14
Using Native Extensions and ADT

In this chapter, we will cover:

- Compiling from the command line using Windows
- Compiling from the command line using Mac OS X
- Using a native extension
- Packaging a native extension
- Enabling Interpreter Mode
- Running ADT outside your Flash Professional CS5 installation

Introduction

The Flash platform is constantly evolving. Adobe is particularly focused on AIR, with new releases of the SDK being made on a regular basis. While this momentum is generally a good thing, it can leave AIR's integration with Flash Professional, which sees lengthier development cycles, out of sync. This is evident by the fact that Flash Professional CS5.5, by default, uses the older AIR 2.6 SDK and has no direct support for more recent features of the AIR Development Tool (ADT). Those using Flash Professional CS5 are restricted further—with AIR 2.0 being the only available option.

This chapter will cover how the command line can be used to compile `.swf` files into iOS applications rather than using Flash Professional directly. For CS5.5 users, we will also see how the command line can be used to provide support for ActionScript native extensions and to create **Interpreter Mode** builds. The command line is equally important for CS5 users as it will not only allow the creation of Interpreter Mode builds, but also enable applications to take advantage of the improved rendering engine provided by AIR 2.6 and above.

Compiling from the command line using Windows

Throughout the course of this book, we have concentrated exclusively on the compilation of `.ipa` files directly from the Flash IDE. However, it is also possible to compile from the command line. Let us see how this is done.

The steps covered here are for Microsoft Windows. If you are using Mac OS X, then refer to the *Compiling from the command line using Mac OS X* recipe.

Getting ready

The command line compiler is a Java application and requires the Java runtime environment in order to run. If you don't already have Java installed on your system, then download it from `www.java.com/getjava`.

While AIR 3.0 and above supports both the 32-bit and 64-bit versions of the Java executable, older versions of AIR only support 32-bit Java. We will stick to the 32-bit version throughout this chapter.

An FLA has been provided for this recipe and should be copied to your `Documents` folder. To do this, navigate to your `Documents` folder using Windows Explorer and create the following folder structure within it: `packt\flash-ios-cookbook\chapter14\`. Now from the book's accompanying code bundle, copy `chapter14\recipe1\` to `<documents_folder>\packt\flash-ios-cookbook\chapter14\`.

Throughout this recipe, replace `<documents_folder>` with the path to your `Documents` folder. Its location depends on the version of Windows you are running. For Windows Vista and Windows 7, it can be found at `C:\Users\<username>\Documents\`. If you are using Windows XP, then it is at `C:\Documents and Settings\<username>\My Documents\`.

Incidentally, the FLA you will be working from is a version of the Bubbles app from *Chapter 3*.

How to do it...

A `.swf` file is required when compiling a native iOS app from the command line. Let us begin by publishing a SWF from our FLA:

1. Launch Flash Professional and open `chapter14\recipe1\bubbles.fla` from your `Documents` folder.

2. Generate a SWF by selecting **Control** | **Test Movie** | **in AIR Debug Launcher (Mobile)**. The SWF will output to the same folder as the FLA and ADL will launch allowing you to preview it. Close the preview window once you are satisfied that publication was successful.
3. Open a command prompt window.

> How you open a command prompt window depends on your version of Microsoft Windows.
>
> If you are a Windows 7 or Vista owner, then click on the Windows **Start** button and type **cmd** into the search box, followed by *Enter*. If you are using Windows XP, click on the Windows **Start** button and select **Run**. Then from the Run dialog box, type **cmd** and press *Enter*.

4. Move to the folder where your `.swf` file is located by entering the following into the command line:

   ```
   cd <documents_folder>\packt\flash-ios-cookbook\chapter14\recipe1
   ```

> In the line above, remember to replace `<documents_folder>` with the path to your `Documents` folder. Its location depends on the version of Windows you are running.

5. Okay, now we are ready to compile an IPA from the command line. Let us take a look at the general format of the command you will be entering:

   ```
   [JAVA_FILE] -jar
   [ADT_FILE]  -package
               -target [DEPLOYMENT_TYPE]
               -provisioning-profile [PROV_FILE]
               -storetype pkcs12 -keystore [CERT_FILE]
               -storepass [PASSWORD]
               [OUTPUT_IPA]
               [APP_DESC_FILE]
               [SRC_FILES]
   ```

   Each of the capitalized blocks will need to be replaced with the appropriate file path or option. Here is what each block represents:

   - `[JAVA_FILE]`: Your computer's 32-bit Java application launcher
   - `[ADT_FILE]`: Either the AIR Development Tool (ADT) or the Packager for iPhone (PFI)
   - `[DEPLOYMENT_TYPE]`: The deployment type to be used
   - `[PROV_FILE]`: The provisioning profile associated with your app

- `[CERT_FILE]`: The P12 certificate file required to digitally sign your app
- `[PASSWORD]`: The password you associated with your P12 certificate
- `[OUTPUT_IPA]`: The name of the IPA file to be compiled
- `[APP_DESC_FILE]`: Your FLA's application descriptor file
- `[SRC_FILES]`: The SWF file published from your FLA followed by any other assets to be included within the IPA

Let us work through each of them, one at a time.

Find the location of the 32-bit Java executable on your computer and replace `[JAVA_FILE]` with it. Here is an example path to the file: `"C:\Program Files (x86)\Java\jre6\bin\java"`. Also notice the quotation marks placed around the file path. These are required when working with paths that contain spaces.

> The `Program Files (x86)` folder will simply be named `Program Files` on 32-bit versions of Microsoft Windows.
>
> Also, the path to the Java executable may differ slightly depending on the current version of the Java runtime environment that you have installed. This chapter will assume you are using Java 6.

Next you need to locate either the AIR Development Tool (ADT) or the Packager for iPhone (PFI) depending on the version of AIR you're using. If you're using Flash Professional CS5 then replace `[ADT_FILE]` with the file path to the PFI: `"C:\Program Files (x86)\Adobe\Adobe Flash CS5\PFI\lib\pfi.jar"`. For CS5.5, replace `[ADT_FILE]` with ADT's file path: `"C:\Program Files (x86)\Adobe\Adobe Flash CS5.5\AIR2.6\lib\adt.jar"`.

> The `Program Files (x86)` folder will simply be named `Program Files` on 32-bit versions of Microsoft Windows.
>
> If you are using the latest AIR SDK, then you might expect to change the AIR version number listed in ADT's file path from `"Adobe Flash CS5.5\AIR2.6\"` to, for example, `"Adobe Flash CS5.5\AIR3.0\"`.
>
> However, there is no need to do this as the overlay process for the AIR SDK preserves the original folder locations. The latest version of the SDK actually sits in the 2.6 SDK's original location.
>
> For more detail, refer to the *Installing the AIR SDK* recipe from *Chapter 2*.

You will also need to select one of the following deployment types for your app:

- `ipa-test`: Quick publication for device testing
- `ipa-debug`: Quick publication for device debugging
- `ipa-ad-hoc`: Ad hoc
- `ipa-app-store`: Apple App Store deployment

For this recipe, substitute `[DEPLOYMENT_TYPE]` with `ipa-test`.

The location of your provisioning profile and P12 certificate file are also required. Replace `[PROV_FILE]` with `"<documents_folder>\packt\flash-ios-cookbook\developer-files\Flash_iOS_Cookbook.mobileprovision"` and `[CERT_FILE]` with `"<documents_folder>\packt\flash-ios-cookbook\developer-files\ios_dev.p12"`. Also substitute `[PASSWORD]` with the P12 certificate's password.

The name of the IPA you wish to compile is represented by `[OUTPUT_IPA]`. For this recipe use `bubbles.ipa`.

Also, point the command line compiler to your application descriptor file, which is represented by `[APP_DESC_FILE]`. This is an XML file that can either be hand-written or generated by Flash Professional. Flash will generate an application descriptor file when you set or edit an FLA's AIR for iOS settings. Therefore, one named `bubbles-app.xml` will already be available for this recipe and can be found within your FLA's root folder.

Finally, replace `[SRC_FILES]` with the name of the SWF you published at the start of this recipe.

6. Okay, with each of the capitalized blocks replaced, your command line should look something like this:

```
"C:\Program Files (x86)\Java\jre6\bin\java" -jar "C:\Program Files
(x86)\Adobe\Adobe Flash CS5.5\AIR2.6\lib\adt.jar" -package -target
ipa-test -provisioning-profile "<documents_folder>\packt\flash-
ios-cookbook\developer-files\Flash_iOS_Cookbook.mobileprovision"
-storetype pkcs12 -keystore "<documents_folder>\packt\flash-ios-
cookbook\developer-files\ios_dev.p12" -storepass <p12_password>
bubbles.ipa bubbles-app.xml bubbles.swf
```

Remember to replace `<documents_folder>` with the path to your own personal `Documents` folder and also `<p12_password>` with the password you associated with your P12 certificate.

7. Now press *Enter* to begin compilation. If successful, a file named `bubbles.ipa` will be generated and written to your FLA's root folder. This file can be deployed to your iOS device for testing. Compilation can take some time, particularly for large projects.

How it works...

The command line compiler is the same tool that Flash Professional uses to compile an `.ipa` file. Flash Professional CS5.5 uses the ADT, which comes with AIR 2.6 and above, while Flash Professional CS5 uses the PFI, which was made available as part of the AIR 2.0 SDK.

Each compiles an IPA from the following files:

- A P12 certificate file
- A provisioning profile associated with your app
- An application descriptor file
- A SWF file published from your FLA

After opening a command prompt window, it is convenient to change to the FLA's root directory. Doing so allows you to specify the SWF and application descriptor file without having to type their full file paths.

There's more...

This recipe has covered the mandatory command line options. Let us now explore a few more.

Including other source files

In addition to your `.swf` file, when creating a build from the command line, you must specify all other source files to be packaged with the resultant IPA. This includes application launch images, icons, and any files that are to be loaded from the device's file system at runtime.

To do this, simply include the relative path to each file or folder immediately after your SWF. Here is an example:

```
... bubbles.swf Default.png AppIconsForPublish\icon29.png
AppIconsForPublish\icon57.png AppIconsForPublish\icon512.png
resources\video resources\bitmaps
```

This snippet states that a single launch image (`Default.png`), three icons (`icon29.png`, `icon57.png`, and `icon512.png`), a folder containing video (`resources\video`), and a folder containing bitmaps (`resources\bitmaps`) are to be packaged along with the application's SWF.

Debug options

When creating a debug build from the command line, it is possible to specify the IP address of the development computer that is running the remote debugger. Simply include the `-connect` option along with an IP address. For example:

```
-target ipa-debug -connect 192.168.0.4
```

When launched, your app will immediately attempt to connect to the specified IP address rather than its default.

When using the `-connect` option, remember to create a debug build by first setting the `-target` option to `ipa-debug`.

Interpreter Mode

Since AIR 2.7, it has been possible to dramatically speed up compile times by creating an Interpreter Mode build. Such builds can only be created from the command line and are covered in the *Enabling Interpreter Mode* recipe in this chapter.

Creating a batch file

The ADT/PFI command is lengthy and ideally you won't want to repeatedly enter it into the command line each time you need to make a build. Consider creating a batch file that stores your compiler command.

Using a text editor such as Notepad, enter the following:

```
set JAVA="C:\Program Files (x86)\Java\jre6\bin\java"
set ADT="C:\Program Files (x86)\Adobe\Adobe Flash
CS5.5\AIR2.6\lib\adt.jar"
set TARGET=ipa-test
set PROV_FILE="<documents_folder>\packt\flash-ios-
cookbook\developer-files\Flash_iOS_Cookbook.mobileprovision"
set CERT_FILE="<documents_folder>\packt\flash-ios-
cookbook\developer-files\ios_dev.p12"
set PASSWORD=<p12_password>
set OUTPUT_IPA=bubbles.ipa
set APP_DESC_FILE=bubbles-app.xml
set SCR_FILES=bubbles.swf

%JAVA% -jar %ADT% -package -target %TARGET%
-provisioning-profile %PROV_FILE% -storetype pkcs12
-keystore %CERT_FILE%-storepass %PASSWORD%
%OUTPUT_IPA% %APP_DESC_FILE% %SRC_FILES%
```

Depending on the version of Flash Professional and Microsoft Windows you use, you may need to adjust some of the file paths within this batch file.

Also, remember to replace `<documents_folder>` with the path to your own personal `Documents` folder and also `<p12_password>` with the password you associated with your P12 certificate.

Save the file as `build.bat` to the FLA's root folder.

Now you can initiate a command line build by simply double-clicking on the `.bat` file from an Explorer window.

You can re-use this batch file with your own projects by simply changing the value of the appropriate variables declared at the top.

See also

- *Creating a P12 certificate using Windows*, *Chapter 1*
- *Creating a development provisioning profile*, *Chapter 1*
- *Installing the AIR SDK*, *Chapter 2*
- *Compiling from Flash Professional*, *Chapter 2*
- *Editing the application descriptor file*, *Chapter 3*
- *Enabling Interpreter Mode*
- *Running ADT outside your Flash Professional CS5 installation*

Compiling from the command line using Mac OS X

Throughout the course of this book, we have concentrated exclusively on the compilation of `.ipa` files directly from the Flash IDE. However, it is also possible to compile from the command line. Let us see how this is done.

The steps covered here are for Mac OS X. If you are using Microsoft Windows, then refer to the *Compiling from the command line using Windows* recipe.

Getting ready

The command line compiler is a Java application and requires the Java runtime environment in order to run. With the release of Lion, Apple no longer automatically installs Java with OS X. If you need to install the Java runtime or confirm that it is installed, then follow the instructions on the following web page provided by Adobe: `http://kb2.adobe.com/cps/909/cpsid_90908.html`.

An FLA has been provided for this recipe and should be copied to your `Documents` folder. To do this, navigate to your `Documents` folder using Finder and create the following folder structure within it: `packt/flash-ios-cookbook/chapter14/`. Now from the book's accompanying code bundle, copy `chapter14/recipe2/` to `Documents/packt/flash-ios-cookbook/chapter14/`.

Incidentally, the FLA you will be working from is a version of the Bubbles app from *Chapter 3*.

How to do it...

A `.swf` file is required when compiling a native iOS app from the command line. Let us begin by publishing a SWF from our FLA:

1. Launch Flash Professional and open `Documents/chapter14/recipe2/bubbles.fla`.
2. Generate a SWF by selecting **Control | Test Movie | in AIR Debug Launcher (Mobile)**. The SWF will output to the same folder as the FLA and ADL will launch allowing you to preview it. Close the preview window once you are satisfied that publication was successful.
3. Launch the Terminal application from the `Applications/Utilities` folder.
4. From the terminal window, move to the folder where your `.swf` file is located by entering the following:

   ```
   cd ~/Documents/packt/flash-ios-cookbook/chapter14/recipe2
   ```

5. Okay, now we are ready to compile an IPA from the command line. Let us take a look at the general format of the command you will be entering:

   ```
   [ADT_FILE] -package
              -target [DEPLOYMENT_TYPE]
              -provisioning-profile [PROV_FILE]
              -storetype pkcs12 -keystore [CERT_FILE]
              -storepass [PASSWORD]
              [OUTPUT_IPA]
              [APP_DESC_FILE]
              [SRC_FILES]
   ```

   Each of the capitalized blocks will need to be replaced with the appropriate file path or option. Here is what each block represents:

   - `[ADT_FILE]`: Either the AIR Development Tool (ADT) or the Packager for iPhone (PFI)
   - `[DEPLOYMENT_TYPE]`: The deployment type to be used
   - `[PROV_FILE]`: The provisioning profile associated with your app
   - `[CERT_FILE]`: The P12 certificate file required to digitally sign your app
   - `[PASSWORD]`: The password you associated with your P12 certificate
   - `[OUTPUT_IPA]`: The name of the IPA file to be compiled
   - `[APP_DESC_FILE]`: Your FLA's application descriptor file
   - `[SRC_FILES]`: The SWF file published from your FLA followed by any other assets to be included within the IPA.

Let us work through each of them, one at a time.

Replace `[ADT_FILE]` with either the location of the AIR Development Tool (ADT) or the Packager for iPhone (PFI) depending on the version of AIR you are using. If you are using Flash Professional CS5, then the file path to the PFI is: `"/Applications/Adobe Flash CS5/PFI/bin/pfi"`. For CS5.5, the file path is: `"/Applications/Adobe Flash CS5.5/AIR2.6/bin/adt"`.

> If you are using the latest AIR SDK, then you might expect to change the AIR version number listed in ADT's file path from `"Adobe Flash CS5.5/AIR2.6/"` to, for example, `"Adobe Flash CS5.5/AIR3.0/"`.
>
> However, there is no need to do this as the overlay process for the AIR SDK preserves the original folder locations. The latest version of the SDK actually sits in the 2.6 SDK's original location.
>
> For more detail, refer to the *Installing the AIR SDK* recipe from *Chapter 2*.

You will also need to select one of the following deployment types for your app:

- `ipa-test`: Quick publication for device testing
- `ipa-debug`: Quick publication for device debugging
- `ipa-ad-hoc`: Ad hoc
- `ipa-app-store`: Apple App Store deployment

For this recipe, substitute `[DEPLOYMENT_TYPE]` with `ipa-test`.

The location of your provisioning profile and P12 certificate file are also required. Replace `[PROV_FILE]` with `~/Documents/packt/flash-ios-cookbook/developer-files/Flash_iOS_Cookbook.mobileprovision` and `[CERT_FILE]` with `~/Documents/packt/flash-ios-cookbook/developer-files/<p12_filename>`. Also substitute `[PASSWORD]` with the P12 certificate's password.

> In the previous paragraph, and throughout this recipe, replace `<p12_filename>` with the name of your own P12 certificate. This will be the file name you saved your certificate as during the *Creating a P12 certificate using Mac OS X* recipe from *Chapter 1*.

The name of the IPA you wish to compile is represented by `[OUTPUT_IPA]`. For this recipe use `bubbles.ipa`.

Also, point the command line compiler to your application descriptor file, which is represented by `[APP_DESC_FILE]`. This is an XML file that is generated by Flash when you set or edit a FLA's AIR for iOS settings. One named `bubbles-app.xml` will already be available for this recipe and can be found within your FLA's root folder.

Finally, replace `[SRC_FILES]` with the name of the SWF you published at the start of this recipe.

Okay, with each of the capitalized blocks replaced, your command line should look something like this:

```
"/Applications/Adobe Flash CS5.5/AIR2.6/bin/adt" -package
-target ipa-test -provisioning-profile ~/Documents/packt/flash-
ios-cookbook/developer-files/Flash_iOS_Cookbook.mobileprovision
-storetype pkcs12 -keystore ~/Documents/packt/flash-ios-
cookbook/developer-files/<p12_filename> -storepass
 <p12_password> bubbles.ipa bubbles-app.xml bubbles.swf
```

In the line above, replace `<p12_filename>` with the name (including `.p12` file extension) of your P12 certificate and `<p12_password>` with the password you associated with it.

6. Now press *Enter* to begin compilation. If successful, a file named `bubbles.ipa` will be generated and written to your FLA's root folder. This file can be deployed to your iOS device for testing. Compilation can take some time, particularly for large projects.

How it works...

The command line compiler is the same tool that Flash Professional uses to compile an `.ipa` file. Flash Professional CS5.5 uses the ADT, which comes with AIR 2.6 and above, while Flash Professional CS5 uses the PFI, which was made available as part of the AIR 2.0 SDK.

Each compiles an IPA from the following files:

- A P12 certificate file
- A provisioning profile associated with your app
- An application descriptor file
- A SWF file published from your FLA

After opening a terminal window, it is convenient to change to the FLA's root directory. Doing so allows you to specify the SWF and application descriptor file without having to type their full file paths.

There's more...

This recipe has covered the mandatory command line options. Let us now explore a few more.

Including other source files

In addition to your `.swf` file, when creating a build from the command line, you must specify all other source files to be packaged with the resultant IPA file. This includes application launch images, icons, and any files that are to be loaded from the device's file system at runtime.

To do this, simply include the relative path to each file or folder immediately after your SWF. Here is an example:

```
... bubbles.swf Default.png AppIconsForPublish/icon29.png
AppIconsForPublish/icon57.png AppIconsForPublish/icon512.png
resources/video resources/bitmaps
```

This snippet states that a single launch image (`Default.png`), three icons (`icon29.png`, `icon57.png`, and `icon512.png`), a folder containing video (`resources/video`), and a folder containing bitmaps (`resources/bitmaps`) are to be packaged along with the application's SWF.

Debug options

When creating a debug build from the command line, it is possible to specify the IP address of the development computer that is running the remote debugger. Simply include the `-connect` option along with an IP address. For example:

```
-target ipa-debug -connect 192.168.0.4
```

When launched, your app will immediately attempt to connect to the specified IP address rather than its default.

When using the `-connect` option, remember to create a debug build by first setting the `-target` option to `ipa-debug`.

Interpreter Mode

Since AIR 2.7, it has been possible to dramatically speed-up compile times by creating an Interpreter Mode build. Such builds can only be created from the command line and are covered in the *Enabling Interpreter Mode* recipe in this chapter.

Creating a shell script

The ADT/PFI command is lengthy and ideally you won't want to repeatedly enter it into the command line each time you need to make a build. Consider creating a shell script that you can use to store and execute your compiler command.

Let us create a script named `build.sh` in your FLA's root folder. We will use the Nano text editor.

Launch Nano from the command line with:

```
nano build.sh
```

Nano runs within the terminal window. Enter the following script:

```
#!/bin/sh
ADT="/Applications/Adobe Flash CS5.5/AIR2.6/bin/adt"
TARGET=ipa-test
PROV_FILE=~/Documents/packt/flash-ios-cookbook/
developer-files/Flash_iOS_Cookbook.mobileprovision
CERT_FILE=~/Documents/packt/flash-ios-cookbook/
developer-files/<p12_filename>
PASSWORD=<p12_password>
OUTPUT_IPA=bubbles.ipa
APP_DESC_FILE=bubbles-app.xml
SRC_FILES=bubbles.swf

"$ADT" -package -target $TARGET -provisioning-profile
$PROV_FILE -storetype pkcs12 -keystore $CERT_FILE -storepass
$PASSWORD $OUTPUT_IPA $APP_DESC_FILE $SCR_FILES
```

Remember to replace `<p12_filename>` with the name (including `.p12` file extension) of your P12 certificate and `<p12_password>` with the password you associated with it.

Also, if you are using Flash Professional CS5, then change the value of the `ADT` variable to the Packager for iPhone's file path: `"/Applications/Adobe Flash CS5/PFI/bin/pfi"`.

Press *Ctrl* + *X* to exit Nano. Before Nano is closed, you will be asked if you would like to save the modified buffer. Press the *Y* key then *Enter* to save the file using its default name. The script will be saved and Nano will close.

Finally, make the script executable by entering the following into the command line:

```
chmod +x build.sh
```

Now whenever you want to make a command line build, you can by typing:

```
./build.sh
```

This batch file can be re-used with your own projects by simply changing the value of the appropriate variables declared at the top.

See also

- *Creating a P12 certificate using Mac OS X, Chapter 1*
- *Creating a development provisioning profile, Chapter 1*
- *Installing the AIR SDK, Chapter 2*
- *Compiling from Flash Professional, Chapter 2*
- *Editing the application descriptor file, Chapter 3*
- *Enabling Interpreter Mode*
- *Running ADT outside your Flash Professional CS5 installation*

Using a native extension

In an attempt to ensure that AIR's reach extends as far as possible, Adobe is careful not to introduce APIs that are specific to a single platform. Theoretically, any APIs you use for iOS should also work on other platforms such as Windows, Mac OS X, and Android. Therefore, a feature of iOS is, typically, only directly supported if it can be made available across all platforms supported by the AIR SDK.

However, this doesn't necessarily mean that you can't take advantage of platform-specific features within your applications. Since the release of AIR 3.0, developers have been able to do just that through the use of custom ActionScript libraries that are implemented with native code. These are known as native extensions and allow AIR for iOS apps to take advantage of the same platform and device-specific capabilities and APIs available to native applications.

As an example, let us take a look at Adobe's Gyroscope native extension and see how you would use its API within your project's ActionScript.

Getting ready

Native extensions are only available from Adobe AIR 3.0 onwards. Ensure you have overlaid the latest AIR SDK onto Flash Professional CS5.5. If you haven't, then follow the steps detailed in the *Installing the AIR SDK* recipe from *Chapter 2*.

You will also need to download the Gyroscope native extension, but first we will create a folder for it and this recipe's example code. Using either Windows Explorer or Finder, create the following folder structure within your `Documents` folder: `packt\flash-ios-cookbook\chapter14\recipe3\`.

Now within a web browser, visit the Adobe AIR Developer Center at: `www.adobe.com/devnet/air/native-extensions-for-air.html`. Here you will find a list of native extensions that are currently available. Select the **Gyroscope** extension and download `Gyroscope.zip` to your `recipe3` folder from the **Gyroscope native extension sample** page. Extract the `.zip` file to the same folder.

Finally, an FLA has been provided as a starting point for this recipe. From the book's accompanying code bundle, copy the contents of `chapter14\recipe3` to `packt\flash-ios-cookbook\chapter14\recipe3\` within your `Documents` folder.

Sitting on the FLA's stage are three dynamic text fields named `gyroXField`, `gyroYField`, and `gyroZField`. We will use them to display data coming from the device's gyroscope.

How to do it...

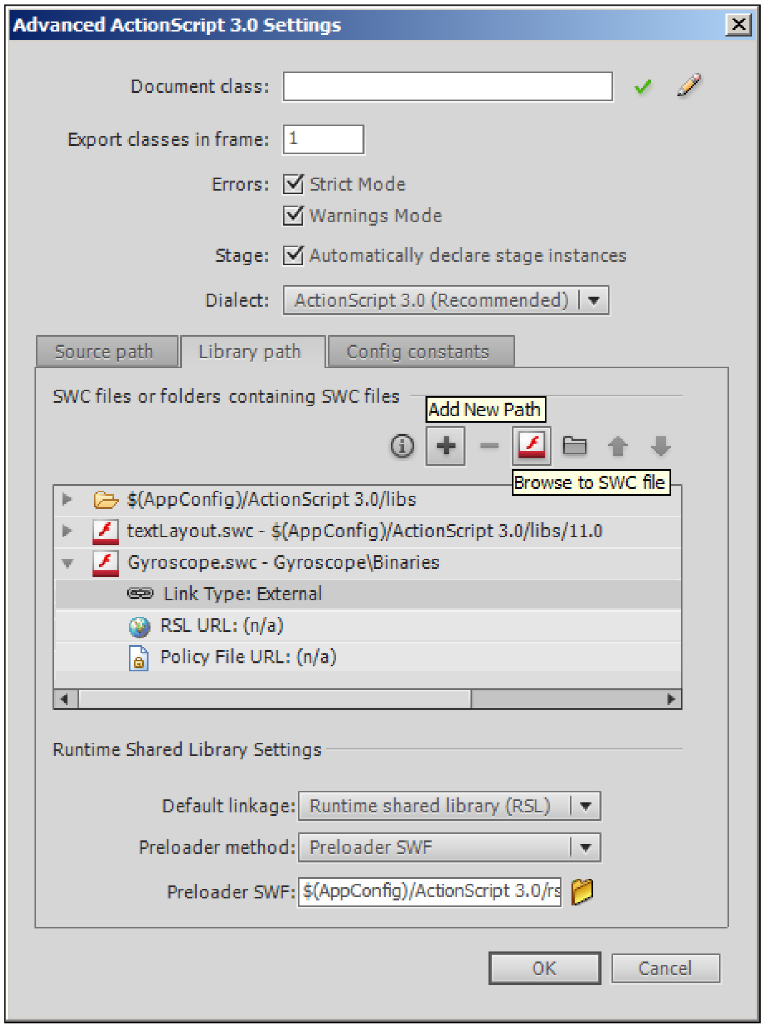
Every native extension comes with a `.swc` file, which defines the ActionScript classes that the extension provides. Before we can use these classes, the `.swc` file needs to be linked to our FLA:

Linking the SWC

Perform the following steps to link the extension's SWC file:

1. From your `Documents` folder, open `chapter14\recipe3\recipe.fla` into Flash Professional CS5.5.
2. Select **File | ActionScript Settings** from Flash Professional's drop-down menu.
3. From the **Advanced ActionScript 3.0 Settings** panel, click on the **Library path** tab.

4. Click on the **Add New Path** icon, which is represented by a **+** symbol, then click on the **Browse to SWC file** icon as shown in the following screenshot:



Advanced ActionScript 3.0 Settings

Document class:

Export classes in frame: 1

Errors: ☑ Strict Mode

☑ Warnings Mode

Stage: ☑ Automatically declare stage instances

Dialect: ActionScript 3.0 (Recommended)

Source path | Library path | Config constants

SWC files or folders containing SWC files

Add New Path

Browse to SWC file

$(AppConfig)/ActionScript 3.0/libs

textLayout.swc - $(AppConfig)/ActionScript 3.0/libs/11.0

Gyroscope.swc - Gyroscope\Binaries

Link Type: External

RSL URL: (n/a)

Policy File URL: (n/a)

Runtime Shared Library Settings

Default linkage: Runtime shared library (RSL)

Preloader method: Preloader SWF

Preloader SWF: $(AppConfig)/ActionScript 3.0/rs

OK | Cancel

5. From your FLA's root folder, browse to, and select `Gyroscope\Binaries\Gyroscope.swc`. It will be added to a list of SWC files to be used by your app.
6. Expand the `Gyroscope.swc` entry by clicking on the icon to the left of its path. Double-click on the **Link Type** entry and change it from **Merged into code** to **External**.
7. Close the **Advanced ActionScript 3.0 Settings** panel by clicking on the **OK** button at the bottom.
8. Save your FLA.

Using the extension's ActionScript API

Now let us write code that will access the device's gyroscope through the native extension's ActionScript API.

1. Create a document class and name it `Main`.
2. Import the following classes belonging to the native extension:

   ```
   import flash.display.MovieClip;
   import com.adobe.nativeExtensions.Gyroscope;
   import com.adobe.nativeExtensions.GyroscopeEvent;
   ```

3. Also add a member variable of type `Gyroscope`:

   ```
   private var gyro:Gyroscope;
   ```

4. Within the constructor, create an instance of the extension's `Gyroscope` class and listen for updates from it:

   ```
   public function Main() {
     if(Gyroscope.isSupported)
     {
       gyro = new Gyroscope();
       gyro.setRequestedUpdateInterval(50);
       gyro.addEventListener(GyroscopeEvent.UPDATE, updated);
     }
   }
   ```

5. Add a method that displays data from the gyroscope whenever it is updated:

   ```
   private function updated(e:GyroscopeEvent):void {
     gyroXField.text = String(e.x);
     gyroYField.text = String(e.y);
     gyroZField.text = String(e.z);
   }
   ```

6. Save the class as `Main.as`. Also save your FLA.
7. Generate a SWF by selecting **Control** | **Test Movie** | **in AIR Debug Launcher (Mobile)** from Flash Professional's drop-down menu.

   Don't create the SWF by selecting **File** | **Publish** (*Alt* + *Shift* + *F12* | *Shift* + *Cmd* + *F12*). Doing so will also attempt to compile the SWF into an IPA, which we aren't ready to do just yet.

   A SWF will be created but you will receive the following runtime error from ADL:

   VerifyError: Error #1014: Class com.adobe.nativeExtensions::Gyroscope could not be found.

   Ignore this error for the time being.

You now have your very first SWF that makes use of an ActionScript API provided by a native extension. We are not quite in a position to create an `.ipa` file yet but we will address that in the next recipe.

How it works...

A native extension is a combination of native code and one or more ActionScript classes. As an AIR for iOS developer, you will only ever work with the extension's ActionScript API, which calls the native code on your behalf.

Included with every native extension are the following two files:

- A SWC file containing the extension's ActionScript API
- An ANE file containing the native code and resources used by the extension

This recipe only made use of the SWC file, which was used to provide our FLA's document class with access to the Gyroscope API.

While many SWC's are statically linked to an FLA, a native extension's SWC should be set as an external library file. This allows its class definitions to be used during development while preventing the SWC itself from being added to the app's SWF when published.

This may seem odd at first, but it is important to understand that a native extension's SWC does not contain any implementation for its API. Instead that is taken care of by the native code within the ANE file. The SWC is only required during the development of your ActionScript whereas the ANE is required during compilation of the final `.ipa` file.

A native extension can support multiple platforms; however, this is not a requirement. This has implications as it means there is no guarantee that you will be able to test your SWF using ADL on your development computer. This is the case with the Gyroscope extension, which only provides iOS and Android support. It is also the reason why an error was thrown at runtime within ADL.

There's more...

Here is some additional information regarding ActionScript native extensions.

Conditional compilation

Consider using conditional compilation when working with native extensions that do not provide an implementation for your development computer. This will prevent errors being thrown from ADL and allow you to test your code to some degree without having to run it on a device.

Conditional compilation allows you to turn blocks of code on or off depending on the value of configuration constants that you define within the ActionScript Settings panel. For example, you could define constants to determine whether the code for the Gyroscope native extension should be included with the published SWF or a mock version of the code for testing purposes on desktop.

More information can be found by performing a search for `conditional compilation` within Adobe Community Help.

Other available native extensions

The Adobe AIR Developer Center has many example extensions for download including some written by the AIR community. There is also documentation and tutorials detailing how to write your own extensions. Take a look at: `www.adobe.com/devnet/air/native-extensions-for-air.html`.

There are many other places where you can download native extensions, such as the Native Extensions for AIR website, which has an ever-increasing catalogue of iOS and Android extensions: `http://extensionsforair.com`.

Native extensions for some of the more popular iOS APIs have also already been written. For example, an In-App Purchasing extension can be downloaded from `http://code.google.com/p/in-app-purchase-air-ios` and support for iAds has also been made available at `http://code.google.com/p/iad-air-ios`.

While many of these extensions can be downloaded free of charge, some are available for a small fee. For example, a Game Center native extension can be purchased from `www.milkmangames.com/blog/tools`.

The list of native extensions will continue to grow over time.

See also

- *Packaging a native extension*

Packaging a native extension

Flash Professional CS5.5 does not directly support the packaging of applications that use native extensions. Instead, you will need to use the AIR Development Tool (ADT) from the command line.

Let us walk through the steps required to package the previous recipe's Gyroscope example into a native iOS app.

Getting ready

Before attempting this recipe, you will need to have published a SWF file from the *Using a native extension* recipe's example FLA. Familiarity with the command line and ADT is also a prerequisite. If you haven't already done so, complete the following recipes:

- *Packaging from the command line* (*using Windows* or *Mac OS X*)
- *Using a native extension*

Continue to work from the code you wrote during the *Using a native extension* recipe.

How to do it...

To package a native extension with your app, we will split the recipe into the following three tasks:

1. Obtaining the extension's unique ID.
2. Declaring the extension within your application descriptor file.
3. Compiling your app using ADT.

Obtaining the extension's unique ID

Every native extension you wish to use must be declared within your application descriptor file. First you will need to obtain a unique ID associated with each extension. Perform these steps to obtain the Gyroscope extension's ID:

1. Using Windows Explorer or Finder, browse to `Gyroscope\Binaries\`.
2. Inside this folder, you will find a file named `extID.txt`. It contains the Gyroscope extension's unique ID. Open the file within a text editor and copy `com.adobe.gyroscope` to the clipboard.
3. Close the file.

Declaring the extension within your application descriptor file

We will add an entry for the Gyroscope extension within your application descriptor file:

1. If you haven't already done so, open Flash Professional CS5.5.
2. Select **File** | **Open** (*Ctrl* + *O* | *Cmd* + *O*) from Flash Professional's drop-down menu.
3. When prompted for a file, select `recipe-app.xml` from the FLA's root folder and click on **Open**.

4. Near the end of the application descriptor file, add an `<extensions>` element. Paste the Gyroscope extension's ID from the clipboard into an `<extensionID>` child element:

   ```
   </iPhone>

   <extensions>
     <extensionID>com.adobe.gyroscope</extensionID>
   </extensions>

   </application>
   ```

5. Save your application descriptor file and close it.

Compiling your app using ADT

With the extension declared, you can package it with your application:

> If you haven't already done so, open `recipe.fla` and generate a SWF by selecting **Control | Test Movie | in AIR Debug Launcher (mobile)**. The SWF will be output to the FLA's root folder.

1. Depending on your choice of operating system, open either a command prompt or terminal window, and move to your FLA's root folder. For Windows, do this by entering the following command:

   ```
   cd <documents_folder>\packt\flash-ios-cookbook\chapter14\recipe3
   ```

   > Replace `<documents_folder>` with the path to your own personal `Documents` folder.

   If you are using Mac OS X, then enter the following into your terminal window instead:

   ```
   cd ~/Documents/packt/flash-ios-cookbook/chapter14/recipe3
   ```

2. To package a native extension with your app, you will need to use the `-extdir` command line option to specify the directory where your extension's ANE file resides. Here is the general form of the ADT command with the `-extdir` option highlighted:

   ```
   adt -package
       -target [DEPLOYMENT_TYPE]
       -provisioning-profile [PROV_FILE]
       -storetype pkcs12 -keystore [CERT_FILE]
       -storepass [PASSWORD]
   ```

```
[OUTPUT_IPA]
[APP_DESC_FILE]
[SRC_FILES]
-extdir [EXTENSIONS_DIR]
```

Let us go ahead and create a native iOS application using the `-extdir` option. Here is how the command line will look for Windows 7 and Vista:

```
"C:\Program Files (x86)\Java\jre6\bin\java" -jar "C:\Program
Files (x86)\Adobe\Adobe Flash CS5.5\AIR2.6\lib\adt.jar" -package
-target ipa-test -provisioning-profile "<documents_folder>\
packt\flash-ios-cookbook\developer-files\
Flash_iOS_Cookbook.mobileprovision" -storetype pkcs12
-keystore "<documents_folder>\packt\
flash-ios-cookbook\developer-files\ios_dev.p12"
-storepass <p12_password> c14-r3.ipa recipe-app.xml recipe.swf
-extdir "Gyroscope\Binaries"
```

Replace `<documents_folder>` with the path to your own personal `Documents` folder and also `<p12_password>` with the password you associated with your P12 certificate.

For Mac OS X you should type:

```
"/Applications/Adobe Flash CS5.5/AIR2.6/bin/adt" -package
-target ipa-test -provisioning-profile ~/Documents/
packt/flash-ios-cookbook/developer-files/
Flash_iOS_Cookbook.mobileprovision
-storetype pkcs12 -keystore ~/Documents/packt/
flash-ios-cookbook/developer-files/<p12_filename>
-storepass <p12_password> c14-r3.ipa recipe-app.xml recipe.swf
-extdir "Gyroscope\Binaries"
```

Replace `<p12_filename>` with the name (including `.p12` file extension) of your P12 certificate and `<p12_password>` with the password you associated with it.

3. Now press *Enter* to begin compilation. If successful, a file named `c14-r3.ipa` will be created that has native integration with the gyroscope found in many iOS devices.
4. Deploy the IPA. For testing, you will need a device featuring a gyroscope and iOS 4 or above.

When launched, you should see the data from your device's gyroscope being displayed within the app's three dynamic text fields.

How it works...

ADT is the same tool that Flash Professional CS5.5 uses to compile and package SWF files into native iOS applications. However, at present, Flash Professional does not provide support for ADT's `-extdir` option meaning you will need to package your app from the command line.

The `-extdir` option is used to point to the folder where the ANE files for any native extensions required by your application live. Only the Gyroscope extension was used for this recipe, which is why we opted to simply set `-extdir` to point to the extension's own `Binaries` folder. However, if you are working with multiple extensions, then copy all the ANE files into a single location.

> Be careful **not** to use a trailing slash when specifying the path for the `-extdir` option. Doing so will result in an error. For example, `Gyroscope\Binaries` is valid, whereas `Gyroscope\Binaries\` is not.

It is also important that you declare each extension within your application descriptor file. This is done by specifying the extension's unique ID. Typically, the ID is stored within a text file in the same folder as the extension's `.swc` and `.ane` files. Simply copy and paste each extension ID into an `<extensions>` block within your application descriptor file. This is shown in the following example:

```
<extensions>
  <extensionID>com.adobe.gyroscope</extensionID>
  <extensionID>com.adobe.Vibration</extensionID>
  <extensionID>com.adobe.appPurchase</extensionID>
</extensions>
```

Once you have edited the application descriptor file and published a SWF from your FLA, you will be able to call ADT from the command line to compile and package your app.

There's more...

Here is one final piece of information regarding native extensions.

Publishing from future versions of Flash Professional

Flash Professional CS5.5 was released before AIR 3.0 SDK, explaining why support for native extension packaging isn't currently provided. However, it is likely that the next major release of Flash Professional will include full support for native extensions.

See also

- *Using a native extension*
- *Compiling from the command line using Windows*
- *Compiling from the command line using Mac OS X*

Enabling Interpreter Mode

Compilation to a native iOS app can take several minutes depending on the complexity of your Flash project and the hardware specification of your computer. Such lengthy times can affect your workflow and impact productivity. In an attempt to streamline the iterative testing and debugging workflow, AIR 2.7 introduced **Interpreter Mode**; a new build target which reduces compile times to no more than a couple of seconds.

Flash Professional CS5.5 does not directly support the creation of Interpreter Mode builds. Instead, you will need to use the AIR Development Tool (ADT) from the command line.

Let us see how this is done.

Getting ready

Interpreter Mode builds are only available from Adobe AIR 2.7 onwards. If you are using Flash Professional CS5.5, then make sure you have overlaid the latest AIR SDK. If you haven't already done so, follow the steps detailed in the *Installing the AIR SDK* recipe from *Chapter 2*.

Additionally, you will need to know how to compile your Flash projects from the command line. If you are not familiar with command line compilation, then complete one of the following two recipes from this chapter before continuing:

- *Compiling from the command line using Windows*
- *Compiling from the command line using Mac OS X*

If you are using Flash Professional CS5 and would like to make an Interpreter Mode build, then first refer to the *Running ADT outside your Flash Professional CS5 installation* recipe at the end of this chapter.

An FLA has been provided as a starting point for this recipe and should be copied to your `Documents` folder. To do this, navigate to your `Documents` folder and create the following folder structure within it: `packt\flash-ios-cookbook\chapter14\`. Now copy `chapter14\recipe5\` from the book's accompanying code bundle to `packt\flash-ios-cookbook\chapter14\` within your `Documents` folder.

If you are using Microsoft Windows, then the location of your `Documents` folder depends on the version of Windows you are running. For Windows Vista and Windows 7, it can be found at `C:\Users\<username>\Documents\`. If you are using Windows XP, then it is at `C:\Documents and Settings\<username>\My Documents\`.

Incidentally, the FLA you will be working from is a version of the Bubbles app from *Chapter 3*.

How to do it...

A `.swf` file is required when compiling a native iOS app from the command line. We will begin by publishing a SWF from our FLA:

1. Launch Flash Professional and open `chapter14\recipe5\bubbles.fla` from your `Documents` folder.
2. Generate a SWF by selecting **Control | Test Movie | in AIR Debug Launcher (Mobile)**. The SWF will output to the same folder as the FLA and ADL will launch allowing you to preview it. Close the preview window once you are satisfied that publication was successful.

   Don't create the SWF by selecting **File | Publish** (*Alt* + *Shift* + *F12* | *Shift* + *Cmd* + *F12*). Doing so will also attempt to compile the SWF into an IPA from within Flash Professional, which we are trying to avoid.

3. Depending on your choice of operating system, open either a command prompt or terminal window, and move to the folder where the `.swf` file is located. For Windows, do this by entering the following command:

   ```
   cd <documents_folder>\packt\flash-ios-cookbook\chapter14\recipe5
   ```

   Replace `<documents_folder>` with the path to your own personal `Documents` folder.

   If you are using Mac OS X, then enter the following into your terminal window instead:

   ```
   cd ~/Documents/packt/flash-ios-cookbook/chapter14/recipe5
   ```

4. To initiate an Interpreter Mode build, a new deployment type needs to be specified on the command line. You can select from one of the following:
   - `ipa-test-interpreter`: Interpreter Mode publication for testing
   - `ipa-debug-interpreter`: Interpreter Mode publication for device debugging

Here is the general form of the ADT command with both deployment types listed:

```
adt -package
    -target [ipa-test-intepreter|ipa-debug-intepreter]
    -provisioning-profile [PROV_FILE]
    -storetype pkcs12 -keystore [CERT_FILE]
    -storepass [PASSWORD]
    [OUTPUT_IPA]
    [APP_DESC_FILE]
    [SRC_FILES]
```

Let us go ahead and make an Interpreter Mode build using the `ipa-test-interpreter` target option. Here is how the command line will look for Windows 7 and Vista:

```
"C:\Program Files (x86)\Java\jre6\bin\java" -jar "C:\Program
Files (x86)\Adobe\Adobe Flash CS5.5\AIR2.6\lib\adt.jar"
-package -target ipa-test-interpreter -provisioning-profile
"<documents_folder>\packt\flash-ios-cookbook\
developer-files\Flash_iOS_Cookbook.mobileprovision"
-storetype pkcs12 -keystore "<documents_folder>\packt\
flash-ios-cookbook\developer-files\ios_dev.p12" -storepass
<p12_password> bubbles.ipa bubbles-app.xml bubbles.swf
```

Replace `<documents_folder>` with the path to your own personal `Documents` folder and also `<p12_password>` with the password you associated with your P12 certificate.

If you are using Windows XP, then before executing the command, refer to the *Compiling from the command line using Windows* recipe for path changes that are required.

For Mac OS X, you should type:

```
"/Applications/Adobe Flash CS5.5/AIR2.6/bin/adt" -package
-target ipa-test-interpreter -provisioning-profile
~/Documents/packt/flash-ios-cookbook/developer-files
/Flash_iOS_Cookbook.mobileprovision-storetype pkcs12
-keystore ~/Documents/packt/
flash-ios-cookbook/developer-files/<p12_filename>
-storepass <p12_password> bubbles.ipa bubbles-app.xml
bubbles.swf
```

Replace `<p12_filename>` with the name (including `.p12` file extension) of your P12 certificate and `<p12_password>` with the password you associated with it.

5. Now press *Enter* to begin compilation. If successful, a file named `bubbles.ipa` will be generated almost instantaneously! Deploy the IPA to your device for testing.

How it works...

As you witnessed, compiling with the Interpreter Mode build target is extremely fast, but why? Rather than performing ahead-of-time compilation using LLVM, ADT simply creates a special IPA that has both your `.swf` file and an AIR runtime interpreter packaged within it. Generating such an IPA takes significantly less time than converting your `.swf` file to ARMv7 machine code.

Although this build target provides faster compilation, your application's code will execute slower on the device as it has to be interpreted. Nevertheless, for many stretches of your development cycle, Interpreter Mode builds will be extremely useful and save significant time.

However, when testing and debugging performance-critical areas of your application, you should still use either the standard `ipa-test` or `ipa-debug` targets as they provide a true reflection of your app's performance.

It should also be noted that Interpreter Mode builds should only be used during development. They cannot be used for distribution.

See also

- *Installing the AIR SDK, Chapter 2*
- *Compiling from the command line using Windows*
- *Compiling from the command line using Mac OS X*
- *Running ADT outside your Flash Professional CS5 installation*

Running ADT outside your Flash Professional CS5 installation

If you are publishing from Flash Professional CS5, then you will be limited to the AIR 2.0 SDK and its Packager for iPhone (PFI) command line tool.

While you can't overlay the most recent version of the AIR SDK into Flash Professional CS5, you can install it separately on your development computer. Doing so will allow you to package your SWF into an IPA using the latest version of the AIR Development Tool (ADT) instead. Although this won't grant you access to post AIR 2.0 APIs from Flash CS5, it will enable improved rendering performance, faster runtime code execution, and allow you to create Interpreter Mode builds.

In this recipe, we will use ADT to compile a SWF published from Flash Professional CS5 into an IPA.

Getting ready

If you are not familiar with command line compilation, then complete one of the following two recipes from this chapter before continuing:

- *Compiling from the command line using Windows*
- *Compiling from the command line using Mac OS X*

Additionally, an FLA and its application descriptor file have been provided and should be copied to your `Documents` folder. Using either Windows Explorer or Finder, navigate to your `Documents` folder and create the following sub-folders: `packt\flash-ios-cookbook\chapter14\`. Now copy `chapter14\recipe6\` from the book's accompanying code bundle to `packt\flash-ios-cookbook\chapter14\` within your `Documents` folder.

> If you are using Microsoft Windows, then the location of your `Documents` folder depends on the version of Windows you are running. For Windows Vista and Windows 7, it can be found at `C:\Users\<username>\Documents\`. If you are using Windows XP, then it is at `C:\Documents and Settings\<username>\My Documents\`.

How to do it...

This recipe will be split into the following two parts:

1. Installing the latest AIR SDK.
2. Compiling from the command line.

We will begin by installing the latest version of the AIR SDK.

Installing the latest AIR SDK

ADT is provided with the AIR SDK and can be downloaded from Adobe's website. Follow these steps to download and install it:

1. Launch your preferred web browser and visit the AIR SDK download page at `www.adobe.com/special/products/air/sdk`.
2. From the page, click on the **Download now** link for either Windows or Mac OS X. An archive file containing the AIR SDK will begin to download.
3. Once downloaded, extract the archive and copy it to a suitable location on your computer. If you are using Windows, then copy it to `C:\`. For Mac OS X, copy it to `/Users/<username>/Applications/`.

Compiling from the command line

With the SDK installed, let us use ADT to compile a SWF into an IPA. We will start by publishing the SWF file from Flash Professional CS5.

1. From your `Documents` folder, open `chapter14\recipe6\render-test-gpu.fla` into Flash Professional CS5.
2. Generate a SWF by selecting **Control** | **Test Movie** | **in AIR Debug Launcher (Mobile)**. The SWF will output to the same folder as the FLA and ADL will launch allowing you to preview it. Close the preview window once you are satisfied that publication was successful.

> **Don't** create the SWF by selecting **File** | **Publish** (*Alt* + *Shift* + *F12* | *Shift* + *Cmd* + *F12*). Doing so will also attempt to compile the SWF into an IPA from within Flash Professional, which we are trying to avoid.

3. Depending on your choice of operating system, open either a command prompt or terminal window, and move to the folder where the `.swf` file is located. For Windows, do this by entering the following command:

   ```
   cd <documents_folder>\packt\flash-ios-cookbook\chapter14\recipe6
   ```

   > Replace `<documents_folder>` with the path to your own personal `Documents` folder.

   If you are using Mac OS X, then enter the following into your terminal window instead:

   ```
   cd ~/Documents/packt/flash-ios-cookbook/chapter14/recipe6
   ```

4. Now that you have a `.swf` file, use ADT to initiate a build from the command line. Here is the general form of the ADT command:

   ```
   adt -package
       -target [DEPLOYMENT_TYPE]
       -provisioning-profile [PROV_FILE]
       -storetype pkcs12 -keystore [CERT_FILE]
       -storepass [PASSWORD]
       [OUTPUT_IPA]
       [APP_DESC_FILE]
       [SRC_FILES]
   ```

> Remember, for this recipe we will be running ADT from the AIR SDK folder that lives outside your Flash Professional CS5 installation. If you are using Windows, then you will be calling `C:\AdobeAIRSDK\lib\adt.jar`. For Mac OS X, you will call `/Users/<username>/Applications/AdobeAIRSDK/bin/adt`.

Let us go ahead and make a build using the `ipa-test` target option. Here is how the command line will look for Windows 7 and Vista:

```
"C:\Program Files (x86)\Java\jre6\bin\java"
-jar "C:\AdobeAIRSDK\lib\adt.jar" -package -target ipa-test
-provisioning-profile "<documents_folder>\packt\ flash-ios-
cookbook\developer-files\Flash_iOS_Cookbook.mobileprovision"
-storetype pkcs12 -keystore "<documents_folder>\packt\
flash-ios-cookbook\developer-files\ios_dev.p12" -storepass
<p12_password> render-test-gpu.ipa render-test-gpu-app.xml
render-test-gpu.swf
```

Replace `<documents_folder>` with the path to your own personal `Documents` folder and also `<p12_password>` with the password you associated with your P12 certificate.

If you are using Windows XP, then before executing the command, refer to the *Compiling from the command line using Windows* recipe for path changes that are required.

For Mac OS X, you should type:

```
"/Users/<username>/Applications/bin/adt" -package -target
ipa-test -provisioning-profile ~/Documents/packt/flash-ios-
cookbook/developer-files/Flash_iOS_Cookbook.mobileprovision
-storetype pkcs12 -keystore ~/Documents/packt/flash-ios-
cookbook/developer-files/<12_filename> -storepass <p12_password>
render-test-gpu.ipa render-test-gpu-app.xml render-test-gpu.swf
```

Replace `<p12_filename>` with the name (including `.p12` file extension) of your P12 certificate and `<p12_password>` with the password you associated with it.

5. Now press *Enter* to begin compilation. If successful, a file named `render-test-gpu.ipa` will be generated. Deploy the IPA to your device for testing.

The FLA used in this recipe was selected for a reason. It takes advantage of GPU rendering and, therefore, shows one of the advantages of compiling using ADT from the latest AIR SDK. Applications compiled using ADT will exhibit significantly improved rendering performance compared to those compiled from Flash Professional CS5. To see the difference, compile a version of the IPA from CS5 and install it to your device too.

How it works...

ADT is the same tool that Flash Professional CS5.5 uses to compile an `.ipa` file. By downloading and installing the latest version of the AIR SDK to your development computer, you will be able to utilize ADT even if you don't have Flash Professional CS5.5 installed.

You will still need to use Flash Professional CS5 to generate an application descriptor file and publish your `.swf` file. However, the actual packaging of these files into an `.ipa` will be performed by ADT.

Perhaps the single biggest reason for using ADT above PFI is the improved rendering performance of apps compiled using it. If you are restricted to Flash Professional CS5, then you really should consider downloading and using ADT to create your `.ipa` files.

See also

- *Compiling from Flash Professional*, *Chapter 2*
- *Editing the application descriptor file*, *Chapter 3*
- *Understanding GPU-Blend mode*, *Chapter 6*
- *Understanding GPU-Vector mode*, *Chapter 6*
- *Compiling from the command line using Windows*
- *Compiling from the command line using Mac OS X*
- *Enabling Interpreter Mode*