July 14, 2010

Developing Adobe AIR Applications for Android

Contents

1. CHAPTER ONE Getting Started

This chapter includes information on setting up your development environment. It also includes a Hello World example.

Workflow for developing and testing an AIR for Android app

1. Write the ActionScript code.
2. Create a standard AIR application descriptor file (using the 2.5 namespace).
3. Compile the application.
4. Package the application as an Android package (.apk) with ADT.
5. Install the AIR runtime on the device (if not already installed) with the Android ADB tool.
6. Install the application on device (or Android emulator) with the Android ADB tool.
7. Launch the application on the device.

Available Documentation

In addition to this document, the following documentation is available:

- *Release notes* (on the Adobe prerelease website)
- Building Adobe AIR applications
- ActionScript 3.0 Reference for the Adobe Flash Platform

See the release notes for a comprehensive list of AIR for Android issues.

Flash Platform tool support

You can use your favorite Flash development tool to create an AIR for Android application. However, only Flash Professional CS5 and the AIR 2.5 SDK command-line tools provide direct support for packaging and on-device debugging. There is no Android support in Flash Builder yet.

The AIR for Android prerelease site includes the AIR for Android prerelease extension for Flash Professional CS5. Use this extension to update Flash Professional CS5 for use in building AIR for Android applications.

To install the AIR for Android prerelease extension for Flash Professional CS5:

1. Download the AIR for Android prerelease extension from the AIR for Android web site. This file is named *AIRforAndroid_FlashCS5_mmddyy.zxp*.
2. If Flash Professional CS5 is running, close it.
3. On Windows 7 or Windows Vista, run the Adobe Extension Manager as the Administrator. On the Windows Start menu, right-click Programs > Adobe Extension Manager CS5, and select Run as Administrator.
4. Double-click the .zxp file you downloaded from the prerelease site.

Instructions for updating the AIR SDKs used in Flash Builder and Flash CS4 Professional are provided in the release notes. To develop an AIR for Android application using the command-line tools, you will need a recent Flex SDK and the AIR 2.5 SDK available on the AIR for Android prerelease website.

Use the Android SDK tools to install the application on a device or Android emulator.

AIR for Android does not support HTML-based applications.

The desktop version of the Flex framework is not recommended for use on mobile devices. Adobe is working on a mobile update for the Flex framework, code named Slider. For more information, see http://labs.adobe.com/technologies/flex/mobile/.

Installing the Android SDK

The Android SDK provides many useful tools for developing Android applications. Obtain the Android SDK from:

http://developer.android.com/sdk/index.html

Follow the instructions for “Adding SDK Components” at http://developer.android.com/sdk/adding-components.html to add the Android tools and USB drivers (if needed).

You can use the Android SDK tools to install the AIR runtime and AIR applications on an Android device.

In addition, the Android SDK includes the Android Emulator, which allows simulation of many types of Android devices on your development computer.

The Android SDK also includes the Windows USB driver that is required to connect an Android device to a Windows computer.

The Android SDK requires the Java version 1.6 or later. You can obtain the latest version of Java from http://www.java.com/en/download. Mac OS includes a pre-installed version of Java.

Creating a Hello World application

This section shows you how to build a basic Hello World application. There are separate instructions for using Flash Professional CS5, Flash Builder, and the AIR SDK command-line tools.

Before you build the application, be sure that you have configured your development environment for AIR for Android development. See the previous sections for details.

Hello World—Using Flash Professional CS5

Before starting this tutorial, be sure to install the AIR for Android prerelease extension for Flash Professional CS5, available at the prerelease site. For more information, see "Flash Platform Tool Support" on page 4.

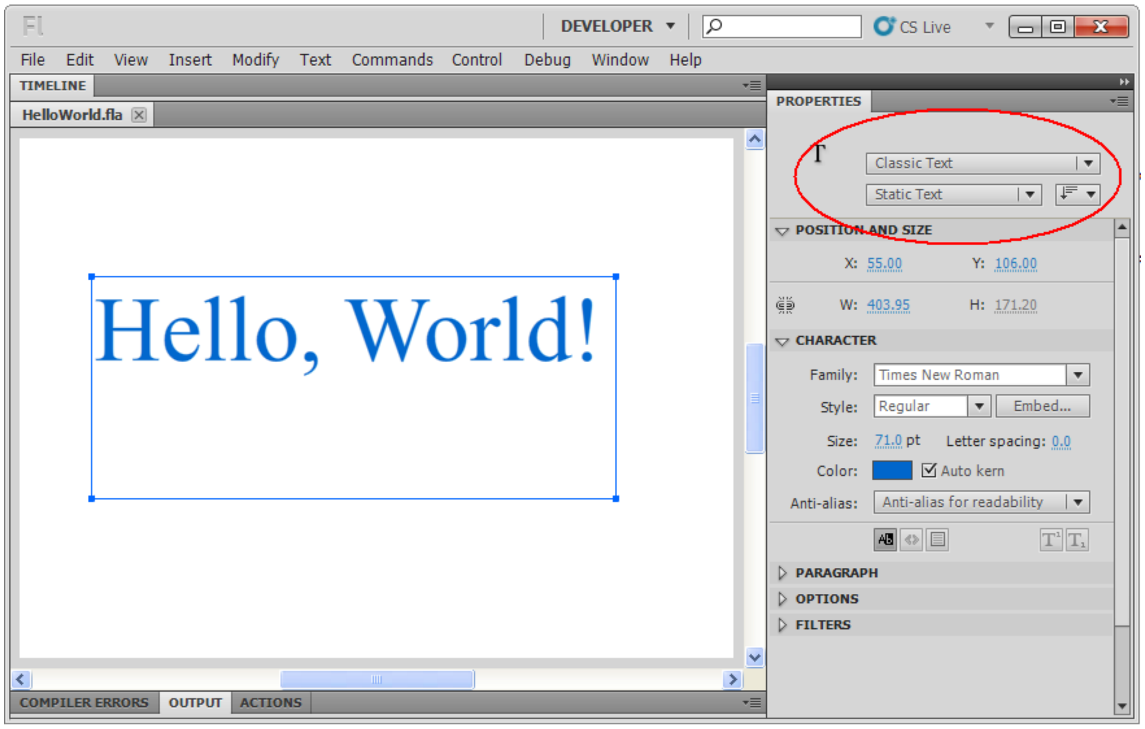
Create a project

1. Open Flash Professional CS5
2. Create a new AIR for Android project.

   The Flash Professional home screen includes a link to create an AIR for Android application. You can also select File > New > AIR, and then select the AIR for Android template.
3. Save the document as HelloWorld.fla

Write the code

Since this tutorial isn't really about writing code, just use the Text tool to write, "Hello, World!" on the stage:



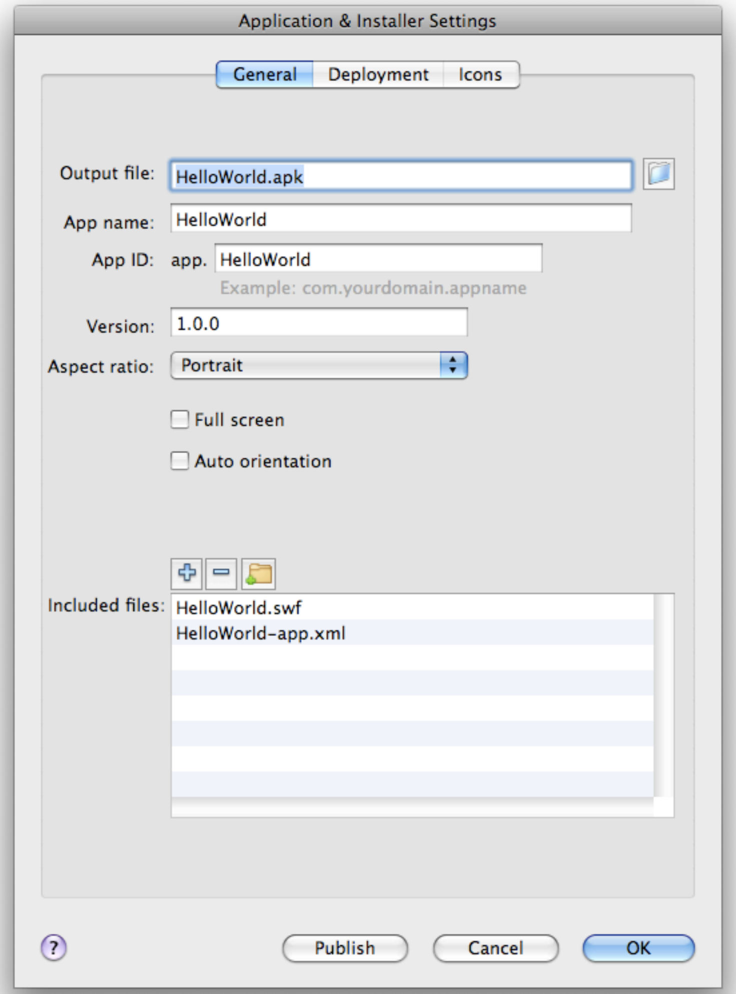
Classic Text
Static Text
X: 55.00 Y: 106.00
W: 403.95 H: 171.20
Family: Times New Roman
Style: Regular
Size: 71.0 pt Letter spacing: 0.0
Color: ☑ Auto kern
Anti-alias: Anti-alias for readability

On the properties pane of the text object, select **Classic Text**.

Set the application properties

1. Select File > AIR for Android Settings.
2. In the General tab, make the following settings:

- Output File: Hello.apk
- App name: Hello
- App ID: Hello
- Aspect ratio: Portrait


Application & Installer Settings

Output file: HelloWorld.apk

App name: HelloWorld

App ID: app. HelloWorld

Example: com.yourdomain.appname

Version: 1.0.0

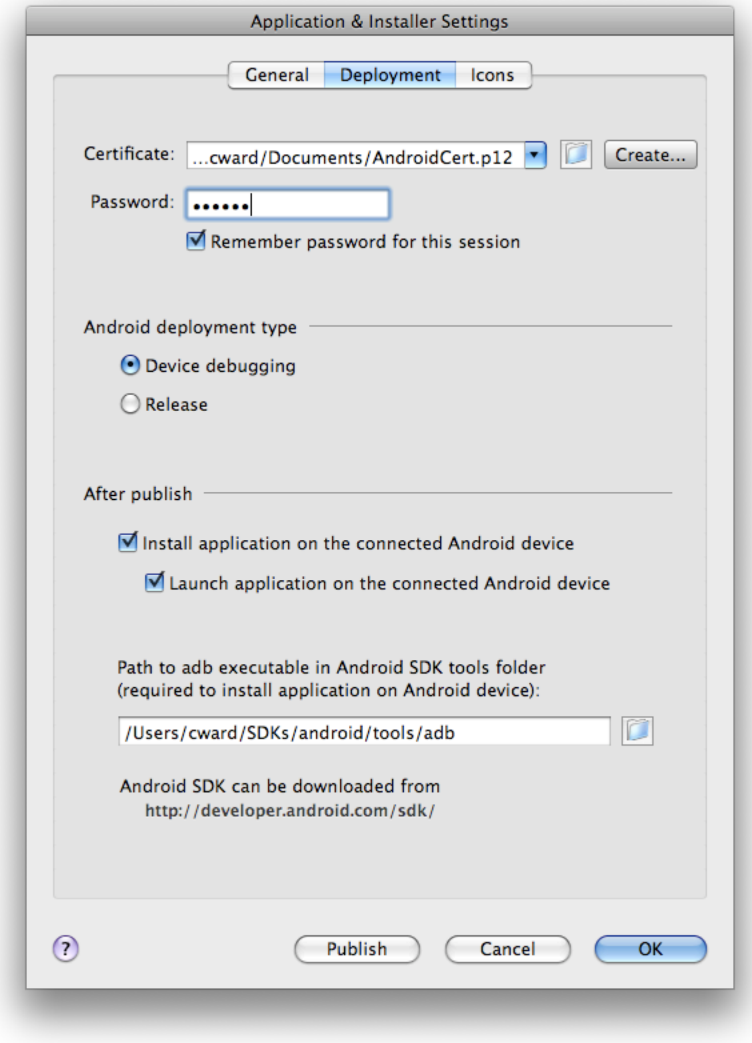
Aspect ratio: Portrait

Full screen

Auto orientation

Included files: HelloWorld.swf HelloWorld-app.xml

3. On the Deployment tab, make the following settings:
   - Certificate: Point to a valid AIR code-signing certificate. You can click the Create button to create a new certificate. (Android apps deployed via the Android Marketplace must have certificates that are valid for 25 years.) Enter the certificate password in the Password field.
   - Android deployment type: Release
   - After Publish: Select both options
   - Enter the path to the ADB tool in the tools subdirectory of the Android SDK.



Application & Installer Settings

Certificate: ...cward/Documents/AndroidCert.p12

Password: ••••••

[✓] Remember password for this session

Android deployment type

(●) Device debugging

( ) Release

After publish

[✓] Install application on the connected Android device

[✓] Launch application on the connected Android device

Path to adb executable in Android SDK tools folder
(required to install application on Android device):

/Users/cward/SDKs/android/tools/adb

Android SDK can be downloaded from
http://developer.android.com/sdk/

4. Close the Android settings dialog by clicking OK.
5. Open the application descriptor (located in the directory where you saved the .fla file).
6. Change the <version> tag to <versionNumber> and make sure that it contains a value of the form: number.number.number

   For example: <versionNumber>0.1.0</versionNumber>
7. Save the file.

Package and Install the application on the Android device

If you have not already done so, install Adobe AIR on the device. Make sure that you install the correct runtime for your device OS or emulator.

1. Make sure that USB debugging is enabled on your device. You can turn USB debugging on in the Settings app under Applications > Development.
2. Connect your device to your computer with a USB cable.
3. Open a command or terminal window.
4. Test the device connection by running the ADB devices command:

```
adb devices
```

If your device is not listed, make sure that USB debugging is enabled on your device. You can turn USB debugging on in the Settings app under Applications > Development.

5. Install the latest AIR runtime, if you have not already done so, using the ADB install command:

   ```
   adb install -r pathToRuntime/Runtime.apk
   ```

   When running ADB from the command line, it is often convenient to add the SDK tools folder to your path environment variable. For help on setting the path, see Setting the path environment variable.

6. Make sure that USB debugging is enabled on your device. You can turn USB debugging on in the device's Settings app under Applications > Development.
7. Connect the device to your computer with a USB cable.
8. Select File > Publish.

Flash Professional CS5 creates the APK file and installs the app on the connected Android device.

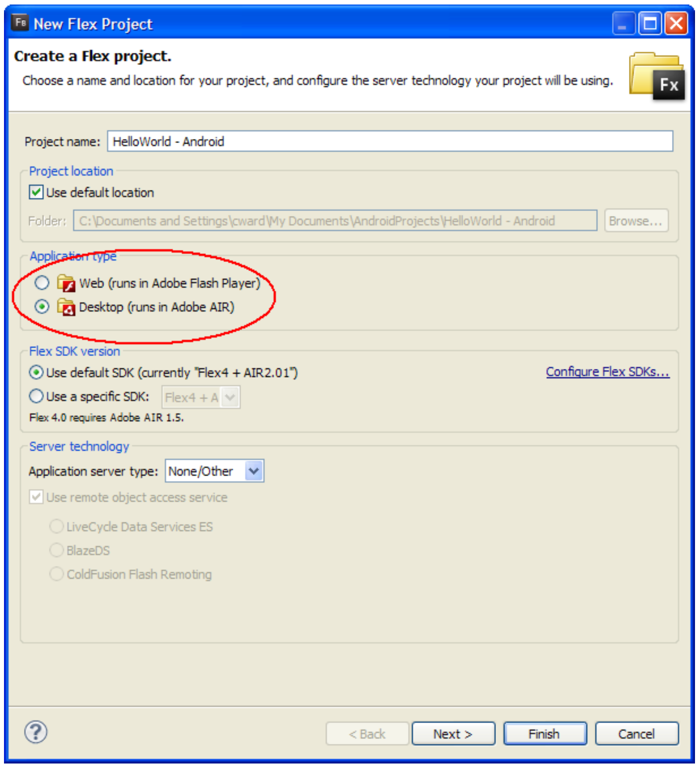
Hello World—Using Flash Builder

Flash Builder doesn't fully support the building and packaging of AIR for Android apps at this time. You can program and compile an ActionScript application (Flex is not recommended), but packaging the .apk file and installing it on a device must be done using command-line tools outside of Flash Builder.

Create a project

1. From the Flash Builder file menu, select New > Flex Project

   The New Flex Project opens:



New Flex Project

Create a Flex project.

Choose a name and location for your project, and configure the server technology your project will be using.

Project name: HelloWorld - Android

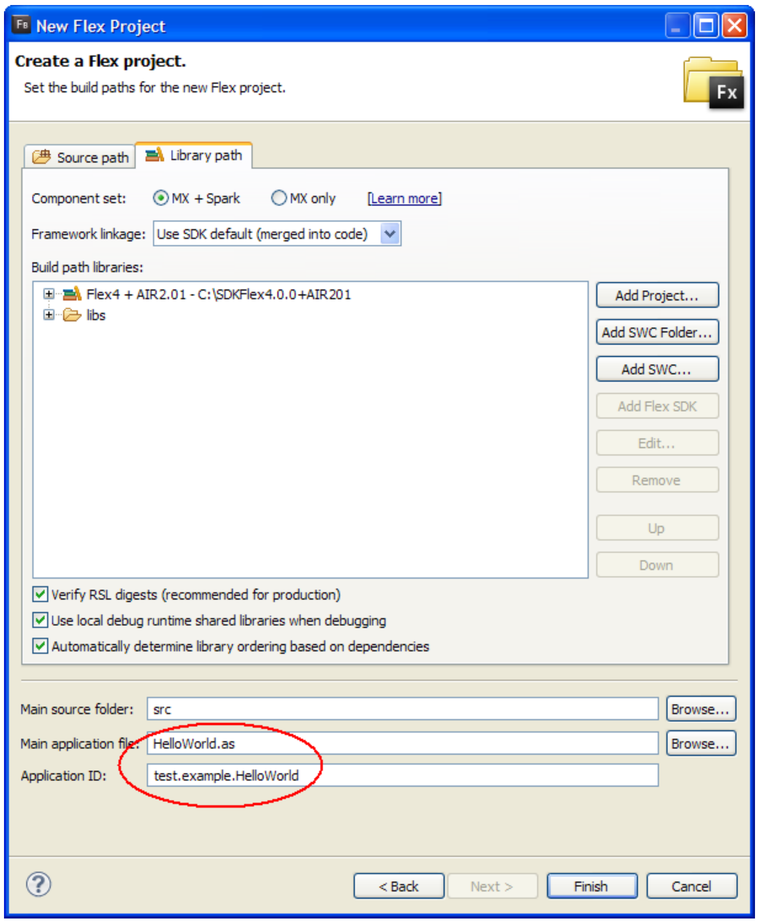
Project location
[✓] Use default location
Folder: C:\Documents and Settings\cward\My Documents\AndroidProjects\HelloWorld - Android

Application type
( ) Web (runs in Adobe Flash Player)
(●) Desktop (runs in Adobe AIR)

Flex SDK version
(●) Use default SDK (currently "Flex4 + AIR2.01")
( ) Use a specific SDK: Flex4 + A
Flex 4.0 requires Adobe AIR 1.5.

Server technology
Application server type: None/Other
[✓] Use remote object access service
( ) LiveCycle Data Services ES
( ) BlazeDS
( ) ColdFusion Flash Remoting

2. Enter a name for the project and choose the **Desktop** application type.
3. Use the Flex SDK containing the AIR 2.5 SDK for the project Flex SDK version. (Click Configure Flex SDK to add it to the list of available SDKs, if necessary.)
4. Click **Next** twice.



Component set: [✓ MX + Spark] MX only [Learn more]

Framework linkage: Use SDK default (merged into code)

Build path libraries:

- Flex4 + AIR2.01 - C:\SDKFlex4.0.0+AIR201
- libs

[✓] Verify RSL digests (recommended for production)

[✓] Use local debug runtime shared libraries when debugging

[✓] Automatically determine library ordering based on dependencies

Main source folder: src

Main application file: HelloWorld.as

Application ID: test.example.HelloWorld

5. Assign “HelloWorld.as” as the name of the main application file. Do not use .mxml as the extension.
6. Click **Finish** to create the project.

Write the code

For this simple exercise, just create a TextField object, assign it some text, and add it to the stage. The finished HelloWorld.as file should look like the following:

```
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 );
        }
    }
}
```

Edit the application descriptor file

Flash Builder automatically creates an application descriptor file for you. The <application> element of the descriptor should indicate that you are using AIR 2.5:

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

(If a different namespace is listed, you may not be using the correct AIR SDK.)

Although you can use the application descriptor file provided by Flash Builder as is, a few changes can make development easier. So, set visible to true, and supportedProfiles to `mobileDevice`. The finished application descriptor, with all the optional elements and comments removed, should look like the following:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/2.5">
    <id>test.example.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>HelloWorld</name>
    <versionNumber>0.1.0</versionNumber>
    <supportedProfiles>mobileDevice</supportedProfiles>
    <initialWindow>
        <content>
            <!--This value will be overwritten
            by Flash Builder in the output app.xml-->
        </content>
        <visible>true</visible>
    </initialWindow>
      </application>
```

This is a simple example. There are other settings that you can use in the application descriptor file. For example, you can add <fullScreen>true</fullScreen> to the <initialWindow> element to build a full-screen application. To enable remote debugging and access-controlled features on Android, you also will have to add Android permissions to the application descriptor. Permissions are not needed for this simple application, so you do not need to add them now.

Compile

Flash Builder automatically builds the project and places the result in the bin-debug folder. You should now have two files in this folder, HelloWord.swf and HelloWorld-app.xml.

Package

At this step, the process of creating an AIR for Android becomes different from your normal AIR-on-the-desktop workflow. To package an AIR for Android application, you must run the ADT tool from the command line. You will need a code signing certificate to complete this step. A self-signed certificate created with ADT is sufficient.

When running ADT from the command line, it is often convenient to add the SDK bin folder to your path environment variable. For help on setting the path, see Setting the path environment variable.

The following example demonstrates running ADT from the Windows command line. The procedure on Mac and Linux is essentially the same.

1. Open a command or terminal window.
2. Change the current directory to the bin-debug folder in your HelloWorld – Android project folder. For example:

   ```
   cd C:\ AndroidProjects\HelloWorld - Android\bin-debug
   ```

3. Run the ADT package command, setting the -target flag to apk:

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

The Android package, HelloWorld.apk, is created in the bin-debug directory.

Install on the device or emulator

Use the Android Debug Bridge (ADB) tool to install the Android package on your device.

1. Connect your device to your computer with a USB cable.
2. Open a command or terminal window.
3. Test the device connection by running the ADB devices command:

   ```
   adb devices
   ```

If your device is not listed, make sure that USB debugging is enabled on your device. You can turn USB debugging on in the Settings app under Applications > Development.

4. Install the Adobe AIR, if you have not already done so, using the ADB install command. Make sure that you install the correct runtime for your device or emulator.

   adb install -r *pathToRuntime*/Runtime.apk

5. Install the application with the ADB install command:

   adb install -r *pathToApp*/HelloWorld.apk

Note, if you have both an emulator running and a device attached, add either the -d or -e flag before the install command to specify which target to install to. For example, to install to the device, use:

adb -d install -r *pathToApp*/HelloWorld.apk

Before you can reinstall an app, you must remove the current version. You can use the -r flag for the adb install command (as shown in the examples) or uninstall from the Android Settings app, using the Applications>Manage Applications section.

Launch

To launch the HelloWorld application in the same way you launch any other installed app.

Hello World—Using the Flex and AIR SDK command-line tools

To create an AIR for Android application without using Flash Builder or Flash Professional, you can use the command-line tools provided in the Flex and AIR SDKs. You should overlay a working copy of the Flex SDK with the latest prerelease AIR SDK. This example uses amxmlc from the Flex SDK and ADT from the AIR SDK.

Setup

Before getting started, you must download the AIR 2.5 SDK from the Adobe prerelease website. Make a copy of the Flex SDK you want to use and overlay the new AIR SDK over it. Instructions are included in the release notes. You can obtain a new copy of the Flex SDK from http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+4, if needed.

You will also need to download the Android SDK tools. See the *Developing AIR applications for Android* document, available on the Adobe prerelease website for more information about getting and using the Android tools.

You must have Java version 1.6 or later installed on your development computer. You can obtain the latest version of Java from http://www.java.com/en/download. Mac OS ships with a version of Java installed.

When running the development tools from the command line, it is often convenient to add the SDK folders containing the tools to your path environment variable. For help on setting the path, see Setting the path environment variable.

Create a project

Create a folder in a convenient location. Create two files within this folder:

- HelloWorld.as
- HelloWorld-app.xml

Write the code

Open HelloWorld.as in an text editor and add the following code:

```
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 );
        }
    }
}
```

Edit the application descriptor file

Open the HelloWorld-app.xml file in a text editor and copy the following XML code into it:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/2.5">
    <id>test.example.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>HelloWorld</name>
    <versionNumber>0.1.0</versionNumber>
    <supportedProfiles>mobileDevice</supportedProfiles>
    <initialWindow>
```

```
        <content>HelloWorld.swf</content>
        <visible>true</visible>
    </initialWindow>
</application>
```

This is a simple example. There are other settings that you can use in the application descriptor file. For example, you can add `<fullScreen>true</fullScreen>` to the `initialWindow` element to build a full-screen application. To enable remote debugging and access-controlled features on Android, you also will have to add Android permissions to the application descriptor. Permissions are not needed for this simple application, so you do not need to add them now.

Compile

Use amxmlc from the Flex SDK to compile the app into a SWF file.

When running the Flex and AIR tools from the command line, it is often convenient to add the SDK bin path to your path environment variable (see Setting the path environment variable for help). The rest of this example demonstrates running the tools from the Windows command line. The procedure on Mac and Linux is essentially the same.

1. Open a command or terminal window.
2. Change the current directory to the folder in which you created the HelloWorld.as file. For example:

   ```
   cd C:\AndroidProjects\HelloWorld - Android
   ```

3. Run amxmlc to compile the application:

   ```
   amxmlc HelloWorld.as
   ```

The SWF file is created. You can run the application on the desktop using ADL, if desired:

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

Package

To package an AIR for Android application, use the AIR ADT tool. You will need a code signing certificate to complete this step. A self-signed certificate created with ADT is sufficient. (Instructions for creating a code signing certificate are included in Building Adobe AIR applications.)

1. Open a command or terminal window.
2. Change the current directory to the folder in which you created the HelloWorld.as file.
3. Run the ADT package command, setting the target flag to *apk*:

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

The Android package, HelloWorld.apk, is created in the bin-debug directory.

Install on the device

Use the Android Debug Bridge (ADB) tool to install the Android package on your device.

The ADB tool is part of the Android SDK, which you can download from: http://developer.android.com/sdk/index.html. Once you have the Android SDK installed, you can continue.

1. Connect your device to your computer with a USB cable.
2. Open a command or terminal window.
3. Test the device connection by running the ADB devices command:

   ```
   adb devices
   ```

   If your device is not listed, make sure that USB debugging is enabled on your device. You can turn USB debugging on in the Settings app under Applications > Development.

4. Install the AIR runtime, if you have not already done so, using the ADB install command. Make sure you install the correct runtime for your device or emulator.

    ```
    adb install -r pathToRuntime/Runtime.apk
    ```

5. Install the application with the ADB install command:

    ```
    adb install -r pathToApp/HelloWorld.apk
    ```

Note, if you have both an emulator running and a device attached, add either the -d or -e flag before the install command to specify which target to install to. For example, to install to the device, use:

```
adb -d install -r pathToApp/HelloWorld.apk
```

Before you can reinstall an app, you must remove the current version. You can use the -r flag for the adb command (as shown in the examples) or uninstall from the Android Settings app, using the Applications>Manage Applications section.

Setting the path environment variable

Setting the PATH on Linux and Mac OS using the Bash shell

When you type a command in a terminal window, the shell, a program that reads what you typed and tries to respond appropriately, must first locate the command program on your file system. The shell looks for commands in a list of directories stored in an environment variable named $PATH. To see what is currently listed in the path, type:

```
echo $PATH
```

This returns a colon-separated list of directories that should look something like this:

```
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin
```

The goal is to add a couple of directories to the list so that the shell can find the ADT tool from the AIR SDK and the ADB tool from the Android SDK. Assuming that you have put the AIR SDK at /Users/fred/SDKs/AIR and the Android SDK at /Users/fred/SDKs/android, then the following command will add the necessary directories to the path:

```
export PATH=$PATH:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

Note: If your path contains blank space characters, escape them with a backslash, as in the following:

```
/Users/fred\ jones/SDKs/AIR\ 2.5\ SDK/bin
```

You can use the echo command again to make sure it worked:

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

So far so good. You should now be able to type the following commands and get an encouraging response:

```
adt -version
adb devices
```

If your modified your $PATH variable correctly, the first command should report the version of ADT. The second command should list the Android emulators and devices attached on your machine.

There is still one problem, however; the next time you fire up a new terminal window, you will notice that the new entries in the path are no longer there. The command to set the path must be run every time you start a new terminal.

A common solution to this problem is to add the command to one of the start-up scripts used by your shell. On Mac OS, you can create the file, .bash_profile, in the ~/username directory and it will be run every time you open a new terminal window. On Ubuntu, the start-up script run when you launch a new terminal window is .bashrc. Other Linux distributions and shell programs have similar conventions.

To add the command to the shell start-up script:

1. Change to your home directory:
   ```
   cd
   ```
2. Create the shell configuration profile (if necessary) and redirect the text you type to the end of the file. Use the appropriate file for your operating system and shell. You can use .bash_profile on Mac OS and .bashrc on Ubuntu, for example.
   ```
   cat >> .bash_profile
   ```
3. Type the text to add to the file:
   ```
   export PATH=$PATH:/Users/cward/SDKs/android/tools:/Users/cward/SDKs/AIR/bin
   ```
   End the text redirection by pressing **CTRL-SHIFT-D** on the keyboard.
4. Display the file to make sure everything is okay:

```
cat .bash_profile
```

5. Open a new terminal window to check the path:

```
echo $PATH
```

Your path additions should be listed.

If you later create a new version of one of the SDKs into different directory, be sure to update the path command in the configuration file. Otherwise, the shell will continue to use the old version.

Setting the PATH on Windows

When you open a command window on Windows, that window inherits the global environment variables defined in the system properties. One of the important variables is the path, which is the list of directories that the command program searches when you type the name of a program to run. To see what is currently included in the path when you are using a command window, you can type:

```
set path
```

This will show a list of semicolon-separated list of directories that looks something like:

```
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

The goal is to add a couple of directories to the list so that the command program can find the ADT tool from the AIR SDK and the ADB tool from the Android SDK. Assuming that you have put the AIR SDK at C:\SDKs\AIR and the Android SDK at C:\SDKs\android, you can add the proper path entries with the following procedure:

1. Open the System Properties dialog by right-clicking on the My Computer icon and choosing Properties from the menu.
2. Under the Advanced tab, click the **Environment Variables** button.
3. Select the Path entry in the System variables section of the Environment Variables dialog
4. Click **Edit**.
5. Scroll to the end of the text in the Variable value field.
6. Enter the following text at the very end of the current value:

```
;C:\SDKs\AIR\bin;C:\SDKs\Android\tools
```

7. Click **OK** in all the dialogs to save the path.

If you have any command windows open, realize that their environments are not updated. Open a new command window and type the following commands to make sure the paths are set up correctly:

```
adt -version
adb devices
```

If you later change the location of the SDKs, or add a new version, remember to update the path variable.

CHAPTER 2
Building AIR applications for Android

This chapter contains details on using the tools to build AIR applications for Android. For instructions on installing development tools and for Hello World tutorial samples, see Chapter 1.

Android icon art

The application launch icon should be supplied as 36x36-, 48x48-, and 72x72-pixel PNG images. These icon sizes are used for low density, medium density and high density screens, respectively. Specify the path to the icon files in the icon element of the application descriptor file:

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

If you do not supply an icon of a given size, the next largest size is used and scaled to fit. If you do not supply any icons, a default system icon is used.

In Flash Professional CS5, you can specify these icons in the AIR for Android Settings dialog box. Select File > AIR for Android Settings. (For AIR for Android support in Flash Professional CS5, install the AIR for Android prerelease extension for Flash Professional CS5, included at the prerelease site.)

Setting application properties

The application descriptor file is an XML file containing properties for the entire application, such as its name, version, copyright, and other settings. Note that the descriptor includes Android permissions for accessing the network and writing to the SDCard. Your application may need different permissions. The application descriptor file is XML file with the following format:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/2.5">
    <id>com.example.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <versionNumber>0.1.0</versionNumber>
    <supportedProfiles>mobileDevice</supportedProfiles>
    <initialWindow>
        <content>HelloWorld.swf</content>
        <visible>true</visible>
        <fullScreen>true</fullScreen>
        <aspectRatio>portrait</aspectRatio>
        <autoOrients>false</autoOrients>
    </initialWindow>
    <icon>
        <image36x36>assets/icon36.png</image36x36>
        <image48x48>assets/icon48.png</image48x48>
        <image72x72>assets/icon72.png</image72x72>
    </icon>
<android>
```

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

</application>
```

Note the AIR 2.5 namespace declaration. This is required for AIR apps on Android.

You can also set application properties in Flash Professional CS5.

Setting application properties in the application descriptor file

The application descriptor file includes the following elements.

The <id> element uniquely identifies your application. The recommended form is a dot-delimited, reverse-DNS-style string, such as "com.company.AppName". Do not include a hyphen character in the id. Note that the Android package ID is the value of the id element in the application descriptor file with "app." prepended to the front.

The <filename> element defines the name used for the APK package file.

The <name> element is the application name displayed on the device. Make sure that this name is not too long to be displayed.

The <version> element used in AIR 2.0 and earlier is no longer valid. Instead, replace <version> with <versionNumber> and <versionLabel>.

The <versionNumber> element can contain three integers, separated by periods. For example: <versionNumber>1.0.2</versionNumber>. Each segment of the version number can be up to three digits long.

The <versionLabel> element is optional and can be used to display an arbitrary label string to users in the AIR install dialogs. For example: <versionLabel>1.0.2 alpha</versionLabel>. If not used, the <versionNumber> value is displayed to users.

The <initialWindow> element contains the following child elements to specify the properties for of the initial appearance of the application:

The <content> element identifies the root SWF file of the application.

<visible>true</visible> This is a required setting.

The<fullScreen> element specifies whether the application uses the entire screen of the Android device (true) or not (false).

The <aspectRatio> element specifies that the initial aspect ratio of the application is in portrait mode (rather than landscape). You can specify <aspectRatio>portrait</aspectRatio> or <aspectRatio>landscape</aspectRatio>.

The <autoOrients> element specifies whether the orientation of content in the application automatically reorients as the device itself changes physical orientation. The default value is true. You can cancel automatic orientation by calling the preventDefault() method of an orientationChanging event dispatched by the Stage object.

When using auto-orientation, for best results set the align property of the Stage to the following:

```
stage.align = StageAlign.TOP_LEFT;
stage.scaleMode = StageScaleMode.NO_SCALE;
```

Specifying <supportedProfiles>mobileDevice</supportedProfiles> Limits the application to be compiled into the mobile device profile. There are three supported profiles:

- desktop—A desktop AIR application.
- extendedDesktop—A desktop AIR application with support for the native process API.
- mobileDevice—An AIR application for a mobile device.

On Android, you must support the mobileDevice profile. If the <supportedProfiles> tag is not included, then it is assumed that your application supports all profiles.

The <icon> element contains the following child elements to specify the icons used by the application:

```
<icon>
    <image32x32>assets/icon32.png</image36x36>
    <image48x48>assets/icon48.png</image48x48>
    <image72x72>assets/icon72.png</image72x72>
</icon>
```

Setting Android permissions

The Android security model requires that each app request permission in order to use features that have security or privacy implications. These permissions must be requested when the app is packaged, and cannot be changed at runtime. The Android operating system informs the user which permissions an app is requesting when the user installs it. If a the permission required for a feature is not requested, the Android operating system might throw an exception, which is transmitted by the runtime to your application, but an exception is not guaranteed. In some cases a feature can fail silently. (A permission failure message is added to the system log.)

In AIR, you specify the Android permissions inside the <android> tag of the application descriptor. The following format is used for adding permissions (where PERMISSION_NAME is the name of an Android permission):

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

The permissions statements inside the <manifest> element are added directly to the Android manifest document. No validation is performed.

The following permissions are required to use various AIR features:

WRITE_EXTERNAL_STORAGE	Allows the application to write to the external memory card on the device.
WAKE_LOCK and DISABLE_KEYGUARD	Allows the application to prevent the device from going to sleep using SystemIdleMode class or while video is playing.
INTERNET	Allows the application to make network requests. Allows remote debugging.
ACCESS_FINE_LOCATION	Allows the application to access GPS data through the

	Geolocation class.
READ_PHONE_STATE	Allows the AIR runtime to mute audio when an incoming call occurs.
CAMERA	Allows the application to access the camera.
RECORD_AUDIO	Allows the application to access the microphone.

For example, to set the permissions for an app that played video, you could add the following to the application descriptor:

```
<android>
    <manifestAdditions>
        <manifest>
        <data>
                <![CDATA[
                    <uses-permission android:name="android.permission.WAKE_LOCK" />
                    <uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
                    <uses-permission android:name="android.permission.INTERNET" />
                    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
                ]]>
            </data>
        </manifest>
    </manifestAdditions>
</android>
```

For more information on the Android security and permissions, see:

Security and Permissions (Android Developers website)

Manifest.permission class (Android Developers website)

Setting application properties in Flash Professional CS5

Flash Professional CS5 generates an application descriptor file based on the settings in the Application & Installer Settings dialog box. However, you can also edit the application descriptor file in a text editor (but don't edit in a text editor while the AIR Android settings dialog is open, or you risk overwriting your edits). Flash Professional names the application descriptor file by adding "-app.xml" to your project name. For example, the application descriptor file for a HelloWorld project is named HelloWorld-app.xml.

To set application properties in Flash Professional CS5, select File > AIR Android Settings. The Application & Installer Settings dialog box (for AIR Android apps), includes three tabs: General, Deployment, and Icons.

In the General tab, make the following settings:

- Output File: the name of the APK file
- App name: The name of the app, which is displayed on the Android device
- App ID: A unique name for the app, such as com.example.myApp
- Version The version of the app, such as 1.0.0
- Aspect ratio: The initial aspect ratio of the application

On the Deployment tab, make the following settings:

- Certificate: Point to a valid AIR code-signing certificate. You can click the Create button to create a new certificate. (Android apps deployed via the Android Marketplace must have certificates that are valid for 25 years.) Enter the certificate password in the Password field.

- Android deployment type: Release or debug
- After Publish: Options to install and launch the application on a connected device
- Enter the path to the ADB tool in the tools subdirectory of the Android SDK.

On the Icons tab, add the paths to the icon PNG files (defined in the previous section).

Packaging an Android application

In Flash Professional CS5, you can package the application by selecting File > AIR for Android Settings. (For AIR for Android support in Flash Professional CS5, install the AIR for Android prerelease extension for Flash Professional CS5, included at the prerelease site.)

Use the AIR ADT tool to package an AIR for Android application. The AIR SDK version 2.5 supports packaging for Android.

The primary difference between packaging an AIR application for Android versus packaging for the desktop is that you add the -target apk flag to the command line parameters. The command line syntax for packaging is:

```
adt -package -target ( apk | apk-debug ) ( CONNECT_OPTIONS ) SIGNING_OPTIONS <output-package> ( FILE_OPTIONS | <input-
    package> )
```

`SIGNING_OPTIONS` are the same as the normal AIR code signing parameters.

`output-package` is the name for the APK file. You can add the .apk extension to this name. If you specify no extension, the packager adds the .apk extension.

`FILE_OPTIONS` are the normal parameters for identifying the application descriptor file, output file, and the files to package. You must run this command from the directory containing the application files.

The following example assumes that:

1. The path to the ADT tool is on your path definition. (See Setting the path environment variable for help.)
2. You are running the command from the directory containing the application source files. The source files are myApp-app.xml (the application descriptor file), myApp.swf, and an icons directory.
3. You have a code-signing certificate named codesign.p12 in the parent directory. This certificate must be valid through 2033 in order for the app to be distributed through the Android Marketplace. (See “Distributing an AIR for Android application” on page 29.)

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

Running this example, ADT will ask you for the password. You can also specify the password right after the certificate name (codesign.p12 in this example).

You can also create an APK package from an AIR or AIRI file:

```
adt -target apk  -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp.air
```

The AIR file must use the AIR 2.5 namespace in the application descriptor file. The file should also include icons sized correctly for Android. (See “Android icon art” on page 21.)

To create a version of the app that you can use with a debugger, use apk-debug as the target and specify connection options. For more information, see “Debugging an application on the device” on page 28.

Note: Developer Serge Jespers has created *Package Assistant Pro*, an AIR application that assists you in packaging AIR apps for Android. You can download the application at http://www.webkitchen.be/package-assistant-pro/.

Installing Adobe AIR and AIR for Android apps

To install Adobe AIR or an AIR application to an Android device or the Android emulator, use the Android Debug Bridge (ADB) tool. The ADB tool is included in the Android SDK.

You can also install Android packages from a URL by placing the APK package on an HTTP server and navigating to the URL with the device web browser. To enable installation of Android packages directly from the web browser, open the Settings app on the device or emulator. In Applications > Development, select the Unknown Sources option.

Installing with the Android Debug Bridge (ADB)

You can use the ADB application, included in the Android SDK, to install an Android package:

1. Connect your Android device to your computer using a USB cable.
   On Windows, you might need to install the Android USB driver, which is available in the Android SDK.
   You can also launch the Android Emulator instead of connecting an Android device.
2. In the device Settings app, go to Applications > Development and enable USB debugging.
3. To test whether the ADB tool can connect to your device, run:

   ```
   adb devices
   ```

   You should see your device and any running emulators listed in the command output. (If your USB-connected device or running emulator isn't listed, run the ADB kill-server command and try again.)
4. To install the AIR runtime, run the install command. Use the -d flag to target a device; use -e to target an emulator:

   ```
   adb -d install -r Runtime_Device_yyyymmdd.apk
   adb -e install -r Runtime_Emulator_yyyymmdd.apk
   ```
5. To install an application run:

   ```
   adb -d install  -r myApp.apk
   adb -e install -r myApp.apk
   ```

Note: Use the -r flag with adb install (indicating a "reinstall"), as shown in the examples, to remove an already-installed version of the app.

See http://developer.android.com/guide/developing/tools/adb.html for ADB documentation on the Android Developers website.

Installing via a URL

To enable installation of Android packages directly from the web browser, open the Settings app on the device or emulator. In Applications > Development, select the Unknown Sources option.

To install an Android package from a URL:

1. Place the APK package on a web server that is accessible to your device.
2. On the device, open the web browser.
3. Enter the URL for the package in the web browser.

The device downloads the apk file and displays it on the browser's Download history page. Tap the APK entry to install it.

Running an Android application on a device

You can launch an Android application from the standard launch screen on the device.

You can also use the ADB tool to launch an application on a connected device. Use the following command:

```
adb -d shell am start -a android.intent.action.MAIN -n app.<app-id>/app.<app-id>.AppEntry
```

For example, consider an application in which the application descriptor file includes the following id element:

```
<id>com.example.test.ball</id>
```

You can launch this application (after it had been installed) using the following command:

```
adb -d shell am start -a android.intent.action.MAIN -n app.com.example.test.ball/app.com.example.test.ball.AppEntry
```

Running AIR applications in the Android emulator

To run your AIR application on the Android emulator, you must install the Android SDK, create at least one virtual device, install the emulator version of the AIR runtime, and then install your AIR application. Note that applications on an emulator typically run much slower than they do on an actual device.

You must have the Android SDK installed. See Installing the Android SDK on page 6.

Create the Android Virtual Device:

1. Launch the Android SDK and AVD Manager application:
   - On Windows, run the SDK Setup.exe file, at the root of the Android SDK directory.
   - On Mac OS, run the android application, in the tools subdirectory of the Android SDK directory
2. Select the Settings option and select the "Force https://" option.
3. Select the Available Packages option. You should see a list of available Android SDKs.
4. Select the Android 2.2 (API 8) option and click the Install Selected button.
5. Select the Virtual Devices option and click the New button.
6. Make the following settings:
   - A name for your virtual device
   - The target API, Android 2.2, API level 8
   - A size for the SD Card (such as 1024)
   - A skin (such as Default HVGA)
7. Click the Create AVD button.

Note that Virtual Device creation may take some time depending on your system configuration.

Now you can launch the new Virtual Device.

1. Select Virtual Device. The virtual device you created above should be listed.
2. Select the Virtual Device, and click the Start button.
3. Click the Launch button on the next screen.

You should see an emulator window. This too may take a few seconds. It may also take some time for the Android operating system to initialize.

You can check that the Android version on the Virtual Device is 2.2. Open the device Settings app and go to About Phone > Firmware Version.

You can use the ADB tool to install Adobe AIR and AIR for Android apps. Be sure to install Adobe AIR before installing any AIR for Android apps. See "Installing Adobe AIR and Android apps" on page 26.

You can now run AIR applications on the Android Virtual Device. See "Running AIR applications in the Android emulator" on page 27.

For more information, see the Android documentation at:

http://developer.android.com/guide/developing/tools/othertools.html#android

http://developer.android.com/guide/developing/tools/emulator.html

Debugging an application on the device

You can create a version of an app that you can install on the device and debug. You can debug using the remote debugging session feature in Flash Professional CS5. You can also use the fdb tool, included in the Flex SDK.

The debug session displays any trace() output from the app, and you can use other debugging features.

Compile the application with debug support.

Request the INTERNET permission in the application descriptor so that the Android operating system permits the app to access the network. You can request this permission by adding the following XML fragment to the descriptor:

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

In Flash Professional CS5, you can publish a debug version of the APK file:

1. Select File > AIR Android settings.
2. In the Deployment tab, specify Device Debugging as the Android Deployment Type.
3. Click the Publish button.

Using the adt tool, you can package the app as a debug version using the -target apk-debug option:

```
adt -package -target apk-debug CONNECT_OPTIONS SIGNING_OPTIONS <output-package> ( FILE_OPTIONS | <input-package> )
```

For a description of the SIGNING_OPTIONS, <output-package>, FILE_OPTIONS, and <input-package>, see Setting Application Properties.

Include one of the following -connect options (CONNECT_OPTIONS) to specify the IP address of the development computer running the debugger:

- -connect—The application will attempt to connect to a debug session on the development computer used to compile the application. For example:

  ```
  adt -package -target apk-debug -connect ...
  ```

- -connect IP_ADDRESS—The application will attempt to connect to a debug session on the computer with the specified IP address. For example:

```
adt -package -target apk-debug -connect 192.0.32.10
```

- -connect HOST_NAME—The application will attempt to connect to a debug session on the computer with the specified host name. For example:

```
adt -package –target apk-debug -connect bobroberts-mac.example.com
```

The -connect option is optional. If not included, the resulting debug application will not attempt to connect to a hosted debugger.

To run a debug session:

1. Install the debug version of the app on the device or the Android emulator.
2. On the device, turn Wi-Fi on and connect to the same network as that of the development computer.
3. Start a debug session using Flash Professional CS5 or the fdb application included in the Flex SDK.
   - In Flash Professional CS5, choose Debug > Begin Remote Debug Session > ActionScript 3.0.
   - From the command line, run the fdb application included in the bin directory of the Flex SDK.
4. Run the app on the device. (See "Running an Android application on a device" on page 27.)

When debugging an application installed on Android, Flash Professional CS5 and fdb support all debugging features, including trace() output, breakpoint control, stepping through code, and variable monitoring.

Distributing an AIR for Android application

Because the AIR application is packaged as a standard Android package, you can distribute it like any other Android application, either through the Android marketplace or another website. However, the general public cannot install AIR for Android applications until the AIR runtime is publicly available.

The Android marketplace currently requires that submitted apps be signed with a certificate valid until at least 2033. (Such a long-lived certificate is not required for developing an APK; this is only a requirement for packages submitted to the Android marketplace.)

You can create a self-signed certificate with the Adobe ADT tool. Use the new -validityPeriod option to extend the certificate expiration date. For example, the following command creates a certificate that is valid for 25 years:

```
adt -certificate -cn ADigitalID 1024-RSA SigningCert.p12 39#wnetx3tl
```

By default, certificates created with ADT are valid for five years. Be sure to use the -validityPeriod option when creating a certificate for use with an AIR for Android application.

CHAPTER 3
Supported Flash and AIR APIs

AIR for Android conforms to the AIR mobile device profile. Not all APIs that are supported on the desktop are supported on mobile devices. The following table lists the features available in the supported AIR profiles. If a feature says "Check" in the profile column, then some devices in that profile may not support the feature. It is a good practice to check the listed support property before using the feature.

Note: This table lists the level of ActionScript support planned for the final release of AIR for Android. Not all of these features may be implemented or usable in the current prerelease build of AIR. For current API and development issues, refer to the release notes on the Adobe prerelease website.

Class or Feature	**desktop**	**extendedDesktop**	**mobileDevice**
Accelerometer (Accelerometer.isSupported)	No	No	Check
ActionScript 2	Yes	Yes	No
Cache bitmap matrix	No	No	Yes
Camera (Camera.isSupported)	Yes	Yes	Check
CameraRoll	No	No	Yes
ContextMenu (ContextMenu.isSupported)	Yes	Yes	No
DatagramSocket (DatagramSocket.isSupported)	Yes	Yes	No
DockIcon (NativeApplication.supportsDockIcon)	Check	Check	No
Drag-and-drop (NativeDragManager.isSupported)	Yes	Yes	Check
EncyptedLocalStore (EncyptedLocalStore.isSupported)	Yes	Yes	No
Flash Access (DRMManger.isSupported)	Yes	Yes	No
Geolocation (Geolocation.isSupported)	No	No	Check

HTMLLoader (HTMLLoader.isSupported)	Yes	Yes	No
IME (IME.isSupported)	Yes	Yes	Check
LocalConnection (LocalConnection.isSupported)	Yes	Yes	No
Microphone (Microphone.isSupported)	Yes	Yes	Check
NativeMenu (NativeMenu.isSupported)	Yes	Yes	No
NativeProcess (NativeProcess.isSupported)	No	Yes	No
NativeWindow (NativeWindow.isSupported)	Yes	Yes	No
NetworkInfo (NetworkInfo.isSupported)	Yes	Yes	Check
Open files with default application	Limited	Yes	No
PrintJob (PrintJob.isSupported	Yes	Yes	No
SecureSocket (SecureSocket.isSupported)	Yes	Yes	No
ServerSocket (ServerSocket.isSupported)	Yes	Yes	No
Stage orientation (Stage.supportsOrientationChange)	No	No	Yes
Start application at login (NativeApplication.supportsStartAtLogin)	Yes	Yes	No
StorageVolumeInfo (StorageVolumeInfo.isSupported)	Yes	Yes	No
System idle mode	No	No	Yes
SystemTrayIcon (NativeApplication.supportsSystemTrayIcon)	Check	Check	No
Text Layout Framework input	Yes	Yes	No
Updater (Updater.isSupported)	Yes	No	No
XMLSignatureValidator (XMLSignatureValidator.isSupported)	Yes	Yes	No

ActionScript API Notes

The following APIs behave differently on Android devices than is currently documented in the ActionScript 3.0 Reference for the Adobe Flash Platform. The differences are:

File System

File.applicationDirectory points to `/data/data/app.appId/app/assets`

File.applicationStorageDirectory points to `/data/data/app.appID/appID/Local Store`

File.desktopDirectory points to `/sdcard`

File.documentsDirectory points to `/sdcard`

File.createTempDirectory() created in `/data/data/app.appId/cache`

File.createTempFile() created in `/data/data/app.appId/cache`

app:/ points to `/data/data/app.appId/app/assets`

app-storage:/ points to `/data/data/app.appId/appId/Local Store`

FileReference.browse() can only select images, video and audio files.

FileReference.browse() the title cannot be changed.

FileReference opens up /sdcard for its operations.

File.moveToTrash() and File.deleteFile() both delete files immediately, since there is no recycle bin. Similarly, File.moveToTrashAsync() and File.deleteFileAsync() behave in the same way.

Hardware Keyboard

Keyboard events are dispatched for the Back, Search, and Menu “soft keys.” The Keyboard class defines constants for these keys:

- Keyboard.BACK
- Keyboard.MENU
- Keyboard.SEARCH

Use these constants instead of the numeric keycodes.

Loader

In AIR for Android, you *can* use the Loader class to load a SWF file and execute its code. However, ActionScript 2.0 is not currently supported. AIR applications for the iPhone cannot execute code in loaded SWF content; AIR for Android does not include this restriction.

Stage orientation

The Stage orientation property and setOrientation() method are deprecated on all devices and do not function on Android.

CHAPTER 4
Designing AIR applications for Android

The processing speed and screen size of mobile devices lend to special design and coding considerations. However, many of the design considerations are common to development for all applications or for mobile applications.

For more information on optimizing applications, see "Optimizing Content for the Flash Platform," at http://help.adobe.com/en_US/as3/mobile/index.html. This document includes many suggestions for optimizing performance of mobile content, Flash Player content, AIR content, and ActionScript-based content in general. Most of these suggestions also apply to AIR applications for mobile devices.

Important: Many of these design considerations and optimization techniques are essential in developing applications that run well on mobile devices.

Improve display object performance

Hardware acceleration is now supported on current pre-release of android, but in general, you can speed up graphics performance in some classes of display objects. Here are a few tips on how to maximize graphics performance.

Try to limit the numbers of items visible on stage. Each item takes some time to render and composite with the other items around it. When you no longer need to display a display object, set its visible property to false or remove it from the stage (removeChild()). Do not simply set its alpha property to 0. Restrict upon the use of global objects and create objects restricted within a block of code.

Avoid blend modes in general, and the layer blend mode in particular. Use the normal blend mode whenever possible.

Display object filters are expensive computationally. Use them sparingly. For example, using a few filters on an introduction screen may be acceptable. However, avoid using filters on many objects or on objects that are being animated or when you must use a high frame rate.

Avoid morph shapes.

Avoid using clipping.

If possible, set the repeat parameter to false when calling the Graphic.beginBitmapFill() method.

Don't overdraw. Use the background color as a background. Don't layer large shapes on top of each other. There is a cost for every pixel that must be drawn.

Avoid shapes with long thin spikes, self -intersecting edges, or lots of fine detail along the edges. These shapes take longer to render than display objects with smooth edges.

Make bitmaps in sizes that are close to, but less than, 2n by 2m bits. The dimensions do not have to be power of 2, but they should be close to a power of 2, without being larger. For example, a 31-by-15–pixel image renders faster than a 33-by-17–pixel image. (31 and 15 are just less than powers of 2: 32 and 16.)

Limit the size of display objects and try to narrow down to an optimal size which is visible enough.

Enable hardware graphics acceleration by including <renderMode>gpu</renderMode> in the application descriptor.

Enable cacheAsBitmapMatrix for display objects.

Using cacheAsBitmapMatrix

To activate cacheAsBitmapMatrix, set both the cacheAsBitmapMatrix and cacheAsBitmap properties of a DisplayObject to true. Setting cacheAsBitmap to true allows the runtime to move a display object (on pixel boundaries) without redrawing the display object graphics. Setting cacheAsBitmapMatrix to true allows the runtime to move (to any x, y location), scale, skew or rotate a display object without redrawing it.

When a display object has children, then you can set cacheAsBitmap and cacheAsBitmapMatrix to true on it if the children do not move or perform any transform independently of the parent. When you set cacheAsBitmap on a parent, all children of the parent are drawn into the same shared off-screen bitmap. Including children in the parent reduces memory overhead, but transforming a child display object relative to the parent will force the runtime to redraw the off-screen bitmap. You should consider the parent display object and its children to be a single "tile" that can be transformed as one unit efficiently. If you need to transform the child objects independently, you should reorganize the parent-child relationship so that you can set the cacheAsBitmap and cacheAsBitmapMatrix properties of the two display objects to true separately. This separation creates separate off-screen bitmaps for each display object.

The memory used by each off-screen bitmap is (4 * width * height * antialiasFactor) bytes. The factor of 4 is because of the color depth of 32bits per pixel. Width and height are for the bounding rectangle of the display object and its children. The antialiasFactor depends on the quality setting of the player, which defaults to high, or 4, in Android. Thus a 10x10 pixel graphic would use 1600 bytes of memory.

When tweening, enabling cacheAsBitmapMatrix should improve performance as long as the tween involves movement, scaling, skewing, or rotation in the x-y plane. If the tween involves other operations, such as color transforms (except manipulating alpha only), then the display object will be redrawn and the benefits of using cacheAsBitmapMatrix are lost.

The performance improvement from enabling cacheAsBitmapMatrix depends upon the complexity of the objects being rasterized. The greater the number of nested display objects, vectors, and the larger the bounds, the greater the CPU saving of caching it.

Information density

信息密度

The physical size of the screen of mobile devices is smaller than on the desktop, although their pixel density is higher. Sharper text is nice to look at, but the glyphs have to have a minimal physical size to be legible. Mobile devices are often used on the move and under poor lighting conditions. Consider how much information you can realistically display onscreen legibly. It might be less than you would on a screen of the same pixel dimensions on a desktop.

Use typographic hierarchy to highlight important information. Use font size, weight, placement, and spacing to express the relative importance of the elements of the user interface. You can use one or more cues at each level of the hierarchy. Apply these cues consistently across your application. A cue can be spatial (indent, line spacing, placement) or graphic (size, style, color of typeface). Applying redundant

cues can be an effective way to make sure that the hierarchy is expressed clearly. However, try using no more than three cues for each level of grouping.

Try to simplify the labels and explanatory text required. For example, use sample input in text field to suggest the content and avoid a separate label.

Fonts and text input

The following fonts are device fonts on Android (check the built-in fonts in /system/fonts folder on your target phone):

- Clockopia.ttf
- DroidSerif-Bold.ttf
- DroidSans-Bold.ttf
- DroidSerif-BoldItalic.ttf
- DroidSans.ttf
- DroidSerif-Italic.ttf
- DroidSansFallback.ttf
- DroidSerif-Regular.ttf
- DroidSansMono.ttf

Use fonts that are 14 pixels or larger.

Use device fonts for editable text fields. Device fonts in text fields also render more quickly than embedded fonts.

Application design considerations

Consider implementing alternatives to using input text fields. For example, to have the user enter a numerical value, you do not need a text field. You can provide two buttons to increase or decrease the value (something like a custom numeric stepper).

Be aware of the space used by the virtual keyboard. When the virtual keyboard is activated (for example when a user taps within a text field), the application adjusts the position of the stage. The automatic repositioning ensures that the selected input text field is visible.

A text field at the top of the stage moves to the top of the visible stage area. (The visible stage area is smaller to accommodate the virtual keyboard.)

A text field at the bottom of the stage stays at the bottom of the new stage area.

A text field in another part of the stage is moved to the vertical center of the stage.

You can add an event listener for the focus event of a text field, to reposition the text field.

A single-line text field includes a clear button (to the right of the text) when the user edits the text. However, this clear button is not displayed if the text field is too narrow.

After editing text in a single-line text field, the user dismisses the virtual keyboard by tapping the key on the keyboard.

After editing text in a multi-line text field, the user dismisses the virtual keyboard by tapping outside of the text field. This removes focus from the text field. Make sure that your design includes area outside of the text field when the virtual keyboard is displayed. If the text field is too large, no other area may be visible.

Using some Flash Professional CS5 components can prevent you from removing focus from a text field. These components are designed for use on desktop machines, where this focus behavior is desirable. One such component is the TextArea component. When it is in focus (and being edited), you cannot remove focus by clicking another display object. Placing some other Flash Professional CS5 components onstage can also prevent the focus from changing from the text field being edited.

Do not rely on keyboard events. For example, some SWF content designed for the web uses the keyboard to let the user control the application. However, on Android, the virtual keyboard is only present when the user edits a text field.

Saving application state

Your application may quit at any time (for example when the phone rings or you app goes to background). The application dispatches a deactivate event whenever you application goes to background and an activate event when it comes back to the top again.

Consider saving the state of your application anytime you receive a deactivate event. For example, you can save settings to a file or a database in the application storage directory. Or you can save data to a local shared object. You can then restore the state of your application when you receive an activate event (obviously there's no guarantee here that your app won't be killed by the OS when not in focus).

If a phone call interrupts an application, it will restart when the call ends. Do not rely on the NativeApplication object dispatching an exiting event when the application quits; it may not as the support for this event hasn't been decided yet and may be missing in the final release too.

Screen orientation changes

Android application content can be viewed in portrait or landscape orientation. Consider how your application will deal with screen orientation changes. For more information, see "Setting an Detecting Screen Orientation."

Hit targets

Consider the size of hit targets when designing buttons and other user interface elements that the user taps. Make these elements large enough that they can be comfortably activated with a finger on a touch screen. Also, make sure that you have enough space between targets. Hit targets should be about 44 pixels to 57 pixels. For more information, see Hit targets on touch screens in the "Optimizing Mobile Applications" guide at http://help.adobe.com/en_US/as3/mobile/index.html.

Memory allocation

Allocating fresh blocks of memory is costly. It can slow down your application or cause performance to lag during animation or interaction as the garbage collection gets triggered.

Try to recycle objects whenever you can, rather than getting rid of one and creating a new one.

Keep in mind that vector objects can consume less memory than arrays. See Vector class versus Array class. For more information on memory usage, see Conserving memory in the "Optimizing Mobile Applications" at http://help.adobe.com/en_US/as3/mobile/index.html.

Drawing API

Try to avoid using the ActionScript drawing API (the Graphics class) to create graphics. Using the drawing API creates objects dynamically on the stage, then renders them to the rasterizer. If possible, create those objects statically in authoring time on the stage instead.

Objects created using drawing APIs, when repeatedly called, are destroyed and recreated every time actionscript is executed. However, static objects reside in memory through different timelines.

Event bubbling

For a deeply-nested display object container, bubbling of events can be expensive. Reduce this expense by handling the event completely in the target object, then calling the stopPropagation() method of the event object. Calling this method prevents the event from bubbling up. Calling this method also means that parent objects do not receive the event.

Related gains can be realized by flattening the display object nesting, to avoid long event chains.

Register for MouseEvent events instead of TouchEvent events when possible. MouseEvent events use less processor overhead than TouchEvent events.

Set the mouseEnabled and mouseChildren properties to false, when possible.

Optimizing video performance

To optimize mobile video playback, ensure that there is little else going on in your application while videois playing. This allows the video decoding and rendering processes to use as much CPU as possible.

Have little or no ActionScript code running while the video plays. Try to avoid running code that runs on a frequent interval timer or on the timeline. Minimize the redrawing of non-video display objects. Especially avoid redrawing display objects that intersect with the video area. This is true even if they are hidden underneath the video. They will still be redrawn and use up processing resources. For example, use simple shapes for the position indicator and update the position indicator just a couple of times a second rather than on every frame. Don't have the video controls overlapping the video area; put them directly below. If you have a video buffering animation, don't hide it behind the video when it is not in use; set it to invisible.

The deactivate/activate event here can be put to use judiciously over here. Whenever you receive a deactivate event you can pause the audio/video playback. Similarly, if you app uses some heavy graphics(I refere to a high frame rate here), it's always better to reduce it as much as possible. You may resume or restore back the application state once you receive an activate event.

The above not only reduce your memory resource consumption but also ensure lesser battery use for the end user.

Reducing application file size

Here are some tips on reducing the file size of your apk file:

- Check background bitmaps to see that they are the right size (not larger than needed).
- Check to see if any extra fonts are being embedded.
- Look at PNG assets for alpha channels and remove them if they are not needed. Use a utility like PNG crunch to reduce the size of PNG assets.
- Convert PNG assets to JPG assets, if possible.
- Consider compressing sound files (by using a lower bit rate)
- Remove any assets that are not used.

Differences between Android and iPhone development

If you are already familiar with the process of creating AIR applications for the iPhone, be aware of the following differences when creating an application for Android:

- The runtime is compiled into AIR for iPhone applications. However, on Android, all the AIR applications share the same runtime. This fact makes Android applications much, much smaller. However, it also means that you must wait for the runtime to be publically available before you can distribute your application.
- You do not need a special developer certificate or provisioning profile on Android. You can sign an Android application with a normal AIR development certificate. (However, Android requires that the certificate remain valid through 2033. For more information, see "Distributing an AIR for Android application" on page 29.)
- Icons are optional on Android. (However, it is certainly recommended that you use a custom launch icon. The ADT tool provides a default icon if you do not provide one.)
- A graphic for the initial screen (Default.png) is not used by Android apps.
- Android devices come in a wider variety of screen sizes and with a wider range of capabilities. For tips on developing for multiple screen sizes, see "Authoring mobile Flash content for multiple

screen sizes" at
http://www.adobe.com/devnet/flash/articles/authoring_for_multiple_screen_sizes.html.

Miscellaneous development notes

You can use the ADB logcat command to view the device system event log. This log can reveal system errors that affect your application.

On Android, the convention is that there is no exit button for an application. When a user switches out of your application, it goes to the background, but continues running. If the operating system runs low on resources, it will then shut down apps. You can detect when your app goes to the background and returns, by listening to deactivate and activate events dispatched by the NativeApplication object. You can shut down your app by calling the NativeApplication.exit() method.

The elements of the application descriptor file that specify window properties are ignored. These elements include: systemChrome, visible, transparent, resizable, maximizable, minimizable, minSize, and maxSize.

Every Android application has a package name. The ADT tool creates this name by appending "app." to the AIR application ID defined in the application descriptor file. Do not add "app." to the beginning of the applicationID element in the application descriptor file.

You can use the following ADB command to list packages installed on a device:

```
adb shell pm list packages
```