合があります。更新されたバージョンの WebKit が原因で、デプロイされた AIR アプリケーションの HTML コンテンツに予期しない変更が発生する可能性があります。このような変更が発生した場合は、アプリケーションをアップデートする必要があります。アップデートのメカニズムを使用すると、新しいバージョンのアプリケーションがあることをユーザーに通知できます。詳しくは、「HTML 環境について」（ActionScript 開発者用）または「HTML 環境について」（HTML 開発者用）を参照してください。

アプリケーションのアップデートについて

Updater クラス（flash.desktop パッケージのクラス）には update() という 1 つのメソッドが用意されています。このメソッドを使用すると、現在実行しているアプリケーションを別のバージョンに更新できます。例えば、AIR ファイルのバージョン（「Sample_App_v2.air」）がユーザーのデスクトップに保存されている場合にアプリケーションをアップデートするコードの例を次に示します。

ActionScript の例：

```
var updater:Updater = new Updater();
var airFile:File = File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version:String = "2.01";
updater.update(airFile, version);
```

JavaScript の例：

```
var updater = new air.Updater();
var airFile = air.File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version = "2.01";
updater.update(airFile, version);
```

アプリケーションで Updater クラスを使用する前に、ユーザーまたはアプリケーションによって、アップデートバージョンの AIR ファイルがコンピューターにダウンロードされている必要があります。詳しくは、257 ページの「ユーザーのコンピューターへの AIR ファイルのダウンロード」を参照してください。

Updater.update() メソッド呼び出しの結果

ランタイムのアプリケーションで update() メソッドを呼び出すと、ランタイムによってアプリケーションが終了され、AIR ファイルからの新しいバージョンのインストールが試行されます。ランタイムによって、AIR ファイルで指定されたアプリケーション ID および発行者 ID が、update() メソッドの呼び出し元アプリケーションのアプリケーション ID および発行者 ID と一致するかどうかが確認されます（アプリケーション ID および発行者 ID について詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください）。さらに、バージョンストリングが、update() メソッドに渡された version ストリングと一致するかどうかが確認されます。インストールが正常に完了した場合、ランタイムによって新しいバージョンのアプリケーションが起動します。それ以外の場合（インストールを完了できなかった場合）は、既存（インストール前）のバージョンのアプリケーションが再度起動します。

Mac OS でアプリケーションのアップデートバージョンをインストールするには、アプリケーションディレクトリにインストールするための適切なシステム権限が必要です。Windows および Linux では、管理権限が必要です。

アップデートバージョンのアプリケーションで、アップデートバージョンのランタイムが必要な場合は、新しいランタイムバージョンがインストールされます。ランタイムをアップデートする作業は、当該コンピューターの管理権限を持つユーザーが実行する必要があります。

ADL を使用してアプリケーションをテストしているときに update() メソッドを呼び出すと、ランタイム例外が発生します。

バージョンストリングについて

update() メソッドの version パラメーターとして指定するストリングは、インストールする AIR ファイルのアプリケーション記述ファイルで version または versionNumber エレメントに指定されたストリングと一致する必要があります。セキュリティ上の理由から、必ず version パラメーターを指定する必要があります。アプリケーションで AIR ファイルのバージョン番号を必ず確認することにより、古いバージョンのアプリケーションが誤ってインストールされることがなくなります（古いバージョンのアプリケーションには、最新バージョンのアプリケーションでは修正済みのセキュリティ上の脆弱性が含まれている場合があります）。また、ダウングレード攻撃を回避するために、AIR ファイルのバージョンストリングとインストールされたアプリケーションのバージョンストリングが一致するかどうかをアプリケーションで確認する必要があります。

AIR 2.5 より前の AIR では、バージョンストリングを任意の形式で指定できます。例えば、「2.01」や「version 2」のように指定できます。AIR 2.5 以降は、バージョンストリングを、ピリオドで区切った最大 3 つの 3 桁の数値で指定する必要があります。例えば、「.0」、「1.0」および「67.89.999」は、すべて有効な有効バージョン番号です。アプリケーションをアップデートする前に、アップデートのバージョン番号を検証します。

Adobe AIR アプリケーションで Web 経由で AIR ファイルをダウンロードする場合の優れた方法として、ダウンロードしている Adobe AIR アプリケーションのバージョンを Web サービスで通知できるメカニズムがあります。その後、このストリングを、アプリケーションで update() メソッドの version パラメーターとして使用できます。AIR ファイルをこれ以外の方法で取得した場合は AIR ファイルのバージョンが不明な場合がありますが、AIR アプリケーションで AIR ファイルを確認してバージョン情報を特定することができます（AIR ファイルは ZIP 圧縮されたアーカイブで、アプリケーション記述ファイルはそのアーカイブの 2 つ目のレコードです）。

アプリケーション記述ファイルについて詳しくは、200 ページの「AIR アプリケーション記述ファイル」を参照してください。

アプリケーションアップデートの署名ワークフロー

アップデートをアドホックな方法で発行すると、複数のアプリケーションバージョンを管理するタスクが複雑になり、さらに証明書の期限日の追跡も難しくなります。アップデートを発行できるようになるまでに、証明書が期限切れとなる可能性もあります。

Adobe AIR ランタイムでは、移行署名なしで発行されたアプリケーションアップデートを新しいアプリケーションとして扱います。ユーザーは、このようなアプリケーションアップデートをインストールする前に、現在の AIR アプリケーションをアンインストールする必要があります。

この問題を解決するには、アップデートされた各アプリケーションを、最新の証明書を使用して、個別のデプロイ URL にアップロードしてください。また、証明書が 180 日の猶予期間に入っている場合に移行署名を適用するよう通知するメカニズムを含めてください。詳しくは、195 ページの「AIR アプリケーションのアップデートバージョンの署名」を参照してください。

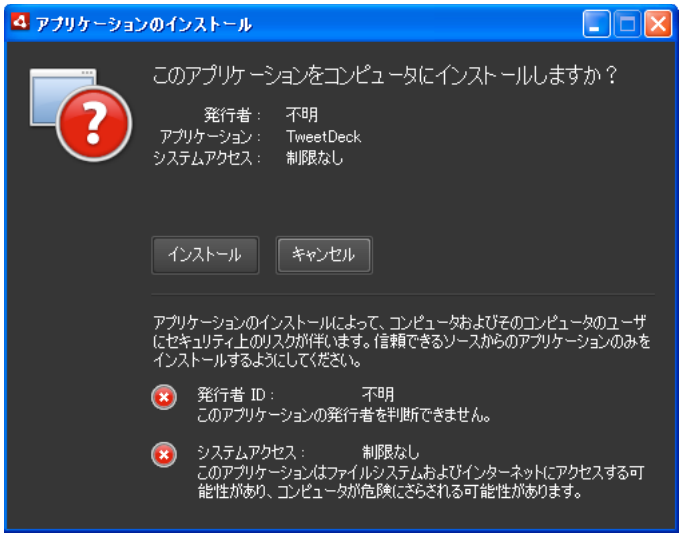
署名の適用方法については、162 ページの「ADT コマンド」を参照してください。

移行署名の適用手順を効率よく実行するには、次のタスクを実行します。

- アップデートされた各アプリケーションを、個別のデプロイ URL にアップロードします。
- アップグレード記述 XML ファイルと、そのアップデート用の最新の証明書を、同じ URL にアップロードします。
- 最新の証明書を使用して、アップデートされたアプリケーションに署名します。
- 異なる URL にある以前のバージョンの署名に使用した証明書に基づいて、アップデートされたアプリケーションに移行署名を適用します。

カスタムのアプリケーションアップデートユーザーインターフェイスの表示

AIR には、デフォルトのアップデートインターフェイスがあります。



ユーザーがアプリケーションのバージョンを最初にマシンにインストールしたときは、常にこのインターフェイスが使用されます。ただし、独自のインターフェイスを定義して以降のインスタンスに使用することができます。アプリケーションにカスタムのアップデートインターフェイスが定義されている場合は、現在インストールされているアプリケーションのアプリケーション記述ファイルで customUpdateUI エレメントを指定します。

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

アプリケーションがインストールされ、インストールされたアプリケーションとアプリケーション ID および発行者 ID が一致する AIR ファイルがユーザーによって開かれると、デフォルトの AIR アプリケーションインストーラーではなくランタイムによってアプリケーションが起動します。詳しくは、211 ページの「customUpdateUI」を参照してください。

アプリケーションが実行されたとき（NativeApplication.nativeApplication オブジェクトから load イベントが送出されたとき）に、アプリケーションを（Updater クラスを使用して）アップデートするかどうかをアプリケーションで指定できます。アップデートするように指定する場合、アプリケーション独自の（標準で実行されるインターフェイスとは異なる）インストールインターフェイスをユーザーに表示することができます。

ユーザーのコンピューターへの AIR ファイルのダウンロード

Updater クラスを使用するには、ユーザーまたはアプリケーションによって、ユーザーのローカルコンピューターに AIR ファイルがあらかじめ保存されている必要があります。

注意：AIR 1.5 にはアップデートフレームワークが含まれており、開発者が AIR アプリケーションに優れたアップデート機能を実装する際に役立ちます。このフレームワークを使用するのは、Update クラスの update() メソッドを直接使用するよりも格段に簡単です。詳しくは、260 ページの「アップデートフレームワークの使用」を参照してください。

次のコードでは、URL（http://example.com/air/updates/Sample_App_v2.air）から AIR ファイルを読み込んで、その AIR ファイルをアプリケーション記憶域ディレクトリに保存します。

ActionScript の例：

```
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.");
}
```

JavaScript の例：

```
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.");
}
```

詳しくは、以下を参照してください。

- ファイルの読み取りと書き込みのワークフロー（ActionScript 開発者用）
- ファイルの読み取りと書き込みのワークフロー（HTML 開発者用）

アプリケーションが初めて実行されたかどうかの確認

アプリケーションのアップデートの完了後に、ユーザーに「はじめに」または「ようこそ」などのメッセージを表示することができます。起動時に、メッセージを表示するかどうかを判断できるように、初めて実行されたかどうかがアプリケーションで確認されます。

注意：AIR 1.5 にはアップデートフレームワークが含まれており、開発者が AIR アプリケーションに優れたアップデート機能を実装する際に役立ちます。このフレームワークにより、アプリケーションのバージョンが初めて実行されたかどうかを簡単に確認できます。詳しくは、260 ページの「アップデートフレームワークの使用」を参照してください。

そのための 1 つの方法として、アプリケーションの初期化時にファイルがアプリケーション記憶域ディレクトリに保存されます。アプリケーションが起動されるたびに、そのファイルが存在するかどうかをアプリケーションで確認します。ファイルが存在しない場合、アプリケーションは現在のユーザーに初めて実行されたことになります。ファイルが存在する場合は、既に少なくとも 1 回はアプリケーションが実行されています。ファイルが存在し、ファイルに現在のバージョン番号よりも古いバージョン番号が含まれていれば、ユーザーが新しいバージョンを初めて実行していると判断できます。

この考え方を表した Flex の例を次に示します。

```
<?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";
                } 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>
```

次の例は、この考え方を JavaScript で表した場合を示しています。

```
<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.";
            }
            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>
```

アプリケーションでデータをローカル（アプリケーション記憶域ディレクトリなど）に保存している場合、初めて実行されたときに保存された（以前のバージョンの）データがあるかどうかを確認することができます。

アップデートフレームワークの使用

アプリケーションのアップデートの管理は煩雑になることがあります。**AdobeAIR** アプリケーション用アップデートフレームワークには、開発者が AIR の堅牢なアップデート機能を実現できる API が用意されています。AIR アップデートフレームワークでは、開発者向けに次のタスクを実行します。

- 一定の間隔に基づいて、またはユーザーの要求時に、定期的にアップデートをチェックします。

- Web ソースから AIR ファイル（アップデート）をダウンロードします。
- 新しくインストールされたバージョンが初めて実行されるときにユーザーに警告を通知します。
- ユーザーによるアップデートのチェックの必要性を確認します。
- 新しいアップデートバージョンの情報をユーザーに表示します。
- ダウンロードの進行状況とエラー情報をユーザーに表示します。

AIR アップデートフレームワークには、アプリケーションで使用できるサンプルユーザーインターフェイスが用意されています。このサンプルユーザーインターフェイスにより、アプリケーションのアップデートに関する基本情報と設定オプションがユーザーに示されます。アプリケーションでカスタムユーザーインターフェイスを定義して、アップデートフレームワークに使用することもできます。

AIR アップデートフレームワークを使用すると、AIR アプリケーションのアップデートバージョンに関する情報を単純な XML 設定ファイルに格納できます。ほとんどのアプリケーションでは、これらの設定ファイルに基本コードを含めるように設定することで、エンドユーザーが使いやすいアップデート機能を実現できます。

アップデートフレームワークを使用しない場合でも、Adobe AIR には、AIR アプリケーションの新しいバージョンへのアップグレードに使用できる Updater クラスが用意されています。Updater クラスを使用して、ユーザーのコンピューター上の AIR ファイルに含まれているバージョンにアプリケーションをアップグレードできます。ただし、アップグレードの管理では、ローカルに保存されている AIR ファイルに基づいてアプリケーションをアップデートする以外にも作業が必要になることがあります。

AIR アップデートフレームワークのファイル

AIR アップデートフレームワークは、AIR 2 SDK の frameworks/libs/air ディレクトリにあります。このフレームワークには、次のファイルが含まれています。

- applicationupdater.swc - ActionScript で使用するための、アップデートライブラリの基本機能が定義されています。このバージョンには、ユーザーインターフェイスは含まれていません。
- applicationupdater.swf - JavaScript で使用するための、アップデートライブラリの基本機能が定義されています。このバージョンには、ユーザーインターフェイスは含まれていません。
- applicationupdater_ui.swc - アプリケーションでアップデートオプションの表示に使用できるユーザーインターフェイスを含む、アップデートライブラリの基本機能（Flex 4 バージョン）が定義されています。
- applicationupdater_ui.swf - アプリケーションでアップデートオプションの表示に使用できるユーザーインターフェイスを含む、アップデートライブラリの基本機能（JavaScript バージョン）が定義されています。

詳しくは、次の節を参照してください。

- 261 ページの「Flex 開発環境の設定」
- 262 ページの「HTML ベースの AIR アプリケーションへのフレームワークファイルのインクルード」
- 262 ページの「基本のサンプル：ApplicationUpdaterUI バージョンの使用」

Flex 開発環境の設定

AIR 2 SDK の frameworks/libs/air ディレクトリにある SWC ファイルでは、Flex および Flash の開発で使用できるクラスが定義されています。

Flex SDK でのコンパイル時にアップデートフレームワークを使用するには、amxmlc コンパイラーの呼び出しに ApplicationUpdater.swc ファイルまたは ApplicationUpdater_UI.swc ファイルを含めます。次の例では、コンパイラーが Flex SDK ディレクトリの lib サブディレクトリに ApplicationUpdater.swc ファイルを読み込みます。

```
amxmlc -library-path+=lib/ApplicationUpdater.swc  -- myApp.mxml
```

次の例では、コンパイラーが Flex SDK ディレクトリの lib サブディレクトリに ApplicationUpdater_UI.swc ファイルを読み込みます。

```
amxmlc -library-path+=lib/ApplicationUpdater_UI.swc  -- myApp.mxml
```

Flash Builder を使用して開発している場合は、プロパティダイアログボックスで「Flex ビルドパス」設定の「ライブラリパス」タブに SWC ファイルを追加します。

SWC ファイルは、amxmlc コンパイラー（Flex SDK を使用）または Flash Builder で参照するディレクトリにコピーしてください。

HTML ベースの AIR アプリケーションへのフレームワークファイルのインクルード

アップデートフレームワークの frameworks\html ディレクトリには、次のファイルが含まれています。

- applicationupdater.swf - ユーザーインターフェイスを除く、アップデートライブラリの基本機能が定義されています。
- applicationupdater_ui.swf - アプリケーションでアップデートオプションの表示に使用できるユーザーインターフェイスを含む、アップデートライブラリの基本機能が定義されています。

AIR アプリケーションの JavaScript コードでは、SWF ファイルで定義されているクラスを使用できます。

アップデートフレームワークを使用するには、アプリケーションディレクトリ（またはサブディレクトリ）に applicationupdater.swf ファイルまたは applicationupdater_ui.swf ファイルを含めます。次に、そのファイルを読み込む script タグを、フレームワークを使用する HTML ファイルの JavaScript コードに含めます。

```
<script src="applicationUpdater.swf" type="application/x-shockwave-flash"/>
```

applicationupdater_ui.swf ファイルを読み込む場合は、次の script タグを使用します。

```
<script src="applicationupdater_ui.swf" type="application/x-shockwave-flash"/>
```

これらの 2 つのファイルで定義されている API については、このドキュメントの後半で説明します。

基本のサンプル：ApplicationUpdaterUI バージョンの使用

アップデートフレームワークの ApplicationUpdaterUI バージョンには、アプリケーションで簡単に使用できる基本インターフェイスが用意されています。以下に基本のサンプルを示します。

まず、アップデートフレームワークを呼び出す AIR アプリケーションを作成します。

1 アプリケーションが HTML ベースの AIR アプリケーションである場合は、applicationupdaterui.swf ファイルを読み込みます。

```
<script src="ApplicationUpdater_UI.swf" type="application/x-shockwave-flash"/>
```

2 AIR アプリケーションのプログラムロジックで、ApplicationUpdaterUI オブジェクトをインスタンス化します。

ActionScript では、次のコードを使用します。

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

JavaScript では、次のコードを使用します。

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

このコードを、アプリケーションが読み込まれたときに実行される初期化関数に追加できます。

3 updateConfig.xml という名前のテキストファイルを作成し、次のコードを追加します。

```
<?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>
```

updateConfig.xml ファイルの URL エレメントを、Web サーバー上のアップデート記述ファイルの最終的な場所に一致するように編集します（次の手順を参照してください）。

delay は、アプリケーションでアップデートをチェックする間隔を示す日数です。

4 AIR アプリケーションのプロジェクトディレクトリに updateConfig.xml ファイルを追加します。

5 Updater オブジェクトに updateConfig.xml ファイルを参照させ、このオブジェクトの initialize() メソッドを呼び出します。

ActionScript では、次のコードを使用します。

```
appUpdater.configurationFile = new File("app:/updateConfig.xml");
appUpdater.initialize();
```

JavaScript では、次のコードを使用します。

```
appUpdater.configurationFile = new air.File("app:/updateConfig.xml");
appUpdater.initialize();
```

6 最初のアプリケーションとは異なるバージョンを持つ、もう 1 つの AIR アプリケーションを作成します（バージョンはアプリケーション記述ファイルの version エレメントで指定されます）。

次に、AIR アプリケーションのアップデートバージョンを Web サーバーに追加します。

1 AIR ファイルのアップデートバージョンを Web サーバーに配置します。

2 updateDescriptor.2.5.xml という名前のテキストファイルを作成し、次の内容を追加します。

```
<?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>
```

updateDescriptor.xml ファイルの versionNumber、URL および description を、アップデート AIR ファイルに一致するように編集します。このアップデート記述形式は、AIR 2.5 SDK 以降に含まれるアップデートフレームワークを使用するアプリケーションで使用されます。

3 updateDescriptor.1.0.xml という名前のテキストファイルを作成し、次の内容を追加します。

```
<?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>
```

updateDescriptor.xml ファイルの version、URL および description を、アップデート AIR ファイルに一致するように編集します。このアップデート記述形式は、AIR 2 SDK 以前に含まれるアップデートフレームワークを使用するアプリケーションで使用されます。

注意：この 2 つ目のアップデート記述ファイルの作成は、AIR 2.5 より前に作成されたアプリケーションへのアップデートをサポートする場合にのみ必要です。

4 updateDescriptor.2.5.xml および updateDescriptor.1.0.xml ファイルを、アップデート AIR ファイルがある同じ Web サーバーディレクトリに追加します。

これは基本のサンプルですが、多くのアプリケーションに十分なアップデート機能を提供できます。このドキュメントの後半では、ニーズに最も適したアップデートフレームワークの使用方法について説明します。

アップデートフレームワークの使用方法を示す別の例については、Adobe AIR デベロッパーセンターで次のサンプルアプリケーションを参照してください。

- Flash ベースアプリケーションにおけるアップデートフレームワーク
  （http://www.adobe.com/go/learn_air_qs_update_framework_flash_jp）

AIR 2.5 へのアップデート

アプリケーションへのバージョン番号の割り当て規則が AIR 2.5 で変更されたので、AIR 2 アップデートフレームワークは、AIR 2.5 アプリケーション記述子のバージョン情報を解析できません。互換性がないので、AIR 2.5 を使用するようにアプリケーションを更新する前に、アプリケーションが新しいアップデートフレームワークを使用するように変更する必要があります。したがって、2.5 より前のバージョンの AIR から、AIR 2.5 以降のアプリケーションに更新するには、2 つのアップデートが必要です。最初のアップデートでは、AIR 2 名前空間を使用して、AIR 2.5 アップデートフレームワークを含めます（引き続き、AIR 2.5 SDK を使用するアプリケーションパッケージを作成できます）。2 つ目のアップデートで、AIR 2.5 名前空間を使用して、アプリケーションの新しい機能を含めることができます。

AIR Updater クラスディレクトリを使用する AIR 2.5 アプリケーションへのアップデート以外を実行しない、中間アップデートを含めることもできます。

次の例に、バージョン 1.0 から 2.0 にアプリケーションをアップデートする方法を示します。バージョン 1.0 では古い 2.0 名前空間を使用しています。バージョン 2.0 では、2.5 名前空間を使用しており、AIR 2.5 API を使用して新しい機能が実装されています。

1. アプリケーションのバージョン 1.0 に基づいて、アプリケーションの中間バージョンとして、バージョン 1.0.1 を作成します。
   a. アプリケーションを作成するときは、AIR 2.5 Application Updater フレームワークを使用します。

      注意：Flash テクノロジに基づく AIR アプリケーションの場合は applicationupdater.swc または applicationupdater_ui.swc を、HTML ベースの AIR アプリケーションの場合は applicationupdater.swf または applicationupdater_ui.swf を使用してください。

   b. 次のように古い名前空間とバージョンを使用して、バージョン 1.0.1 用のアップデート記述ファイルを作成します。

```
<?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. AIR 2.5 API と 2.5 名前空間を使用するバージョン 2.0 のアプリケーションを作成します。
3. アップデート記述ファイルを作成し、アプリケーションを 1.0.1 バージョンから 2.0 バージョンにアップデートします。

```
<?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>
```

アップデート記述ファイルの定義と Web サーバーへの AIR ファイルの追加

AIR アップデートフレームワークを使用するときは、使用できるアップデートに関する基本情報を、Web サーバーに格納されるアップデート記述ファイルに定義します。アップデート記述ファイルは単純な XML ファイルです。アプリケーションに含まれているアップデートフレームワークは、このファイルをチェックして、新しいバージョンがアップロードされているかどうかを調べます。

アップデート記述ファイルの形式は、AIR 2.5 で変更されました。新しい形式では、別の名前空間が使用されます。元の名前空間は、「http://ns.adobe.com/air/framework/update/description/1.0」です。AIR 2.5 の名前空間は、「http://ns.adobe.com/air/framework/update/description/2.5」です。

AIR 2.5 より前に作成された AIR アプリケーションは、バージョン 1.0 アップデート記述子のみを読み取ることができます。AIR 2.5 以降に含まれるアップデータフレームワークを使用して作成された AIR アプリケーションは、バージョン 2.5 アップデート記述子のみを読み取ることができます。バージョンの互換性がないので、2 つのアップデート記述ファイルを作成することが必要になる場合があります。AIR 2.5 バージョンのアプリケーションのアップデートロジックは、新しい形式を使用するアップデート記述子をダウンロードする必要があります。以前のバージョンの AIR アプリケーションでは、元の形式を使用し続ける必要があります。どちらのファイルも、アップデートをリリースするたびに変更する必要があります（AIR 2.5 より前に作成されたバージョンのサポートを停止するまで）。

アップデート記述ファイルには次のデータが含まれます。

- versionNumberversion：新しいバージョンの AIR アプリケーションです。AIR 2.5 アプリケーションのアップデートに使用されるアップデート記述子の versionNumber エレメントを使用します。値は、新しい AIR アプリケーション記述ファイルの versionNumber エレメントで使用されるストリングと同じであることが必要です。アップデート記述ファイルのバージョン番号がアップデート AIR ファイルのバージョン番号と一致しない場合、アップデートフレームワークは例外をスローします。
- version：AIR アプリケーションの新しいバージョンです。AIR 2.5 より前に作成されたアプリケーションのアップデートに使用されるアップデート記述子の version エレメントを使用します。値は、新しい AIR アプリケーション記述ファイルの version エレメントで使用されるストリングと同じであることが必要です。アップデート記述ファイルのバージョンがアップデート AIR ファイルのバージョンと一致しない場合、アップデートフレームワークは例外をスローします。
- versionLabel：ユーザーに表示する、人が読めるバージョンストリングです。versionLabel は省略可能ですが、バージョン 2.5 アップデート記述ファイルでのみ指定できます。アプリケーション記述子で versionLabel を使用し、同じ値に設定する場合にのみ使用します。
- url：アップデート AIR ファイルの場所です。これは、アップデートバージョンの AIR アプリケーションが含まれるファイルです。
- description：新しいバージョンに関する詳細です。この情報を、アップデートプロセスの間にユーザーに表示できます。

version エレメントと url エレメントは必須です。description エレメントはオプションです。

次にバージョン 2.5 アップデート記述ファイルのサンプルを示します。

```
<?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>
```

次にバージョン 1.0 アップデート記述ファイルのサンプルを示します。

```
<?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>
```

複数の言語を使用して description タグを定義する必要がある場合は、複数の text エレメントを使用して lang 属性を定義します。

```
<?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>
```

アップデート記述ファイルをアップデート AIR ファイルと共に Web サーバーに配置します。

アップデートフレームワークの templates ディレクトリには、サンプルのアップデート記述ファイルがあります。このサンプルには、単一言語と複数言語の両方のバージョンが含まれています。

Updater オブジェクトのインスタンス化

コードに AIR アップデートフレームワークを読み込んだ後（261 ページの「Flex 開発環境の設定」および 262 ページの「HTML ベースの AIR アプリケーションへのフレームワークファイルのインクルード」を参照）、次のように Updater オブジェクトをインスタンス化する必要があります。

ActionScript の例：

```
var appUpdater:ApplicationUpdater = new ApplicationUpdater();
```

JavaScript の例：

```
var appUpdater = new runtime.air.update.ApplicationUpdater();
```

前述のコードでは、（ユーザーインターフェイスを含まない）ApplicationUpdater クラスを使用しています。（ユーザーインターフェイスを提供する）ApplicationUpdaterUI クラスを使用する場合は、次のコードを使用します。

ActionScript の例：

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

JavaScript の例：

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

このドキュメントの以降のコード例は、appUpdater という名前の Updater オブジェクトをインスタンス化済みであることを前提としています。

アップデート設定の構成

ApplicationUpdater と ApplicationUpdaterUI はいずれも、アプリケーションに付属する設定ファイルを使用するか、アプリケーションで ActionScript または JavaScript を使用して設定できます。

XML 設定ファイルでのアップデート設定の定義

アップデート設定ファイルは XML ファイルです。このファイルには、次のエレメントを含めることができます。

- updateURL - String 値です。リモートサーバー上のアップデート記述ファイルの場所を表します。任意の有効な URLRequest の場所を指定できます。updateURL プロパティは、設定ファイルまたはスクリプトで定義する必要があります（265 ページの「アップデート記述ファイルの定義と Web サーバーへの AIR ファイルの追加」を参照してくださ

い）。このプロパティは、Updater を使用する前（Updater オブジェクトの initialize() メソッドを呼び出す前。詳しくは、269 ページの「アップデートフレームワークの初期化」を参照してください）に定義する必要があります。

- delay - Number 値です。アップデートをチェックする間隔（日数）を表します（0.25 のような値を指定できます）。値 0（デフォルト値）は、Updater で定期的なチェックが自動的に行われないことを示します。

ApplicationUpdaterUI の設定ファイルには、updateURL エレメントと delay エレメントに加えて次のエレメントを含めることができます。

- defaultUI - dialog エレメントのリストです。各 dialog エレメントには、ユーザーインターフェイスのダイアログボックスに対応する name 属性があります。各 dialog エレメントには、ダイアログボックスを表示するかどうかを定義する visible 属性があります。デフォルト値は true です。name 属性で使用できる値は次のとおりです。
  - checkForUpdate - アップデートの有無をチェックダイアログボックス、アップデートなしダイアログボックスおよびアップデートエラーダイアログボックスに対応
  - downloadUpdate - アップデートのダウンロードダイアログボックスに対応
  - downloadProgress - ダウンロードの進捗状況ダイアログボックスとダウンロードエラーダイアログボックスに対応
  - installUpdate - アップデートのインストールダイアログボックスに対応
  - fileUpdate - ファイルのアップデートダイアログボックス、ファイルのアップデートなしダイアログボックスおよびファイルエラーダイアログボックスに対応
- unexpectedError - 予期しないエラーダイアログボックスに対応

  false に設定した場合、対応するダイアログボックスはアップデート時に表示されません。

次に ApplicationUpdater フレームワークの設定ファイルの例を示します。

```
<?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>
```

次に、defaultUI エレメントの定義を含む、ApplicationUpdaterUI フレームワークの設定ファイルの例を示します。

```
<?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>
```

configurationFile プロパティでは、次に示すように設定ファイルの場所を指定します。

ActionScript の例：

```
appUpdater.configurationFile = new File("app:/cfg/updateConfig.xml");
```

JavaScript の例：

```
appUpdater.configurationFile = new air.File("app:/cfg/updateConfig.xml");
```

アップデートフレームワークの templates ディレクトリには、config-template.xml という名前のサンプルの設定ファイルがあります。

アップデート設定の ActionScript コードまたは JavaScript コードの定義

これらの設定パラメーターは、次に示すようにアプリケーションのコードを使用して設定することもできます。

```
appUpdater.updateURL = " http://example.com/updates/update.xml";
appUpdater.delay = 1;
```

Updater オブジェクトのプロパティは updateURL および delay です。これらのプロパティでは、設定ファイルの updateURL エレメントおよび delay エレメントと同じ設定（アップデート記述ファイルの URL およびアップデートのチェックの間隔）を定義します。設定ファイルおよびコード内の設定を両方とも指定した場合は、コードで設定されたプロパティが、設定ファイル内の対応する設定より優先されます。

updateURL プロパティは、設定ファイルまたはスクリプトで定義する必要があります（265 ページの「アップデート記述ファイルの定義と Web サーバーへの AIR ファイルの追加」を参照してください）。このプロパティは、Updater を使用する前、つまり Updater オブジェクトの initialize() メソッドを呼び出す前（269 ページの「アップデートフレームワークの初期化」を参照してください）に定義する必要があります。

ApplicationUpdaterUI フレームワークでは、Updater オブジェクトの次のような追加のプロパティを定義します。

- isCheckForUpdateVisible - アップデートの有無をチェックダイアログボックス、アップデートなしダイアログボックスおよびアップデートエラーダイアログボックスに対応
- isDownloadUpdateVisible - アップデートのダウンロードダイアログボックスに対応
- isDownloadProgressVisible - ダウンロードの進捗状況ダイアログボックスとダウンロードエラーダイアログボックスに対応
- isInstallUpdateVisible - アップデートのインストールダイアログボックスに対応
- isFileUpdateVisible - ファイルのアップデートダイアログボックス、ファイルのアップデートなしダイアログボックスおよびファイルエラーダイアログボックスに対応
- isUnexpectedErrorVisible - 予期しないエラーダイアログボックスに対応

各プロパティは、ApplicationUpdaterUI ユーザーインターフェイスの 1 つ以上のダイアログボックスに対応しています。各プロパティは Boolean 値で、デフォルト値は true です。false に設定した場合、対応するダイアログボックスはアップデート中に表示されません。

これらのダイアログボックスのプロパティは、アップデート設定ファイル内の設定をオーバーライドします。

アップデートプロセス

AIR アップデートフレームワークでは、次のステップでアップデートプロセスが実行されます。

1. Updater の初期化により、delay で定義されている間隔内にアップデートのチェックが実行されたかどうかが確認されます（266 ページの「アップデート設定の構成」を参照してください）。アップデートのチェックが必要な場合は、アップデートプロセスが続行されます。
2. アップデート記述ファイルがダウンロードされ、解釈されます。
3. アップデート AIR ファイルがダウンロードされます。
4. アプリケーションのアップデートバージョンがインストールされます。

これらの各ステップが完了するたびに、Updater オブジェクトからイベントが送出されます。ApplicationUpdater バージョンでは、プロセスのステップが正常に完了したことを示すイベントをキャンセルできます。これらのイベントのいずれかをキャンセルすると、プロセスのその次のステップがキャンセルされます。ApplicationUpdaterUI バージョンでは、プロセスのステップごとに、プロセスをキャンセルするか進行させるかをユーザーが選択できるダイアログボックスが表示されます。

イベントをキャンセルした場合、Updater オブジェクトのメソッドを呼び出すとプロセスを再開できます。

ApplicationUpdater バージョンでアップデートプロセスが進行するときは、現在の状態が currentState プロパティに記録されます。このプロパティは、次のいずれかの値のストリングに設定されます。

- "UNINITIALIZED" - Updater は初期化されていません。

- "INITIALIZING" - Updater の初期化中です。
- "READY" - Updater の初期化が完了しました。
- "BEFORE_CHECKING" - アップデート記述ファイルはまだチェックされていません。
- "CHECKING" - アップデート記述ファイルのチェック中です。
- "AVAILABLE" - アップデート記述ファイルは使用可能です。
- "DOWNLOADING" - AIR ファイルのダウンロード中です。
- "DOWNLOADED" - AIR ファイルのダウンロードが完了しました。
- "INSTALLING" - AIR ファイルのインストール中です。
- "PENDING_INSTALLING" - Updater の初期化は完了し、保留中のアップデートがあります。

Updater オブジェクトのメソッドの中には、Updater が特定の状態の場合にのみ実行されるものもあります。

アップデートフレームワークの初期化

アップデートを初期化するには、設定プロパティを設定した後に（262 ページの「基本のサンプル：ApplicationUpdaterUI バージョンの使用」を参照してください）initialize() メソッドを呼び出します。

```
appUpdater.initialize();
```

このメソッドは、次の処理を行います。

- アップデートフレームワークを初期化し、保留中のアップデートがあればそれを同期的にサイレントインストールします。このメソッドを呼び出すとアプリケーションが再起動される場合があるため、アプリケーションの起動時にこのメソッドを呼び出す必要があります。
- 延期されている更新があるかどうかをチェックし、ある場合はインストールします。
- アップデートプロセス中にエラーが発生した場合は、アップデートファイルとバージョン情報をアプリケーションのストレージ領域から消去します。
- delay の間隔が過ぎている場合は、アップデートプロセスを開始します。それ以外の場合はタイマーを再起動します。

このメソッドを呼び出すと、Updater オブジェクトから次のイベントが送出されます。

- UpdateEvent.INITIALIZED - 初期化が完了したときに送出されます。
- ErrorEvent.ERROR - 初期化中にエラーが発生したときに送出されます。

UpdateEvent.INITIALIZED イベントが送出されると、アップデートプロセスが完了します。

initialize() メソッドを呼び出すと、タイマーの delay の設定に基づいてアップデートプロセスが開始され、すべてのステップが実行されます。しかし、Updater オブジェクトの checkNow() メソッドを呼び出すことにより、任意の時点でアップデートプロセスを開始することもできます。

```
appUpdater.checkNow();
```

アップデートプロセスが既に実行中の場合は、このメソッドを呼び出しても何も起こりません。それ以外の場合は、アップデートプロセスが開始されます。

checkNow() メソッドを呼び出すと、Updater オブジェクトから次のイベントが送出されます。

- UpdateEvent.CHECK_FOR_UPDATE - アップデート記述ファイルのダウンロードを試みる直前に送出されます。

checkForUpdate イベントをキャンセルした場合、Updater オブジェクトの checkForUpdate() メソッドを呼び出すことができます（次の節を参照してください）。このイベントをキャンセルしない場合、アップデートプロセスが進行し、アップデート記述ファイルがチェックされます。

ApplicationUpdaterUI バージョンでのアップデートプロセスの管理

ApplicationUpdaterUI バージョンでは、ユーザーはユーザーインターフェイスのダイアログボックスにある「キャンセル」を使用してプロセスをキャンセルできます。また、ApplicationUpdaterUI オブジェクトの cancelUpdate() メソッドを呼び出して、アップデートプロセスをプログラムによってキャンセルすることもできます。

ApplicationUpdaterUI オブジェクトのプロパティを設定するか、アップデート設定ファイルでエレメントを定義して、Updater に表示される確認ダイアログボックスを指定します。詳しくは、266 ページの「アップデート設定の構成」を参照してください。

ApplicationUpdater バージョンでのアップデートプロセスの管理

ApplicationUpdater オブジェクトによって送出されるイベントオブジェクトの preventDefault() メソッドを呼び出して、アップデートプロセスのステップをキャンセルできます（268 ページの「アップデートプロセス」を参照）。デフォルト動作をキャンセルすると、アプリケーションでは続行するかどうかをユーザーに確認するメッセージを表示できます。

以降の節では、プロセスのステップがキャンセルされたときにアップデートプロセスを続行する方法について説明します。

アップデート記述ファイルのダウンロードと解釈

アップデートプロセスが開始される前（Updater がアップデート記述ファイルのダウンロードを試みる直前）に、ApplicationUpdater オブジェクトから checkForUpdate イベントが送出されます。checkForUpdate イベントのデフォルト動作をキャンセルすると、アップデート記述ファイルはダウンロードされません。checkForUpdate() メソッドを呼び出すと、アップデートプロセスを再開できます。

```
appUpdater.checkForUpdate();
```

checkForUpdate() メソッドを呼び出すと、アップデート記述ファイルが非同期でダウンロードおよび解釈されます。checkForUpdate() メソッドを呼び出すと、Updater オブジェクトから次のイベントが送出されます。

- StatusUpdateEvent.UPDATE_STATUS - アップデート記述ファイルのダウンロードと解釈が正常に完了しました。このイベントには、次のプロパティがあります。
  - available - Boolean 値。現在のアプリケーションと異なるバージョンが使用可能である場合は true に設定され、それ以外の場合（バージョンが同じである場合）は false に設定されます。
  - version - String 値。アップデートファイルのアプリケーション記述ファイル内のバージョンです。
  - details - Array 値。ローカライズされたバージョンの説明が存在しない場合、この配列の 1 番目のエレメントで空のストリング（""）が返され、2 番目のエレメントで説明が返されます。

    （アップデート記述ファイルに）複数のバージョンの説明が存在する場合、この配列には複数のサブ配列が含まれます。各配列には 2 つのエレメントが含まれます。1 番目のエレメントは言語コード（"en" など）で、2 番目のエレメントはその言語の対応する説明（String 値）です。265 ページの「アップデート記述ファイルの定義と Web サーバーへの AIR ファイルの追加」を参照してください。
- StatusUpdateErrorEvent.UPDATE_ERROR - エラーが発生したので、アップデート記述ファイルをダウンロードまたは解釈できませんでした。

アップデート AIR ファイルのダウンロード

Updater がアップデート記述ファイルを正常にダウンロードおよび解釈した後、ApplicationUpdater オブジェクトから updateStatus イベントが送出されます。デフォルト動作では、アップデートファイル（使用可能な場合）のダウンロードが開始されます。デフォルト動作をキャンセルすると、downloadUpdate() メソッドを呼び出してアップデートプロセスを再開できます。

```
appUpdater.downloadUpdate();
```

このメソッドを呼び出すと、AIR ファイルのアップデートバージョンが非同期でダウンロードされます。

downloadUpdate() メソッドからは次のイベントが送出されます。

- UpdateEvent.DOWNLOAD_START - サーバーとの接続が確立されました。ApplicationUpdaterUI ライブラリを使用している場合は、このイベントにより、ダイアログボックスにダウンロードの進捗状況を追跡するためのプログレスバーが表示されます。
- ProgressEvent.PROGRESS - ファイルのダウンロードの進行中に定期的に送出されます。
- DownloadErrorEvent.DOWNLOAD_ERROR - 接続中またはアップデートファイルのダウンロード中にエラーが発生した場合に送出されます。HTTP ステータスが無効（「404 - ファイルが見つかりません」など）である場合にも送出されます。このイベントには errorID プロパティがあります。このプロパティは、追加のエラー情報を定義する整数です。追加の subErrorID プロパティには、さらに詳しいエラー情報が含まれます。
- UpdateEvent.DOWNLOAD_COMPLETE - アップデート記述ファイルのダウンロードおよび解釈が正常に完了しました。このイベントをキャンセルしない場合、ApplicationUpdater バージョンはアップデートバージョンのインストールに進みます。ApplicationUpdaterUI バージョンでは、処理を進行させるかどうかをユーザーが選択できるダイアログボックスが表示されます。

アプリケーションのアップデート

アップデートファイルのダウンロードが完了すると、ApplicationUpdater オブジェクトから downloadComplete イベントが送出されます。デフォルト動作をキャンセルすると、installUpdate() メソッドを呼び出してアップデートプロセスを再開できます。

```
appUpdater.installUpdate(file);
```

このメソッドを呼び出すと、AIR ファイルのアップデートバージョンがインストールされます。このメソッドには file というパラメーターが 1 つあります。このパラメーターは、アップデートとして使用する AIR ファイルを参照する File オブジェクトです。

ApplicationUpdater オブジェクトから beforeInstall イベントが送出されるのは、installUpdate() メソッドを呼び出した後です。

- UpdateEvent.BEFORE_INSTALL - アップデートのインストールの直前に送出されます。この時点でアップデートのインストールをキャンセルすると、ユーザーが現在の作業を終えてからアップデートを実行できるので、場合によっては便利です。Event オブジェクトの preventDefault() メソッドを呼び出すと、次に再起動するまでインストールが延期され、新しいアップデートプロセスも開始されません（これには、checkNow() メソッドの呼び出しによるアップデートと定期的なチェックによるアップデートの両方が含まれます）。

任意の AIR ファイルからのインストール

installFromAIRFile() メソッドを呼び出すと、ユーザーのコンピューターの AIR ファイルからアップデートバージョンをインストールできます。

```
appUpdater.installFromAIRFile();
```

このメソッドにより、AIR ファイルからアプリケーションのアップデートバージョンがインストールされます。

installFromAIRFile() メソッドからは次のイベントが送出されます。

- StatusFileUpdateEvent.FILE_UPDATE_STATUS - installFromAIRFile() メソッドを使用して送信されたファイルが ApplicationUpdater で正常に検証された後に送出されます。このイベントには、次のプロパティがあります。
  - available - 現在のアプリケーションと異なるバージョンが使用可能である場合は true に設定され、それ以外の場合（バージョンが同じである場合）は false に設定されます。
  - version - 使用できる新しいバージョンを表すストリングです。

- path - アップデートファイルのネイティブパスを表します。

StatusFileUpdateEvent オブジェクトの available プロパティが true に設定されている場合でも、このイベントをキャンセルできます。このイベントをキャンセルすると、アップデートの進行がキャンセルされます。キャンセルされたアップデートを続行するには、installUpdate() メソッドを呼び出します。

- StatusFileUpdateErrorEvent.FILE_UPDATE_ERROR - エラーが発生したので、AIR アプリケーションをインストールできませんでした。

アップデートプロセスのキャンセル

cancelUpdate() メソッドを呼び出すと、アップデートプロセスをキャンセルできます。

```
appUpdater.cancelUpdate();
```

このメソッドにより、保留中のダウンロードがあればキャンセルされ、ダウンロードされた不完全なファイルがあれば削除され、定期的なチェックのタイマーが再起動されます。

Updater オブジェクトの初期化中は、このメソッドを呼び出しても何も起こりません。

ApplicationUpdaterUI のインターフェイスのローカライズ

ApplicationUpdaterUI クラスには、アップデートプロセス用のデフォルトのユーザーインターフェイスが用意されています。このインターフェイスには、ユーザーがプロセスの起動、プロセスのキャンセルおよびその他の関連アクションの実行に使用できるダイアログボックスが含まれています。

アップデート記述ファイルの description エレメントでは、アプリケーションの説明を複数の言語で定義できます。次に示すように、複数の text エレメントを使用して lang 属性を定義します。

```
<?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>
```

アップデートフレームワークは、エンドユーザーのロケールチェーンに最も適した説明を使用します。詳しくは、「アップデート記述ファイルの定義と Web サーバーへの AIR ファイルの追加」を参照してください。

Flex 開発者は、ApplicationUpdaterDialogs バンドルに直接新しい言語を追加できます。

JavaScript 開発者は、Updater オブジェクトの addResources() メソッドを呼び出すことができます。このメソッドにより、言語の新しいリソースバンドルが動的に追加されます。リソースバンドルでは、言語のローカライズされたストリングが定義されます。これらのストリングは、ダイアログボックスの様々なテキストフィールドで使用されます。

JavaScript 開発者は ApplicationUpdaterUI クラスの localeChain プロパティを使用して、ユーザーインターフェイスで使用されるロケールチェーンを定義できます。通常、JavaScript（HTML）開発者のみがこのプロパティを使用します。Flex 開発者は ResourceManager を使用してロケールチェーンを管理できます。

例えば、次の JavaScript コードはルーマニア語およびハンガリー語のリソースバンドルを定義します。

```
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;
```

詳しくは、リファレンスガイドの ApplicationUpdaterUI クラスの addResources() メソッドに関する説明を参照してください。

第 18 章：ソースコードの表示

Web ブラウザーで HTML ページのソースコードを表示できるのと同じように、ユーザーは HTML ベースの AIR アプリケーションのソースコードを表示できます。Adobe® AIR® SDK には、AIRSourceViewer.js という JavaScript ファイルが含まれています。このファイルをアプリケーションで使用すると、ソースコードをエンドユーザーに簡単に表示できます。

ソースビューアの読み込み、設定、および開始

ソースビューアのコードは、AIR SDK の frameworks ディレクトリにある AIRSourceViewer.js という JavaScript ファイルに含まれています。アプリケーションでソースビューアを使用するには、AIRSourceViewer.js をアプリケーションのプロジェクトディレクトリにコピーし、アプリケーションのメインの HTML ファイルでスクリプトタグを使用してファイルを読み込みます。

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

AIRSourceViewer.js ファイルは SourceViewer クラスを定義します。このクラスに JavaScript コードからアクセスするには、air.SourceViewer を呼び出します。

SourceViewer クラスは、getDefault()、setup()、および viewSource() の 3 つのメソッドを定義します。

メソッド	説明
getDefault()	静的メソッド。他のメソッドの呼び出しに使用できる SourceViewer インスタンスを返します。
setup()	設定をソースビューアに適用します。詳しくは、274 ページの「ソースビューアの設定」を参照してください。
viewSource()	ユーザーがホストアプリケーションのソースファイルを参照して開くことができる新しいウィンドウを開きます。

注意：ソースビューアを使用するコードは、アプリケーションセキュリティサンドボックス内（アプリケーションディレクトリ内のファイル）に含まれている必要があります。

例えば、次の JavaScript コードでは、Source Viewer オブジェクトをインスタンス化して、すべてのソースファイルをリスト表示するソースビューアウィンドウを開きます。

```
var viewer = air.SourceViewer.getDefault();
viewer.viewSource();
```

ソースビューアの設定

config() メソッドは特定の設定をソースビューアに適用します。このメソッドは 1 つのパラメーター、configObject を取得します。configObject オブジェクトには、ソースビューアの設定を定義するプロパティがあります。このプロパティは、default、exclude、initialPosition、modal、typesToRemove、typesToAdd です。

default

ソースビューアに表示される初期ファイルの相対パスを指定するストリング。

例えば、次の JavaScript コードでは、初期ファイルとして index.html ファイルが表示されるソースビューアウィンドウが開きます。

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.default = "index.html";
viewer.viewSource(configObj);
```

exclude

ソースビューアのリストから除外されるファイルまたはディレクトリを指定するストリングの配列。このパスはアプリケーションディレクトリに対する相対パスです。ワイルドカード文字はサポートされません。

例えば、次の JavaScript コードでは AIRSourceViewer.js ファイルと、Images および Sounds サブディレクトリ内のファイルを除くすべてのソースファイルをリスト表示するソースビューアウィンドウが開きます。

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.exclude = ["AIRSourceViewer.js", "Images" "Sounds"];
viewer.viewSource(configObj);
```

initialPosition

ソースビューアウィンドウの初期 x 座標と初期 y 座標を指定する 2 つの数値を含む配列。

例えば、次の JavaScript コードはスクリーン座標 [40, 60] (X = 40, Y = 60) にソースビューアウィンドウを開きます。

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.initialPosition = [40, 60];
viewer.viewSource(configObj);
```

modal

ソースビューアをモーダル（true）ウィンドウと非モーダル（false）ウィンドウのどちらにするかを指定するブール値。デフォルトではソースビューアはモーダルです。

例えば、次の JavaScript コードは、ソースビューアウィンドウを開き、ソースビューアウィンドウおよび他のアプリケーションウィンドウの両方で操作できるようにします。

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.modal = false;
viewer.viewSource(configObj);
```

typesToAdd

デフォルトで含まれる種類に加えてソースビューアのリストに含めるファイルの種類を指定するストリングの配列。

デフォルトでは、ソースビューアで次のファイルの種類がリスト表示されます。

- テキストファイル — TXT、XML、MXML、HTM、HTML、JS、AS、CSS、INI、BAT、PROPERTIES、CONFIG
- 画像ファイル — JPG、JPEG、PNG、GIF

  値を指定しない場合、デフォルトの種類がすべて含まれます（typesToExclude プロパティで指定されたファイルの種類は除外されます）。

  例えば、次の JavaScript コードでは VCF ファイルと VCARD ファイルを含むソースビューアウィンドウが開きます。

  ```
  var viewer = air.SourceViewer.getDefault();
  var configObj = {};
  configObj.typesToAdd = ["text.vcf", "text.vcard"];
  viewer.viewSource(configObj);
  ```

  リスト表示するファイルの種類ごとに、「text」（テキストファイルの場合）または「image」（画像ファイルの場合）を指定する必要があります。

typesToExclude

ソースビューアから除外するファイルの種類を指定するストリングの配列。

デフォルトでは、ソースビューアで次のファイルの種類がリスト表示されます。

- テキストファイル — TXT、XML、MXML、HTM、HTML、JS、AS、CSS、INI、BAT、PROPERTIES、CONFIG
- 画像ファイル — JPG、JPEG、PNG、GIF

  例えば、次の JavaScript コードでは、GIF ファイルまたは XML ファイルがリスト表示されずにソースビューアウィンドウが開きます。

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToExclude = ["image.gif", "text.xml"];
viewer.viewSource(configObj);
```

  リスト表示するファイルの種類ごとに、「text」（テキストファイルの場合）または「image」（画像ファイルの場合）を指定する必要があります。

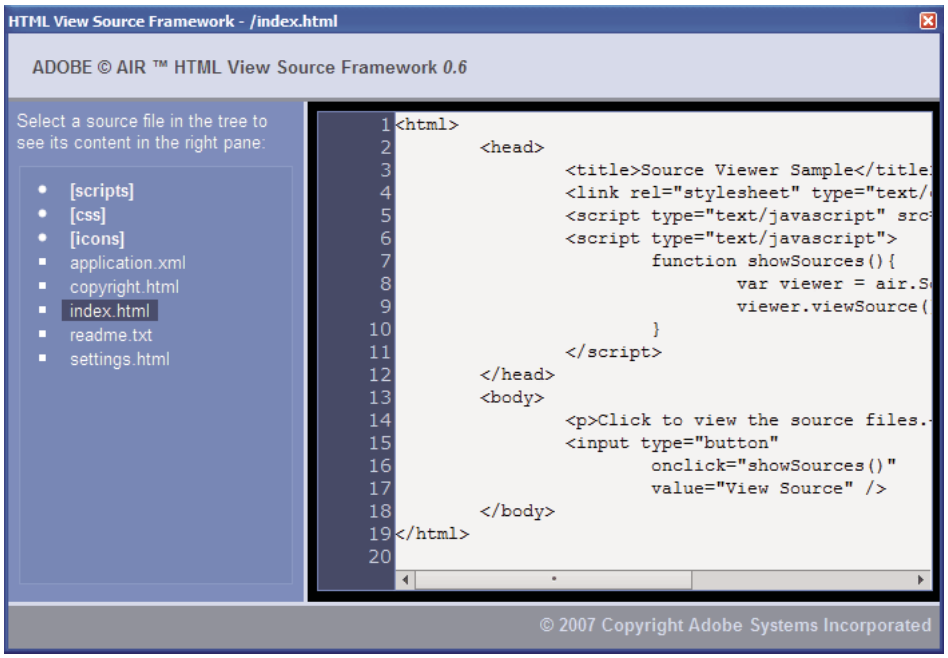
ソースビューアの開始

リンク、ボタン、メニューコマンドなどのユーザーインターフェイスエレメントを含めて、ユーザーがそのインターフェイスを選択するとソースビューアコードが呼び出されるようにする必要があります。例えば、次の簡単なアプリケーションでは、ユーザーがリンクをクリックするとソースビューアが開きます。

```
<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>
```

ソースビューアのユーザーインターフェイス

アプリケーションが SourceViewer オブジェクトの viewSource() メソッドを呼び出すと、AIR アプリケーションでソースビューアウィンドウが開きます。このウィンドウには、ソースファイルとディレクトリのリスト（左側）、および選択したファイルのソースコードを示す表示領域（右側）があります。



ディレクトリは、括弧内に示されます。ユーザーは括弧をクリックしてディレクトリのリストを展開または折りたたむことができます。

ソースビューアでは、認識される拡張子（HTML、JS、TXT、XML など）の付いたテキストファイルのソース、および認識される画像拡張子（JPG、JPEG、PNG、および GIF）の付いた画像ファイルのソースを表示できます。認識されない種類の拡張子が付いたファイルをユーザーが選択した場合、エラーメッセージが表示されます（「Cannot retrieve text content from this filetype」）。

setup() メソッドを使用して除外されたソースファイルはリスト表示されません（274 ページの「ソースビューアの読み込み、設定、および開始」を参照）。

第 19 章：AIR HTML Introspector によるデバッグ

Adobe® AIR® SDK には、AIRIntrospector.js という JavaScript ファイルが用意されています。これをアプリケーションに含めると、HTML ベースのアプリケーションのデバッグに役立ちます。

AIR Introspector について

Adobe AIR HTML/JavaScript Application Introspector（AIR HTML Introspector）には、HTML ベースのアプリケーションの開発とデバッグに役立つ次のような機能があります。

- イントロスペクターツールでは、アプリケーション内のユーザーインターフェイスエレメントを参照して、そのマークアップと DOM プロパティを確認できます。
- オブジェクト参照をイントロスペクション用に送信するコンソールが用意されており、プロパティ値を調整して JavaScript コードを実行できます。また、オブジェクトをコンソールに対して直列化すると、データの編集を制限できます。コンソールからテキストをコピーしたり保存したりすることもできます。
- DOM プロパティと関数をツリー表示できます。
- DOM エレメントの属性とテキストノードを編集できます。
- アプリケーションに読み込まれたリンク、CSS スタイル、イメージおよび JavaScript ファイルを一覧表示します。
- ユーザーインターフェイスの初期 HTML ソースと現在のマークアップソースを表示できます。
- アプリケーションディレクトリ内のファイルにアクセスできます（この機能は、アプリケーションサンドボックス用に開かれた AIR HTML Introspector コンソールでのみ使用できます。非アプリケーションサンドボックスのコンテンツ用に開かれたコンソールでは使用できません）。
- responseText プロパティや responseXML プロパティなど（使用可能な場合）、XMLHttpRequest オブジェクトとそのプロパティのビューアが含まれています。
- ソースコードとファイルから一致するテキストを検索できます。

AIR Introspector コードの読み込み

AIR Introspector コードは、AIR SDK の frameworks ディレクトリにある AIRIntrospector.js という JavaScript ファイルに格納されています。アプリケーションで AIR Introspector を使用するには、AIRIntrospector.js をアプリケーションのプロジェクトディレクトリにコピーし、アプリケーションのメイン HTML ファイルでスクリプトタグを使用してファイルを読み込みます。

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

また、アプリケーションの各ネイティブウィンドウに対応するすべての HTML ファイルにもこのファイルを含めます。

重要：AIRIntrospector.js ファイルは、アプリケーションの開発時とデバッグ時にのみ含めてください。AIR アプリケーションをパッケージ化して配布する際は削除してください。

AIRIntrospector.js ファイルでは Console クラスを定義しています。このクラスに JavaScript コードからアクセスするには、air.Introspector.Console を呼び出します。

注意：AIR Introspector を使用するコードは、アプリケーションセキュリティサンドボックス内（アプリケーションディレクトリ内のファイル）に含まれている必要があります。

「Console」タブによるオブジェクトの検査

Console クラスでは、log()、warn()、info()、error()、dump() の 5 つのメソッドを定義します。

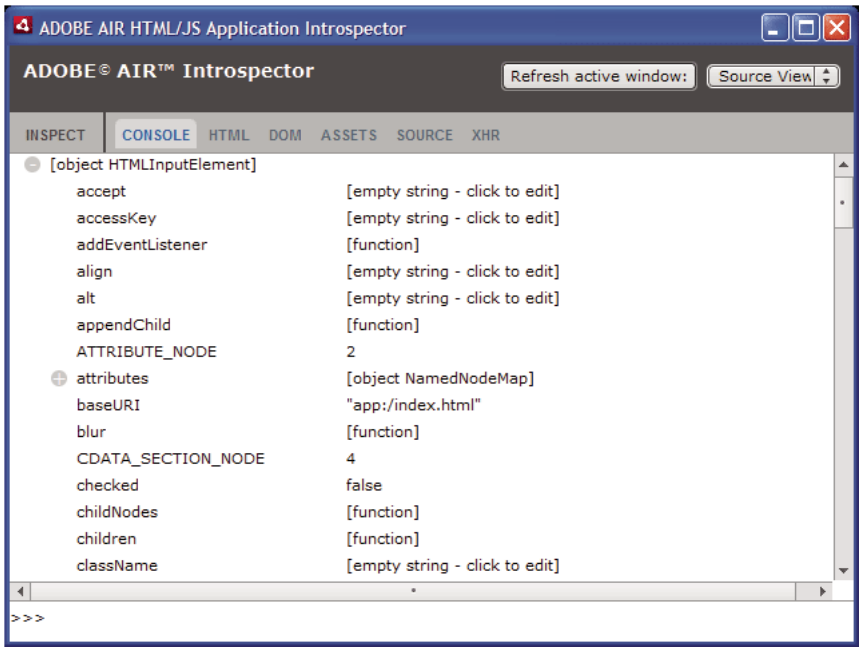
log()、warn()、info()、error() の各メソッドを使用すると、オブジェクトを「Console」タブに送信できます。これらのうち最も基本的なメソッドは log() です。次のコードでは、test 変数で表される単純なオブジェクトを、「Console」タブに送信します。

```
var test = "hello";
air.Introspector.Console.log(test);
```

しかし、複雑なオブジェクトを「Console」タブに送信したほうが便利です。例えば、次の HTML ページに含まれているボタン（btn1）は、そのボタンオブジェクト自体を「Console」タブに送信する関数を呼び出します。

```
<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>
```

ボタンをクリックすると、「Console」タブに btn1 オブジェクトが表示されます。オブジェクトのツリー表示を展開すると、そのプロパティを検査できます。



オブジェクトのプロパティを編集するには、一覧のプロパティ名の右側をクリックして、テキストを変更します。

info()、error()、warn() の各メソッドも、log() メソッドに似ています。ただし、これらのメソッドを呼び出した場合、「Console」の行の最初に次のようなアイコンが表示されます。

メソッド	アイコン
info()	
error()	
warn()	

log()、warn()、info()、error() の各メソッドでは、実際のオブジェクトの参照のみが送信されます。したがって、表示されるプロパティは、表示時点でのプロパティだけです。実際のオブジェクトを直列化するには、dump() メソッドを使用します。このメソッドには、次の 2 つのパラメーターがあります。

パラメーター	説明
dumpObject	直列化されるオブジェクトです。
levels	オブジェクトツリーで検査されるレベルの最大数（ルートレベルを除く）です。デフォルト値は 1 です（ツリーのルートレベルに 1 レベル加えて表示されます）。このパラメーターはオプションです。

dump() メソッドを呼び出すと、オブジェクトが直列化されてから「Console」タブに送信されます。したがって、オブジェクトのプロパティを編集することはできません。例えば、次のようなコードがあるとします。

```
var testObject = new Object();
testObject.foo = "foo";
testObject.bar = 234;
air.Introspector.Console.dump(testObject);
```

このコードを実行すると、testObject オブジェクトとそのプロパティが「Console」タブに表示されますが、このタブでプロパティ値を編集することはできません。

AIR Introspector の設定

コンソールを構成するには、AIRIntrospectorConfig グローバル変数のプロパティを設定します。例えば、次の JavaScript コードでは、各列を 100 文字で折り返すように AIR Introspector を構成します。

```
var AIRIntrospectorConfig = new Object();
AIRIntrospectorConfig.wrapColumns = 100;
```

AIRIntrospectorConfig 変数のプロパティ設定は、AIRIntrospector.js ファイルを（script タグを介して）読み込む前に行ってください。

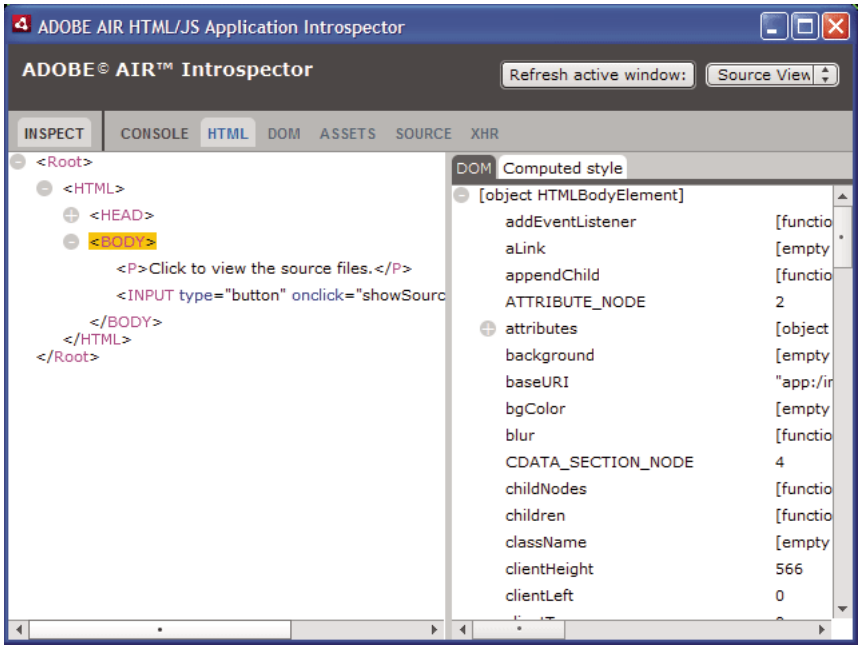
AIRIntrospectorConfig 変数のプロパティには、次の 8 種類があります。

プロパティ	デフォルト値	説明
closeIntrospectorOnExit	true	アプリケーションの他のすべてのウィンドウを閉じたときに Inspector ウィンドウも閉じるように設定します。
debuggerKey	123（F12 キー）	AIR Introspector ウィンドウの表示と非表示を切り替えるキーボードショートカットのキーコードです。
debugRuntimeObjects	true	JavaScript で定義されたオブジェクトに加えてランタイムオブジェクトも展開するように Introspector を設定します。
flashTabLabels	true	「Console」タブと「XMLHttpRequest」タブ内で変更（テキストの登録など）が発生したときにタブが点滅するように設定します。
introspectorKey	122（F11 キー）	Inspect パネルを開くキーボードショートカットのキーコードです。
showTimestamp	true	各行の先頭にタイムスタンプを表示するように「Console」タブを設定します。
showSender	true	メッセージの送信元オブジェクトに関する情報を、各行の先頭に表示するように「Console」タブを設定します。
wrapColumns	2000	ソースファイルを折り返す列数です。

AIR Introspector のインターフェイス

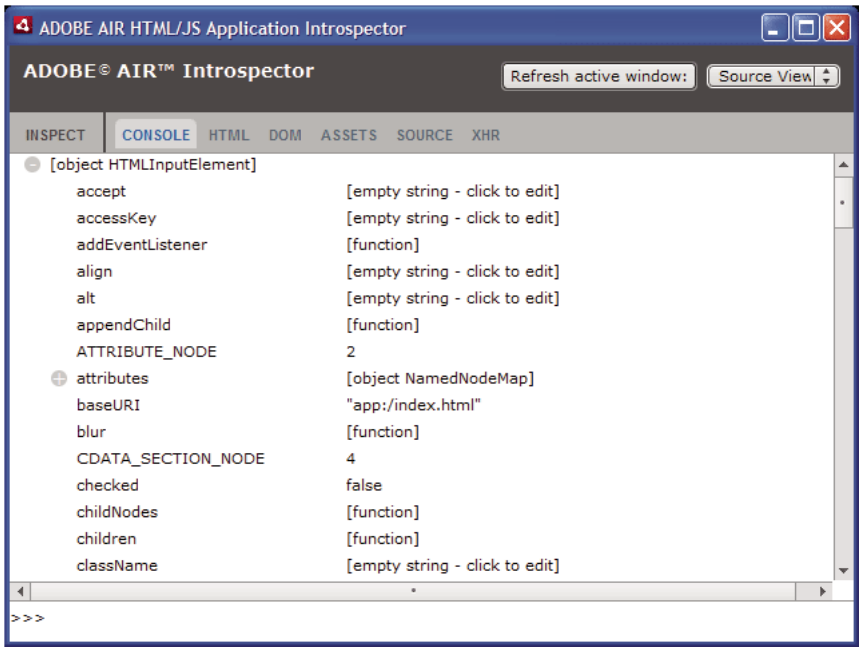
アプリケーションのデバッグ時に AIR Introspector ウィンドウを開くには、F12 キーを押すか、Console クラスのいずれかのメソッドを呼び出します（279 ページの「「Console」タブによるオブジェクトの検査」を参照）。ホットキーを設定して、F12 以外のキーに変更することもできます。281 ページの「AIR Introspector の設定」を参照してください。

次に示すように、AIR Introspector ウィンドウには、「Console」、「HTML」、「DOM」、「Assets」、「Source」、「XHR」の 6 つのタブがあります。



「Console」タブ

「Console」タブには、air.Introspector.Console クラスのいずれかのメソッドにパラメーターとして渡されるプロパティの値が表示されます。詳しくは、279 ページの「「Console」タブによるオブジェクトの検査」を参照してください。

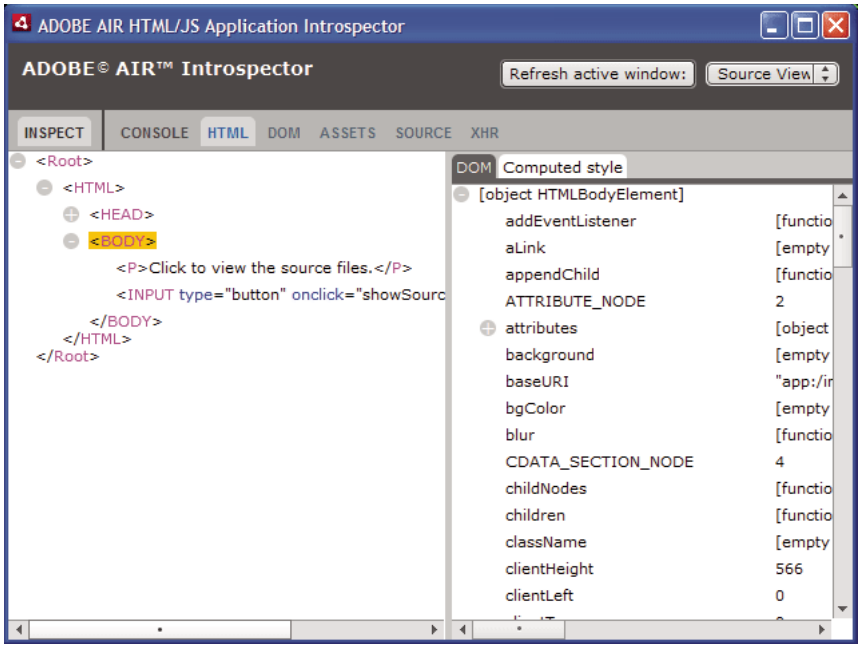


- コンソールをクリアするには、テキストを右クリックして「Clear Console」を選択します。

- 「Console」タブ内のテキストをファイルに保存するには、「Console」タブを右クリックして「Save Console To File」を選択します。
- 「Console」タブ内のテキストをクリップボードに保存するには、「Console」タブを右クリックして「Save Console To Clipboard」を選択します。選択したテキストのみをクリップボードにコピーするには、テキストを右クリックして「コピー」を選択します。
- Console クラス内のテキストをファイルに保存するには、「Console」タブを右クリックして「Save Console To File」を選択します。
- タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

「HTML」タブ

「HTML」タブには、HTML DOM 全体がツリー構造で表示されます。エレメントをクリックすると、そのプロパティがタブの右側に表示されます。ツリーのノードを展開するには「+」アイコンを、折りたたむには「-」アイコンをクリックします。



「HTML」タブでは任意の属性やテキストエレメントを編集でき、編集した値がアプリケーションに反映されます。

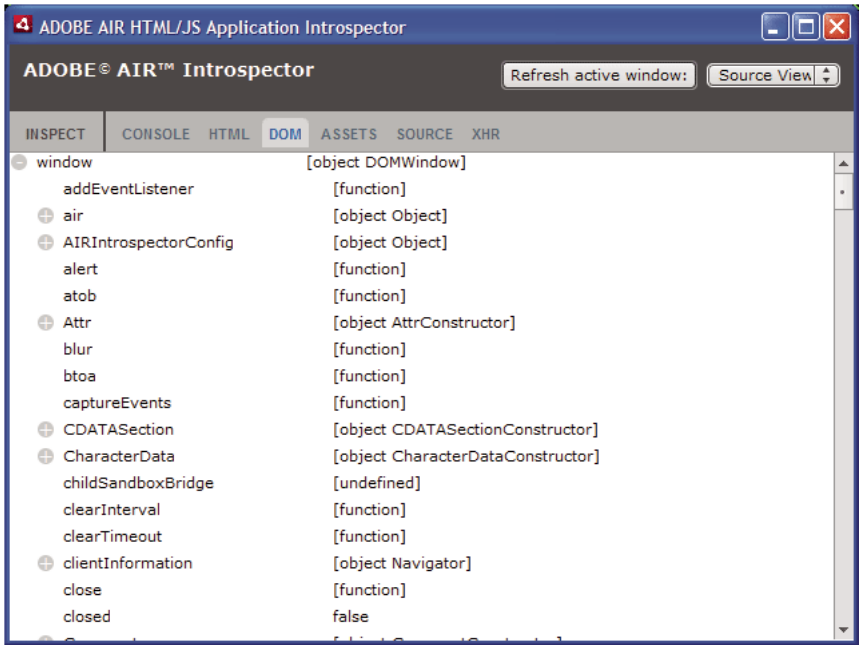
AIR Introspector ウィンドウのタブ一覧の左側にある「Inspect」ボタンをクリックします。メインウィンドウの HTML ページにある任意のエレメントをクリックすると、関連付けられている DOM オブジェクトが「HTML」タブに表示されます。メインウィンドウにフォーカスがあるときは、キーボードショートカットを押して「Inspect」ボタンのオンとオフを切り替えることもできます。デフォルトでは、このキーボードショートカットは F11 です。キーボードショートカットを設定して、F11 以外のキーに変更することもできます。281 ページの「AIR Introspector の設定」を参照してください。

「HTML」タブに表示されているデータを最新の情報に更新するには、AIR Introspector ウィンドウ上部にある「Refresh Active Window」ボタンをクリックします。

タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

「DOM」タブ

「DOM」タブには、window オブジェクトがツリー構造で表示されます。ストリングおよび数値の任意のプロパティを編集でき、編集した値がアプリケーションに反映されます。

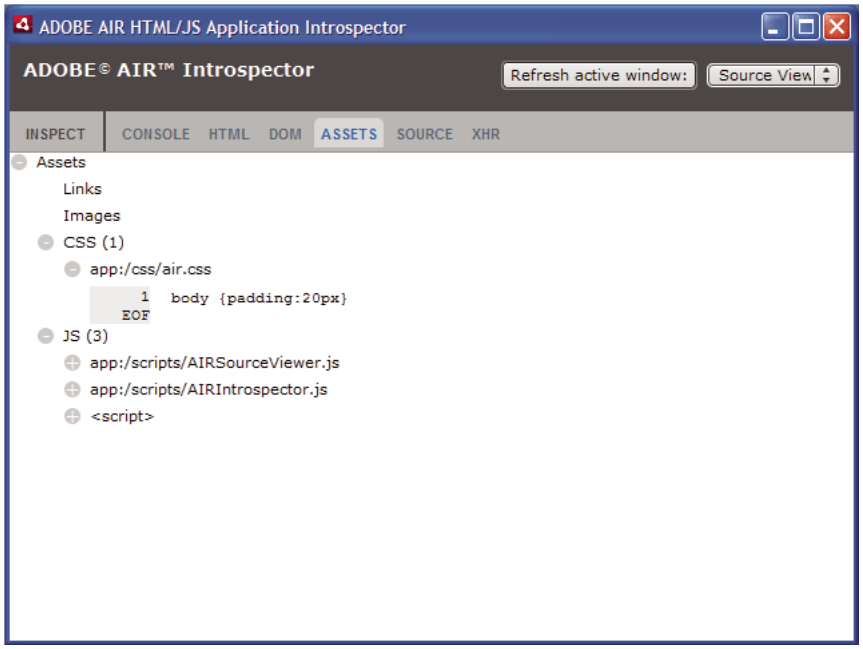


「DOM」タブに表示されているデータを最新の情報に更新するには、AIR Introspector ウィンドウ上部にある「Refresh Active Window」ボタンをクリックします。

タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

「Assets」タブ

「Assets」タブでは、ネイティブウィンドウに読み込まれたリンク、イメージ、CSS および JavaScript ファイルを確認できます。これらのノードのいずれかを展開すると、ファイルの内容または実際に使用されているイメージが表示されます。



「Assets」タブに表示されているデータを最新の情報に更新するには、AIR Introspector ウィンドウ上部にある「Refresh Active Window」ボタンをクリックします。

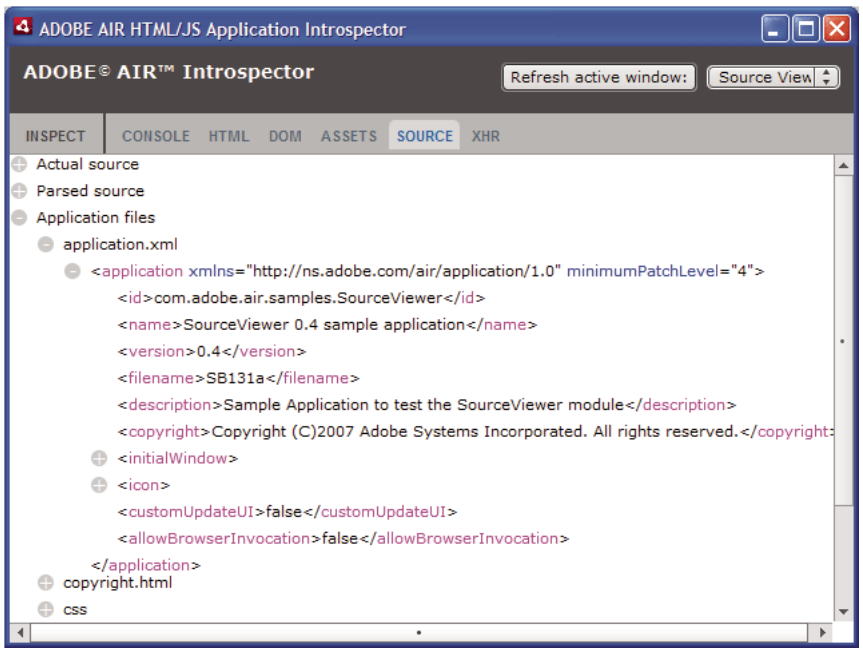
タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

「Source」タブ

「Source」タブには、次の 3 つのセクションがあります。

- Actual source - アプリケーション開始時にルートコンテンツとして読み込まれるページの HTML ソースが表示されます。
- Parsed source - アプリケーション UI を構成する現在のマークアップが表示されます。アプリケーション実行中に Ajax 技術を使用してマークアップコードが生成されるので、実際のソースとは異なる場合があります。

- Application files - アプリケーションディレクトリ内のファイルが一覧表示されます。この一覧は、アプリケーションセキュリティサンドボックス内のコンテンツから AIR Introspector が起動された場合にのみ表示されます。このセクションには、テキストファイルの内容やイメージを表示できます。

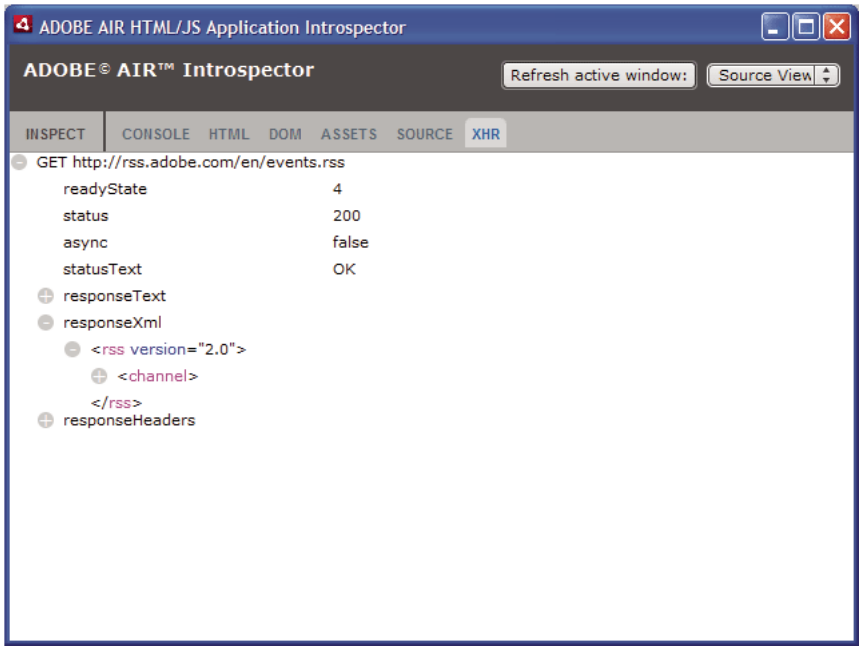


「Source」タブに表示されているデータを最新の情報に更新するには、AIR Introspector ウィンドウ上部にある「Refresh Active Window」ボタンをクリックします。

タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

「XHR」タブ

「XHR」タブには、アプリケーション内のすべての XMLHttpRequest 通信がインターセプトされ、その情報が記録されます。これを使用すると、responseText や responseXML などの XMLHttpRequest プロパティ（使用可能な場合）をツリー表示で確認できます。



タブ内に表示されているテキストを検索するには、Ctrl + F キー（Windows）または Command + F キー（Mac OS）を押します（非表示のツリーノードは検索されません）。

非アプリケーションサンドボックスのコンテンツに対する AIR Introspector の使用

アプリケーションディレクトリのコンテンツを、非アプリケーションサンドボックスにマップされるインラインフレームまたはフレームに読み込むことができます（「AIR のセキュリティ」（ActionScript 開発者用）または「Adobe AIR の HTML セキュリティ」（HTML 開発者用）を参照してください）。AIR Introspector はこのようなコンテンツにも使用できますが、次の規則に従ってください。

- アプリケーションサンドボックスと非アプリケーションサンドボックス（インラインフレーム）の両方のコンテンツに、AIRIntrospector.js ファイルを含める必要があります。
- parentSandboxBridge プロパティを上書きしないでください。AIR Introspector コードでこのプロパティを使用します。プロパティは必要に応じて追加してください。例えば、次のように記述しないでください。

  ```
  parentSandboxBridge = mytrace: function(str) {runtime.trace(str)} ;
  ```

  次のようなシンタックスを使用してください。

  ```
  parentSandboxBridge.mytrace = function(str) {runtime.trace(str)};
  ```

- 非アプリケーションサンドボックスのコンテンツからは、F12 キーを押して AIR Introspector を開くことができません。また、air.Introspector.Console クラスのいずれかのメソッドを呼び出して AIR Introspector を開くこともできません。「Open Introspector」ボタンをクリックすることによってのみ、Introspector ウィンドウを開くことができま

す。デフォルトでは、このボタンがインラインフレームまたはフレームの右上に追加されます（非アプリケーションサンドボックスのコンテンツに課せられるセキュリティ制限により、ボタンのクリックなど、ユーザー操作の結果としてのみ新しいウィンドウを開くことができます）。

- アプリケーションサンドボックスと非アプリケーションサンドボックスのそれぞれに対して、別個の AIR Introspector ウィンドウを開くことができます。AIR Introspector ウィンドウに表示されるタイトルで両者を区別できます。
- AIR Introspector が非アプリケーションサンドボックスから実行されている場合は、「Source」タブにアプリケーションファイルが表示されません。
- AIR Introspector で確認できるのは、起動元のサンドボックス内にあるコードだけです。

第 20 章：AIR アプリケーションのローカライズ

Adobe AIR 1.1 以降

Adobe® AIR® には、多言語のサポートが含まれています。

ActionScript 3.0 および Flex フレームワークにおけるコンテンツのローカライズ作業の概要については、『ActionScript 3.0 開発ガイド』の「アプリケーションのローカライズ」を参照してください。

AIR でサポートされている言語

AIR 1.1 リリースでは、AIR アプリケーションのローカライズ対象として次の言語がサポートされています。

- 簡体字中国語
- 繁体字中国語
- フランス語
- ドイツ語
- イタリア語
- 日本語
- 韓国語
- ポルトガル語（ブラジル）
- ロシア語
- スペイン語

AIR 1.5 リリースでは、次の言語が追加されています。

- チェコ語
- オランダ語
- ポーランド語
- スウェーデン語
- トルコ語

関連項目

マルチ言語対応の Adobe AIR 向け Flex アプリケーション開発

マルチ言語対応の HTML ベースのアプリケーション開発

AIR アプリケーションインストーラーでのアプリケーションの名前と説明のローカライズ

Adobe AIR 1.1 以降

アプリケーション記述ファイルの name エレメントと description エレメントには、複数の言語を指定できます。例えば、次のようにして 3 つの言語（英語、フランス語、ドイツ語）でアプリケーション名を指定します。

```
<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>
```

各テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

name エレメントは、AIR アプリケーションインストーラーが表示するアプリケーション名を定義します。AIR アプリケーションインストーラーは、オペレーティングシステムの設定で定義されているユーザーインターフェイス言語に合わせて適切なローカライズされた値を使用します。

同様に、アプリケーション記述ファイルの description エレメントの複数の言語バージョンを指定できます。このエレメントは、AIR アプリケーションインストーラーが表示する説明のテキストを定義します。

これらの設定意は、AIR アプリケーションインストーラーで使用可能な言語にのみ適用されます。インストール済みのアプリケーションの実行に使用できるロケールは定義しません。AIR アプリケーションでは、AIR アプリケーションインストーラーで使用可能な言語に加えて、複数の言語をサポートするユーザーインターフェイスを提供できます。

詳しくは、204 ページの「AIR アプリケーション記述エレメント」を参照してください。

関連項目

マルチ言語対応の Adobe AIR 向け Flex アプリケーション開発

マルチ言語対応の HTML ベースのアプリケーション開発

AIR HTML ローカリゼーションフレームワークによる HTML コンテンツのローカライズ

Adobe AIR 1.1 以降

AIR 1.1 SDK には、HTML ローカリゼーションフレームワークが含まれています。AIRLocalizer.js JavaScript ファイルによってこのフレームワークを定義します。AIRLocalizer.js は AIR SDK の frameworks ディレクトリに格納されています。このファイルには air.Localizer クラスが含まれています。air.Localizer クラスには、複数のローカライズされたバージョンをサポートするアプリケーションを作成するための機能が用意されています。

AIR の HTML ローカリゼーションフレームワークコードの読み込み

ローカリゼーションフレームワークを使用するには、AIRLocalizer.js ファイルをプロジェクトにコピーします。次に、script タグを使用して、アプリケーションのメイン HTML ファイルに含めます。

```
<script src="AIRLocalizer.js" type="text/javascript" charset="utf-8"></script>
```

それ以降の JavaScript では、air.Localizer.localizer オブジェクトを呼び出すことができます。

```
<script>
    var localizer = air.Localizer.localizer;
</script>
```

air.Localizer.localizer オブジェクトは、ローカライズされたリソースを使用および管理するためのメソッドとプロパティを定義するシングルトンオブジェクトです。Localizer クラスには次のメソッドが含まれています。

メソッド	説明
getFile()	指定されたロケールの指定されたリソースバンドルのテキストを取得します。296 ページの「特定のロケールのリソースの取得」を参照してください。
getLocaleChain()	ロケールチェーン内の言語を返します。296 ページの「ロケールチェーンの定義」を参照してください。
getResourceBundle()	バンドルキーおよび対応する値をオブジェクトとして返します。296 ページの「特定のロケールのリソースの取得」を参照してください。
getString()	リソース用に定義されているストリングを取得します。296 ページの「特定のロケールのリソースの取得」を参照してください。
setBundlesDirectory()	バンドルディレクトリの場所を設定します。295 ページの「AIR HTML ローカライザ設定のカスタマイズ」を参照してください。
setLocalAttributePrefix()	HTML DOM エレメントで使用されるローカライザ属性で使用する接頭辞を設定します。295 ページの「AIR HTML ローカライザ設定のカスタマイズ」を参照してください。
setLocaleChain()	ロケールチェーン内の言語の順序を設定します。296 ページの「ロケールチェーンの定義」を参照してください。
sortLanguagesByPreference()	オペレーティングシステム設定のロケールの順序に基づいてロケールチェーン内のロケールを並べ替えます。296 ページの「ロケールチェーンの定義」を参照してください。
update()	現在のロケールチェーン内のローカライズされたストリングを使用して HTML DOM（または DOM エレメント）を更新します。ロケールチェーンについて詳しくは、293 ページの「ロケールチェーンの管理」を参照してください。update() メソッドについて詳しくは、294 ページの「現在のロケールを使用するための DOM エレメントの更新」を参照してください。

Localizer クラスには次の静的プロパティが含まれています。

プロパティ	説明
localizer	アプリケーションのシングルトン Localizer オブジェクトへの参照を返します。
ultimateFallbackLocale	アプリケーションがユーザー設定をサポートしていない場合に使用されるロケールです。296 ページの「ロケールチェーンの定義」を参照してください。

サポート対象言語の指定

アプリケーションのサポート対象言語を指定するには、アプリケーション記述ファイルの <supportedLanguages> エレメントを使用します。このエレメントは、iOS、Mac キャプティブランタイムおよび Android アプリケーションでのみ使用され、その他の種類のアプリケーションでは無視されます。

<supportedLanguages> エレメントを指定しない場合、Packager によって、アプリケーションの種類に応じてデフォルトで次の処理が実行されます。

- iOS - アプリケーションのサポート対象言語として、AIR ランタイムのサポート対象言語のすべてが、iOS App Store で表示されます。
- Mac キャプティブランタイム - キャプティブバンドルとともにパッケージ化されたアプリケーションには、ローカライズ情報は含まれません。
- Android — アプリケーションバンドルに、AIR ランタイムのサポート対象言語のすべてに対応するリソースが含まれます。

詳しくは、234 ページの「supportedLanguages」を参照してください。

リソースバンドルの定義

HTML ローカリゼーションフレームワークは、ローカライズされたバージョンのストリングをローカリゼーションファイルから読み取ります。ローカリゼーションファイルは、テキストファイル内で直列化された、キーベースの値のコレクションです。ローカリゼーションファイルは、バンドルと呼ばれることもあります。

アプリケーションプロジェクトディレクトリに、locale という名前のサブディレクトリを作成します（別の名前を使用することもできます。295 ページの「AIR HTML ローカライザ設定のカスタマイズ」を参照してください）。このディレクトリには、ローカリゼーションファイルが格納されます。このディレクトリはバンドルディレクトリとも呼ばれます。

アプリケーションがサポートする各ロケールについて、バンドルディレクトリのサブディレクトリを作成します。各サブディレクトリには、ロケールコードに一致する名前を付けます。例えば、フランス語用のディレクトリには「fr」、英語用のディレクトリには「en」という名前を付けます。言語と国コードを含むロケールを定義するには、アンダースコア（_）文字を使用できます。例えば、米国英語用のディレクトリには「en_us」という名前を付けます（または、「en-us」のようにアンダースコアの代わりにハイフンを使用することもできます。HTML ローカリゼーションフレームワークではどちらもサポートしています）。

locale サブディレクトリには、必要な数だけリソースファイルを追加できます。一般的には、言語ごとにローカリゼーションファイルを作成し、その言語のディレクトリにファイルを配置します。HTML ローカリゼーションフレームワークには、ファイルの内容を読み取るための getFile() メソッドが含まれています（296 ページの「特定のロケールのリソースの取得」を参照してください）。

.properties というファイル拡張子を持つファイルは、ローカリゼーションプロパティファイルと呼ばれます。ローカリゼーションプロパティファイルを使用して、ロケールのキーと値のペアを定義できます。プロパティファイルでは、各行で 1 つのストリング値を定義します。例えば、次の例では、ストリング値 "Hello in English." を、greeting という名前のキーに対して定義しています。

```
greeting=Hello in English.
```

次のテキストを含むプロパティファイルは、6 つのキーと値のペアを定義します。

```
title=Sample Application
greeting=Hello in English.
exitMessage=Thank you for using the application.
color1=Red
color2=Green
color3=Blue
```

この例では、英語バージョンのプロパティファイルは、en ディレクトリに格納されています。

このプロパティファイルのフランス語バージョンは fr ディレクトリに配置されています。

```
title=Application Example
greeting=Bonjour en français.
exitMessage=Merci d'avoir utilisé cette application.
color1=Rouge
color2=Vert
color3=Bleu
```

異なる種類の情報用に複数のリソースファイルを定義できます。例えば、legal.properties ファイルに定型の法律に関するテキスト（著作権情報など）を含めることができます。このリソースは複数のアプリケーションで再利用できます。同様に、ユーザーインターフェイスの部分ごとにローカライズされたコンテンツを定義する、個別のファイルを定義することもできます。

複数の言語をサポートするには、これらのファイルで UTF-8 エンコードを使用します。

ロケールチェーンの管理

アプリケーションで AIRLocalizer.js ファイルを読み込むと、アプリケーションで定義されているロケールが調べられます。これらのロケールはバンドルディレクトリのサブディレクトリに対応します（292 ページの「リソースバンドルの定義」を参照してください）。この利用可能なロケールの一覧をロケールチェーンと呼びます。AIRLocalizer.js ファイルは、オペレーティングシステムの設定で定義されている優先順位に基づいて、ロケールチェーンを自動的に並べ替えます（Capabilities.languages プロパティは、オペレーティングシステムのユーザーインターフェイスの言語を、優先される順序で一覧表示します）。

したがって、アプリケーションで「en」、「en_US」および「en_UK」ロケールのリソースを定義している場合、AIR HTML ローカリゼーションフレームワークではロケールチェーンが適切に並べ替えられます。「en」がプライマリロケールであるシステムでアプリケーションを起動した場合、ロケールチェーンは ["en", "en_US", "en_UK"] のように並べ替えられます。この場合、アプリケーションは最初に「en」バンドルでリソースを検索し、次に「en_US」バンドル内を検索します。

システムのプライマリロケールが「en-US」である場合は、並べ替えは ["en_US", "en", "en_UK"] のようになります。この場合、アプリケーションは最初に「en_US」バンドルでリソースを検索し、次に「en」バンドル内を検索します。

デフォルトでは、アプリケーションはロケールチェーン内の最初のロケールをデフォルトのロケールとして定義します。アプリケーションを最初に実行するときに、ユーザーにロケールを選択するよう求めることができます。選択された内容を環境設定ファイルに保存し、それ以降にアプリケーションを起動するときにはそのロケールを使用できます。

アプリケーションでは、ロケールチェーン内の任意のロケールのリソースストリングを使用できます。特定のロケールでリソースストリングが定義されていない場合、アプリケーションはロケールチェーンで定義されている他のロケールで次に一致するリソースストリングを使用します。

ロケールチェーンは、Localizer オブジェクトの setLocaleChain() メソッドを呼び出すことによってカスタマイズできます。296 ページの「ロケールチェーンの定義」を参照してください。

ローカライズされたコンテンツによる DOM エレメントの更新

アプリケーション内のエレメントで、ローカリゼーションプロパティファイル内のキー値を参照できます。例えば、次の例の title エレメントでは、local_innerHTML 属性を指定しています。ローカリゼーションフレームワークでは、この属性を使用してローカライズされた値を検索します。デフォルトでは、フレームワークは「local_」で始まる属性名を検索します。フレームワークは、「local_」に続くテキストと一致する名前を持つ属性を更新します。この場合、フレームワークは title エレメントの innerHTML 属性を設定します。innerHTML 属性は、デフォルトプロパティファイル（default.properties）の mainWindowTitle キーに定義されている値を使用します。

```
<title local_innerHTML="default.mainWindowTitle"/>
```

現在のロケールで一致する値が定義されていない場合、ローカライザフレームワークは残りのロケールチェーンを検索します。ロケールチェーン内で、次に見つかった、値が定義されているロケールが使用されます。

次の例では、p エレメントのテキスト（innerHTML 属性）で、デフォルトプロパティファイルで定義されている greeting キーの値を使用します。

```
<p local_innerHTML="default.greeting" />
```

次の例では、input エレメントの value 属性（および表示テキスト）で、デフォルトプロパティファイルで定義されている btnBlue キーの値を使用します。

```
<input type="button" local_value="default.btnBlue" />
```

現在のロケールチェーンで定義されているストリングを使用するように HTML DOM を更新するには、Localizer オブジェクトの update() メソッドを呼び出します。update() メソッドを呼び出すと、Localizer オブジェクトは DOM を解析し、ローカリゼーション（「local_...」）属性が見つかった場所で操作を適用します。

```
air.Localizer.localizer.update();
```

属性（「innerHTML」など）と、対応するローカリゼーション属性（「local_innerHTML」など）の両方の値を定義できます。この場合、ローカリゼーションフレームワークは、ロケールチェーンで一致する値が見つかった属性値のみを上書きします。例えば、次のエレメントでは、value 属性と local_value 属性の両方を定義しています。

```
<input type="text" value="Blue" local_value="default.btnBlue"/>
```

特定の DOM エレメントのみを更新することもできます。次の節の 294 ページの「現在のロケールを使用するための DOM エレメントの更新」を参照してください。

デフォルトでは、AIR HTML ローカライザは、エレメントのローカリゼーション設定を定義する属性の接頭辞として、「local_」を使用します。例えば、デフォルトの場合、local_innerHTML 属性は、エレメントの innerHTML 値のバンドルおよびリソース名を定義します。また、デフォルトでは、local_value 属性は、エレメントの value 属性で使用されるバンドルおよびリソース名を定義します。ローカライザで「local_」以外の属性接頭辞を使用するように構成できます。295 ページの「AIR HTML ローカライザ設定のカスタマイズ」を参照してください。

現在のロケールを使用するための DOM エレメントの更新

Localizer オブジェクトが HTML DOM を更新する場合、マークされたエレメントは、現在のロケールチェーンで定義されているストリングに基づいて属性値を使用します。HTML ローカライザで HTML DOM を更新するには、Localizer オブジェクトの update() メソッドを呼び出します。

```
air.Localizer.localizer.update();
```

指定した DOM エレメントのみを更新するには、DOM エレメントをパラメーターとして update() メソッドに渡します。update() メソッドのパラメーターは parentNode のみで、これはオプションです。指定した場合、parentNode パラメーターはローカライズする DOM エレメントを定義します。update() メソッドを呼び出して、parentNode パラメーターを指定すると、ローカリゼーション属性を指定するすべての子エレメントについてローカライズされた値が設定されます。

例えば、次のような div エレメントがあるとします。

```
<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>
```

現在のロケールチェーンで定義されているローカライズされたストリングを使用するようにこのエレメントを更新するには、次の JavaScript コードを使用します。

```
var divElement = window.document.getElementById("colorsDiv");
air.Localizer.localizer.update(divElement);
```

ロケールチェーンでキーの値が見つからない場合は、ローカリゼーションフレームワークによって、属性値は「local_」属性の値に設定されます。例えば、前の例で、lblColors キーの値が（ロケールチェーン内のデフォルトプロパティファイルのいずれにも）見つからなかったとします。この場合、ローカリゼーションフレームワークは、「default.lblColors」を innerHTML 値として使用します。この値を使用することで、（開発者に対して）リソースが不足していることが示されます。

update() メソッドは、ロケールチェーン内でリソースが見つからなかった場合、resourceNotFound イベントを送出します。air.Localizer.RESOURCE_NOT_FOUND 定数は、ストリング「resourceNotFound」を定義します。このイベントには、bundleName、resourceName および locale の 3 つのプロパティがあります。bundleName プロパティは、リソースが見つからないバンドルの名前です。resourceName プロパティは、リソースが見つからないバンドルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。

update() メソッドは、指定されたバンドルが見つからなかった場合、bundleNotFound イベントを送出します。air.Localizer.BUNDLE_NOT_FOUND 定数は、ストリング「bundleNotFound」を定義します。このイベントには、bundleName と locale の 2 つのプロパティがあります。bundleName プロパティは、リソースが見つからないバンドルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。

update() メソッドは非同期に動作します（resourceNotFound イベントと bundleNotFound イベントを非同期に送出します）。次のコードは、resourceNotFound イベントと bundleNotFound イベントのイベントリスナーを設定します。

```
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);
}
```

AIR HTML ローカライザ設定のカスタマイズ

Localizer オブジェクトの setBundlesDirectory() メソッドによって、バンドルディレクトリのパスをカスタマイズできます。Localizer オブジェクトの setLocalAttributePrefix() メソッドによって、バンドルディレクトリのパスをカスタマイズし、ローカライザで使用される属性値をカスタマイズできます。

デフォルトのバンドルディレクトリは、アプリケーションディレクトリの locale サブディレクトリとして定義されています。別のディレクトリを指定するには、Localizer オブジェクトの setBundlesDirectory() メソッドを呼び出します。このメソッドは、path というパラメーターを 1 つ使用します。このパラメーターは、目的のバンドルディレクトリへのパスを示すストリングです。path パラメーターの値には、次のいずれかを指定できます。

- アプリケーションディレクトリを基準とした相対パスを定義するストリング（「locales」など）
- app、app-storage または file URL スキームを使用する有効な URL を定義するストリング（「app://languages」（http URL スキームは使用しないでください）
- File オブジェクト

URL およびディレクトリのパスについて詳しくは、次を参照してください。

- File オブジェクトのパス（ActionScript 開発者用）
- File オブジェクトのパス（HTML 開発者用）

例えば、次のコードは、バンドルディレクトリを、アプリケーション記憶域ディレクトリ（アプリケーションディレクトリではない）の languages サブディレクトリに設定します。

```
air.Localizer.localizer.setBundlesDirectory("languages");
```

有効なパスを path パラメーターとして渡します。そうしないと、このメソッドは BundlePathNotFoundError 例外をスローします。このエラーは、「BundlePathNotFoundError」をその name プロパティとして持ち、message プロパティで無効なパスを示します。

デフォルトでは、AIR HTML ローカライザは、エレメントのローカリゼーション設定を定義する属性の接頭辞として、「local_」を使用します。例えば、local_innerHTML 属性は、次の input エレメントの innerHTML 値として使用されるバンドルおよびリソース名を定義します。

```
<p local_innerHTML="default.greeting" />
```

Localizer オブジェクトの setLocalAttributePrefix() メソッドによって、「local_」以外の属性接頭辞を使用できます。この静的メソッドはパラメーターを 1 つ使用します。このパラメーターが属性接頭辞として使用されるストリングです。例えば、次のコードは、ローカリゼーションフレームワークで属性接頭辞として「loc_」を使用するように設定します。

```
air.Localizer.localizer.setLocalAttributePrefix("loc_");
```

ローカリゼーションフレームワークで使用する属性接頭辞をカスタマイズできます。デフォルト値（「local_」）が、コードで使用されている別の属性の名前と競合する場合は、接頭辞をカスタマイズできます。このメソッドを呼び出す場合は、HTML 属性として有効な文字を使用してください（例えば、値に空白文字を含めることはできません）。

HTML エレメントでのローカリゼーション属性の使用について詳しくは、293 ページの「ローカライズされたコンテンツによる DOM エレメントの更新」を参照してください。

バンドルディレクトリおよび属性接頭辞の設定は、アプリケーションセッション間で保持されません。カスタムバンドルディレクトリまたはカスタム属性接頭辞の設定を使用する場合は、アプリケーションを開始するたびに設定してください。

ロケールチェーンの定義

デフォルトでは、AIRLocalizer.js コードを読み込むと、デフォルトのロケールチェーンが設定されます。バンドルディレクトリで使用できるロケールおよびオペレーティングシステムの言語の設定によって、このロケールチェーンが定義されます（詳しくは、293 ページの「ロケールチェーンの管理」を参照してください）。

ロケールチェーンは、Localizer オブジェクトの静的メソッド setLocaleChain() を呼び出すことによって変更できます。例えば、ユーザーが特定の言語の環境設定を指定した場合に、このメソッドを呼び出すことができます。setLocaleChain() メソッドは chain パラメーターを使用します。このパラメーターは、["fr_FR","fr","fr_CA"] などのロケールの配列です。配列内のロケールの順序によって、フレームワークが（以降の操作で）リソースを検索する順序が決まります。チェーン内の最初のロケールでリソースが見つからない場合は、他のロケールのリソースを検索し続けます。chain 引数が指定されていない場合、配列ではない場合または空の配列である場合、この関数は失敗し、IllegalArgumentsError 例外をスローします。

Localizer オブジェクトの静的メソッド getLocaleChain() は、現在のロケールチェーンに含まれるロケールの一覧を示す配列を返します。

次のコードは、現在のロケールチェーンを読み取って、2 つのフランス語ロケールをチェーンの先頭に追加します。

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
```

setLocaleChain() メソッドは、ロケールチェーンを更新すると、「change」イベントを送出します。air.Localizer.LOCALE_CHANGE 定数は、ストリング「change」を定義します。このイベントには localeChain というプロパティがあります。このプロパティは、新しいロケールチェーン内のロケールコードの配列です。次のコードは、このイベントのイベントリスナーを設定します。

```
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);
}
```

静的プロパティ air.Localizer.ultimateFallbackLocale は、アプリケーションがユーザー設定をサポートしていない場合に使用されるロケールを表します。デフォルト値は「en」です。次のコードに示されているように、この値を別のロケールに設定できます。

```
air.Localizer.ultimateFallbackLocale = "fr";
```

特定のロケールのリソースの取得

Localizer オブジェクトの getString() メソッドは、特定のロケールでリソースとして定義されているストリングを返します。このメソッドを呼び出すときに、locale 値を指定する必要はありません。この場合、メソッドはロケールチェーン全体を検索し、特定のリソース名を提供する最初のロケールのストリングを返します。このメソッドのパラメーターは次のとおりです。

パラメーター	説明
bundleName	リソースを含むバンドル。これは、.properties 拡張子を除いたプロパティファイルのファイル名です。（例えば、このパラメーターが「alerts」に設定されている場合、Localizer コードは alerts.properties という名前のローカリゼーションファイル内を検索します。
resourceName	リソース名。
templateArgs	オプション。代替ストリング内の番号付きのタグを置き換えるストリングの配列。例えば、templateArgs パラメーターが ["Raúl", "4"] であり、一致するリソースストリングが "Hello, {0}. You have {1} new messages." である関数呼び出しを考えてみます。. この場合、この関数は "Hello, Raúl. You have 4 new messages." を返します。この設定を無視するには、null 値を渡します。
locale	オプション。使用するロケールコード（「en」、「en_us」または「fr」など）。ロケールが指定されており、一致する値が見つからなかった場合、このメソッドはロケールチェーン内の他のロケールの値を検索しません。ロケールコードが指定されていない場合、この関数はロケールチェーン内で、指定されたリソース名に対応する値を持つ最初のロケールにあるストリングを返します。

ローカリゼーションフレームワークでは、マークされた HTML DOM 属性を更新できます。ただし、他の方法でもローカライズされたストリングを使用できます。例えば、動的に生成される HTML でストリングを使用したり、関数呼び出しのパラメーター値としてストリングを使用したりすることができます。例えば、次のコードは alert() 関数を呼び出します。このとき、fr_FR ロケールのデフォルトプロパティファイルの error114 リソースで定義されているストリングを使用します。

```
alert(air.Localizer.localizer.getString("default", "error114", null, "fr_FR"));
```

getString() メソッドは、指定されたバンドル内でリソースが見つからなかった場合、resourceNotFound イベントを送出します。air.Localizer.RESOURCE_NOT_FOUND 定数は、ストリング「resourceNotFound」を定義します。このイベントには、bundleName、resourceName および locale の 3 つのプロパティがあります。bundleName プロパティは、リソースが見つからないバンドルの名前です。resourceName プロパティは、リソースが見つからないバンドルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。

getString() メソッドは、指定されたバンドルが見つからなかった場合、bundleNotFound イベントを送出します。air.Localizer.BUNDLE_NOT_FOUND 定数は、ストリング「bundleNotFound」を定義します。このイベントには、bundleName と locale の 2 つのプロパティがあります。bundleName プロパティは、リソースが見つからないバンドルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。

getString() メソッドは非同期に動作します（resourceNotFound イベントと bundleNotFound イベントを非同期に送出します）。次のコードは、resourceNotFound イベントと bundleNotFound イベントのイベントリスナーを設定します。

```
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);
}
```

Localizer オブジェクトの getResourceBundle() メソッドは、特定のロケールの指定されたバンドルを返します。メソッドの戻り値は、バンドルのキーに一致するプロパティを持つオブジェクトです（アプリケーションで指定されたバンドルが見つからなかった場合、メソッドは null を返します）。

メソッドは、locale と bundleName の 2 つのパラメーターを使用します。

パラメーター	説明
locale	ロケール（"fr" など）
bundleName	バンドル名

例えば、次のコードは、document.write() メソッドを呼び出して、fr ロケールのデフォルトのバンドルを読み込みます。次に、document.write() メソッドを呼び出して、そのバンドル内の str1 キーと str2 キーの値を書き込みます。

```
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/>");
```

getResourceBundle() メソッドは、指定されたバンドルが見つからなかった場合、bundleNotFound イベントを送出します。air.Localizer.BUNDLE_NOT_FOUND 定数は、ストリング「bundleNotFound」を定義します。このイベントには、bundleName と locale の 2 つのプロパティがあります。bundleName プロパティは、リソースが見つからないバンドルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。

Localizer オブジェクトの getFile() メソッドは、特定のロケールのバンドルの内容をストリングとして返します。バンドルファイルは UTF-8 ファイルとして読み取られます。このメソッドのパラメーターは次のとおりです。

パラメーター	説明
resourceFileName	リソースファイルのファイル名（「about.html」など）。

templateArgs	オプション。代替ストリング内の番号付きのタグを置き換えるストリングの配列。例えば、templateArgs パラメーターが ["Raúl", "4"] であり、一致するリソースファイルに 2 つの行が含まれている関数呼び出しを考えてみます。
	
	`<html>`
	`<body>Hello, {0}. You have {1} new messages.</body>`
	`</html>`
	
	この場合、関数は 2 行のストリングを返します。
	
	`<html>`
	`<body>Hello, Raúl. You have 4 new messages. </body>`
	`</html>`

locale	使用するロケールコード（「en_GB」など）。ロケールが指定されており、一致するファイルが見つからなかった場合、このメソッドはロケールチェーン内の他のロケールで検索を続行しません。ロケールコードが指定されていない場合は、ロケールチェーン内で、resourceFileName に一致するファイルが存在する最初のロケールからテキストを返します。

例えば、次のコードは、fr ロケールの about.html ファイルの内容を使用して、document.write() メソッドを呼び出します。

```
var aboutWin = window.open();
var aboutHtml = localizer.getFile("about.html", null, "fr");
aboutWin.document.close();
aboutWin.document.write(aboutHtml);
```

getFile() メソッドは、ロケールチェーン内でリソースが見つからなかった場合、fileNotFound イベントを送出します。air.Localizer.FILE_NOT_FOUND 定数は、ストリング「resourceNotFound」を定義します。getFile() メソッドは非同期に動作します（fileNotFound イベントを非同期に送出します）。このイベントには、fileName と locale の 2 つのプロパティがあります。fileName プロパティは、見つからなかったファイルの名前です。locale プロパティは、リソースが見つからないロケールの名前です。次のコードは、このイベントのイベントリスナーを設定します。

```
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);
}
```

関連項目

マルチ言語対応の HTML ベースのアプリケーション開発

第 21 章：PATH 環境変数

AIR SDK には、コマンドラインやターミナルから起動できるプログラムがいくつか用意されています。これらのプログラムは、通常、SDK bin ディレクトリへのパスを PATH 環境変数に含めるとより便利になります。

この章で説明する、Windows、Mac OS および Linux へのパスの設定方法に関する情報は、便利なガイドとして利用できます。ただし、コンピューターの構成は様々に異なるので、この手順がすべてのシステムに対して有効なわけではありません。このような場合、オペレーティングシステムのドキュメントから、またはインターネット上で、必要な情報を見つけることができます。

Bash シェルを使用した、Linux および Mac OS への PATH 設定

ターミナルウィンドウでコマンドを入力したとき、シェル（ユーザーが入力した文字を読み取り、それに応答するプログラム）はまず、ファイルシステム上のコマンドプログラムの場所を特定する必要があります。シェルは、$PATH という名前の環境変数に保存されているディレクトリのリスト上でコマンドを探します。PATH に現在含まれているコマンドを調べるには、次のとおり入力します。

```
echo $PATH
```

これにより、次のような、コロンで区切られたディレクトリのリストが返されます。

```
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin
```

AIR SDK bin ディレクトリへのパスをリストに追加する目的は、シェルが ADT および ADL ツールを見つけられるようにすることです。AIR SDK が /Users/fred/SDKs/AIR にあるとします。この場合、次のコマンドによって、必要なディレクトリがパスに追加されます。

```
export PATH=$PATH:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

注意：パスに空白文字が含まれている場合、次のようにバックスラッシュでエスケープします。

```
/Users/fred\ jones/SDKs/AIR\ 2.5\ SDK/bin
```

echo コマンドを使用すると、この記述が正しく機能するかどうかを改めて確認できます。

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/too
ls
```

正しく機能しています。これで、次のコマンドを入力するとバージョンの応答が正しく返されるようになりました。

```
adt -version
```

$PATH 変数を正しく変更すると、このコマンドは ADT のバージョンをレポートします。

しかしまだ問題が 1 つ残っています。次に新しいターミナルウィンドウを開始したときには、パス内の新しいエントリはなくなっています。パスを設定するコマンドを、新しいターミナルを開始するたびに実行する必要があります。

この問題に対する一般的な解決方法は、シェルが使用するいずれかの起動スクリプトにコマンドを追加することです。Mac OS では、~/username ディレクトリに .bash_profile というファイルを作成し、新しいターミナルウィンドウを開始するたびに実行できます。Ubuntu の場合、新しいターミナルウィンドウの開始時に実行する起動スクリプトは .bashrc です。その他の Linux ディストリビューションおよびシェルプログラムにも同様の規則があります。

コマンドをシェルの起動スクリプトに追加するには：

1 ホームディレクトリに変更します。

```
cd
```

2 シェルの設定プロファイルを作成し（必要な場合）、ファイルの最後に入力するテキストを「cat >>」でリダイレクトします。使用しているオペレーティングシステムおよびシェルに対する適切なファイルを使用します。例えば、Mac OS では .bash_profile、Ubuntu では .bashrc を使用できます。

```
cat >> .bash_profile
```

3 ファイルを追加するテキストを入力します。

```
export PATH=$PATH:/Users/cward/SDKs/android/tools:/Users/cward/SDKs/AIR/bin
```

4 キーボードの Ctrl+Shift+D キーを押して、テキストのリダイレクトを終了します。

5 ファイルを表示して、問題がないことを確認します。

```
cat .bash_profile
```

6 新しいターミナルウィンドウを開いて、パスをチェックします。

```
echo $PATH
```

パスの追加がリストされます。

後で、いずれかの SDK の新しいバージョンを別のディレクトリに作成する場合は、設定ファイルの path コマンドを必ず更新します。更新しないと、シェルは古いバージョンの使用を続行します。

Windows への PATH 設定

Windows で コマンドウィンドウを開くと、そのウィンドウは、システムのプロパティで定義されているグローバル環境変数を継承します。この内の重要な変数の 1 つが path です。path はディレクトリのリストであり、実行するプログラムの名前をユーザーが入力したとき、コマンドプログラムはこのリストを検索します。コマンドウィンドウの使用中に、path に現在含まれているコマンドを調べるには、次のとおり入力します。

```
set path
```

これにより、次のような、セミコロンで区切られたディレクトリのリストが返されます。

```
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

AIR SDK bin ディレクトリへのパスをリストに追加する目的は、コマンドプログラムが ADT および ADL ツールを見つけられるようにすることです。AIR SDK が C:¥SDKs¥AIR にあるとします。この場合、次の手順を実行して、適切なパスエントリを追加できます。

1 コントロールパネルから、「システムのプロパティ」ダイアログを開くか、「マイコンピューター」アイコンを右クリックして、メニューから「プロパティ」を選択します。

2 「詳細設定」タブで、「環境変数」ボタンをクリックします。

3 「環境変数」ダイアログの「システム環境変数」セクションで、Path エントリを選択します。

4 「編集」をクリックします。

5 「変数値」フィールドのテキストの末尾にスクロールします。

6 現在の値の最後に、次の値を入力します。

```
;C:\SDKs\AIR\bin
```

7 すべてのダイアログで「OK」をクリックして、パスを保存します。

コマンドウィンドウが 1 つでも開いていると、環境変数は更新されません。新しいコマンドウィンドウを開いて次のコマンドを入力し、パスが正しく設定されていることを確認します。

```
adt -version
```

後で、AIR SDK の場所を変更する場合、または新しいバージョンを追加する場合、path 変数を必ず更新してください。