

Building ADOBE® AIR® Applications with the Packager for iPhone®



Legal notices

For legal notices, see http://help.adobe.com/en_US/legalnotices/index.html.

Contents

Chapter 1: Getting started building AIR applications for the iPhone

| | |
|---|---|
| Important concepts | 1 |
| Obtaining developer tools from Adobe | 4 |
| Obtaining developer files from Apple | 4 |
| Creating a Hello World iPhone application with Flash Professional CS5 | 8 |

Chapter 2: Compiling and debugging iPhone applications

| | |
|--|----|
| iPhone icon and initial screen images | 12 |
| iPhone application settings | 14 |
| Compiling an iPhone application installer (IPA) file | 19 |
| Installing an iPhone application | 21 |
| Debugging an iPhone application | 23 |
| Submitting your iPhone application to the App Store | 25 |

Chapter 3: ActionScript 3.0 API support for mobile devices

| | |
|--|----|
| ActionScript 3.0 APIs unsupported on mobile devices | 26 |
| ActionScript APIs specific to mobile AIR applications | 28 |
| ActionScript 3.0 APIs of special interest to mobile application developers | 32 |

Chapter 4: iPhone application design considerations

| | |
|--|----|
| Hardware acceleration | 33 |
| Other ways to improve display object performance | 35 |
| Information density | 36 |
| Fonts and text input | 36 |
| Saving application state | 37 |
| Screen orientation changes | 37 |
| Hit targets | 37 |
| Memory allocation | 38 |
| Drawing API | 38 |
| Event bubbling | 38 |
| Optimizing video performance | 38 |
| Flex and Flash components | 39 |
| Reducing application file size | 39 |

Chapter 1: Getting started building AIR applications for the iPhone

You can use Adobe® Flash® Platform tools and ActionScript® 3.0 code to build Adobe® AIR® applications for the iPhone and iPod Touch. These applications are distributed, installed, and run just like other iPhone applications.

Note: The remainder of this document refers to the iPhone and iPod touch together as simply “the iPhone.”

The Packager for iPhone® is included with Adobe® Flash® Professional CS5. The Packager for iPhone compiles ActionScript 3.0 bytecode into native iPhone application code. iPhone applications are distributed as iPhone application installer files (.ipa files), via the iTunes Store.

You can use Flash Professional CS5 or Adobe® Flash® Builder™ 4 to edit the source ActionScript 3.0 content for your application.

To develop iPhone applications, use Flash Professional CS5.

You also need to obtain iPhone developer certificates from Apple.

Important: Before developing iPhone applications, review information on designing applications for the iPhone. See “[iPhone application design considerations](#)” on page 33. Also, learn about the developer files required to build an iPhone application. See “[Obtaining developer files from Apple](#)” on page 4.

Important concepts

It is important to understand the concepts and workflow involved before developing an iPhone application using ActionScript 3.0.

Glossary

The following terms are important to understand when building an iPhone application.

iPhone Dev Center site The Apple Computer website (<http://developer.apple.com/iphone/>) where you can do the following:

- Apply to become an iPhone developer.
- Manage and create iPhone development certificates, provisioning profiles and app IDs (which are defined below).
- Submit applications for the App Store.

iPhone development certificate Used to identify a developer for the purpose of developing applications.

You obtain this file from Apple. You convert this certificate to a P12 certificate file to sign the iPhone application you create using ActionScript 3.0. See *P12 certificate file*.

You do not need an iPhone development certificate to simply debug and test Flash Professional CS5 applications on the development computer. However, you need a development certificate to install and test the application on an iPhone.

The development certificate is different from a distribution certificate, which you use to build a final version of your application. You obtain a distribution certificate from Apple when you build a final version of your application.

Certificate signing request A file that contains personal information used to generate a development certificate. Also known as a CSR file.

Provisioning profile A file that lets you test or distribute an iPhone application. You obtain provisioning profile files from Apple. A provisioning profile is assigned to a specific development certificate, an application ID, and one or more device IDs. There are different types of provisioning profiles:

- **Development provisioning profile**—Used to install a test version of an application to the developer’s iPhone.
- **Test provisioning profile**—Also known as an ad-hoc provisioning profile. Used to distribute a test version of the application to multiple users (and iPhone units). With this provisioning profile and the test application, users can test your application without it being submitted to the App Store. Note: you can also use a development provisioning profile to distribute test applications to multiple devices.
- **Distribution provisioning profile**—Used to build an iPhone application to submit your application to the App Store.

App ID A unique string that identifies an iPhone application (or multiple applications) from a specific developer. You create app IDs at the iPhone Dev Center site. Each provisioning profile has an associated app ID or app ID pattern. You use this app ID (or pattern) when developing an application. You use the app ID in the Flash Professional CS5 iPhone Settings dialog box (or in the application descriptor file).

App IDs at the iPhone Dev Center contain a bundle seed ID followed by a bundle identifier. The bundle seed ID is a string of characters, such as 5RM86Z4DJM, that Apple assigns to the App ID. The bundle identifier contains a reverse domain name string that you pick. The bundle identifier may end in an asterisk (*), indicating a wildcard app ID.

Examples are:

- 5RM86Z4DJM.com.example.helloWorld
- 96LPVWEASL.com.example.* (a wildcard app ID)

There are two types of app ID at the iPhone Dev Center:

- **Wildcard app IDs**—At the iPhone Dev Center, these app IDs end in an asterisk (*), such as 96LPVWEASL.com.myDomain.* or 96LPVWEASL.*. With a provisioning profile that uses this kind of app ID, you can generate test applications that use an app ID that matches the pattern. For the application’s app ID, you can replace the asterisk with any string of valid characters. For example, if the iPhone Dev Center site specifies 96LPVWEASL.com.example.* as the app ID, you can use com.example.foo or com.example.bar as the application’s app ID.
- **Specific app IDs**—These define a unique app ID to use in an application. At the iPhone Dev Center, these app IDs do not end in an asterisk. An example is 96LPVWEASL.com.myDomain.myApp. With a provisioning profile that uses this kind of app ID, applications much match the app ID exactly. For example, if the iPhone Dev Center site specifies 96LPVWEASL.com.example.helloWorld as the app ID, you must use com.example.foo as the application’s app ID.

When developing your application, you specify the app ID in the iPhone settings dialog box in Flash Professional CS5 or in the application descriptor file. For more details on app IDs, see the “Deployment tab” section of “[Setting iPhone application properties in Flash Professional CS5](#)” on page 14 or see “[Setting iPhone application properties in the application descriptor file](#)” on page 16.

Important: When specifying the app ID, disregard the bundle seed ID portion of the app ID. For example, if Apple lists your app ID as 96LPVWEASL.com.example.bob.myApp, disregard 96LPVWEASL—use com.example.bob.myApp as the app ID. If Apple lists your app ID as 5RM86Z4DJM.*, disregard 5RM86Z4DJM—this is a wildcard app ID.

You can find the app ID (or wildcard app ID pattern) associated with a provisioning profile at the iPhone Dev Center (<http://developer.apple.com/iphone>). Go to the iPhone Developer Program Portal and then go to the Provisioning section.

P12 certificate file A P12 file (a file with a .p12 extension) is a type of certificate file (a Personal Information Exchange file). The Packager for iPhone uses this type of certificate to build an iPhone application. You convert the developer certificate you receive from Apple into this form of certificate.

Unique Device ID A unique code identifying a specific iPhone. Also known as a UDID or a device ID.

Overview of the development workflow

When developing an application for the iPhone, you follow these steps:

- 1 Install Flash Professional CS5 from Adobe.
- 2 Install iTunes.
- 3 Obtain developer files from Apple. These files include the developer certificate and provisioning profiles. See “[Obtaining developer files from Apple](#)” on page 4.
- 4 Convert the development certificate to a P12 certificate file. Flash CS5 requires the certificate to be a P12 certificate. See “[Obtaining developer files from Apple](#)” on page 4.
- 5 Use iTunes to associate your provisioning profile with your iPhone.
- 6 Write the application in Flash Professional CS5.

It is important to understand the best practices for designing and optimizing code for an iPhone application. See “[iPhone application design considerations](#)” on page 33.

Also, some ActionScript 3.0 APIs are limited or unsupported on the iPhone. See “[ActionScript 3.0 API support for mobile devices](#)” on page 26.

You can also use Flash Builder 4.0 to edit the ActionScript 3.0 code for the application.

You can use Flash Professional CS5 to test your application on the development computer.

- 7 Create icon art and initial screen art for the application. Every iPhone application includes a set of icons that identify it to users. The iPhone displays the initial screen image as the program is loading. See “[iPhone icon and initial screen images](#)” on page 12.
- 8 Edit the iPhone settings. These settings include the following:
 - The identity of the application (including the filename, the application name, the version number, and the app ID)
 - The location of the source icon art for the application
 - The P12 certificate and the provisioning profile assigned to the application
 - The initial aspect ratio of the application

In Flash Professional CS5, you can edit these settings in the iPhone Settings dialog box. For details, see “[Setting iPhone application properties in Flash Professional CS5](#)” on page 14.

You can also edit these settings directly in the application descriptor file. For more information, see “[Setting iPhone application properties in the application descriptor file](#)” on page 16.

- 9 Compile the IPA file using the Packager for iPhone. See “[Compiling an iPhone application installer \(IPA\) file](#)” on page 19.
- 10 Install and test the application on your iPhone. Use iTunes to install the IPA file.

For ad hoc distribution, repeat this general process, but use a test provisioning profile instead of a development provisioning profile. For the final distribution of the application, repeat this process using the distribution provisioning profile. (See the “[Glossary](#)” on page 1 for information on the different types of provisioning profiles.)

When you have built a distribution version of your application, see the instructions in “[Submitting your iPhone application to the App Store](#)” on page 25.

For a quick tutorial on building a basic iPhone application, see “[Creating a Hello World iPhone application with Flash Professional CS5](#)” on page 8.

Obtaining developer tools from Adobe

To develop iPhone applications using ActionScript 3.0, you need Flash Professional CS5.

Important: You should update the version of the Packager for iPhone from the preview version that was included in Flash Professional CS5. In Flash Professional CS5, select Help > Updates.

You can also use Flash Builder 4 to edit ActionScript code. Flash Builder 4 is available at <http://www.adobe.com/products/flashbuilder/>.

Obtaining developer files from Apple

As in developing any application for the iPhone, you must first obtain iPhone developer files from Apple. You need to obtain an iPhone developer certificate and a mobile provisioning profile. You also need to obtain other provisioning profiles. See the “[Glossary](#)” on page 1 for definitions of these files.

Note: Obtaining these files is an important part of the application development process. Be sure to complete this process before developing your application. Obtaining developer files is not a simple process. Read these instructions and the instructions at the Apple iPhone Dev Center site carefully.

Obtaining and working with iPhone developer files

You need to obtain an iPhone developer certificate and provisioning profiles from Apple. You also need to convert the certificate into a P12 certificate.

Install iTunes

You need iTunes to install your application on your iPhone. Also, you use iTunes to determine the device ID of your iPhone. You will need to know the device ID when applying for an iPhone developer certificate.

Apply for an iPhone developer certificate and create a provisioning profile

If you have not already done so, sign up to be a registered iPhone developer at the Apple iPhone Dev Center site (<http://developer.apple.com/iphone/>).

Note: You do not need the iPhone SDK or XCode to develop AIR applications for the iPhone. You do need to be a registered iPhone developer. And you need to obtain a developer certificate and a provisioning profile.

- 1 Log in to the iPhone Dev Center using your iPhone developer account ID.
- 2 At the iPhone Dev Center, apply for (and purchase) an iPhone developer certificate.

You will receive an e-mail message from Apple containing your iPhone Developer Program activation code.

- 3 Return to the iPhone Dev Center. Follow the instructions on activating your developer program (and enter your activation code when prompted).
- 4 When your activation code is accepted, go to the iPhone developer Program Portal section of the iPhone Dev Center.
- 5 Create a certificate signing request file. You will use this file to obtain a iPhone Development Certificate. For instructions, see “[Generating a certificate signing request](#)” on page 5.
- 6 In the next step, you will be asked to provide the Device ID (or Unique Device ID) for your iPhone. You can obtain the UDID from iTunes:
 - a Connect your iPhone with a USB cable. Then, in iTunes, select the summary tab for the iPhone.
 - b Once you have downloaded the provisioning profile from the iPhone developer center site, add it to iTunes.
 - c Then click the Serial Number displayed. The UDID is now displayed. Click Command-C on Mac or Control-C on Windows to copy the UDID to the clipboard.

- 7 Create and install a provisioning profile and an iPhone development certificate.

Follow the instructions at the iPhone Dev Center. Look for instructions at the iPhone Developer Program Portal section. You may want to use the Development Provisioning Assistant to obtain your development certificate and create your provisioning profile.

Ignore steps involving XCode. You do not need to use XCode to develop iPhone applications using Flash Professional CS5.

- 8 In iTunes, select File > Add To Library. Then select the provisioning profile file (which has mobileprovision as the filename extension). Then sync your iPhone with iTunes.

This lets you test the application associated with this provisioning profile on your iPhone.

To verify that a specific provisioning profile is added to iTunes, you can try to add it to the library. If iTunes asks for you to replace an existing provisioning profile, you can press the Cancel button. (The profile is already installed.) Also, you can check the provisioning profiles installed on your iPhone:

- a Open the Settings application on your iPhone.
 - b Open the General category.
 - c Tap Profiles. The Profiles page lists your installed provisioning profiles.
- 9 If you have not done so, download the iPhone development certificate file (a .cer file).

The Development Provisioning Assistant may have provided you with a link to download this file. You can also find the file at the Certificates section of the Provisioning Portal at the Apple iPhone Dev Center site (<http://developer.apple.com/iphone/>).

- 10 Next, you will convert the iPhone developer certificate to a P12 file. For instructions, see “[Converting a developer certificate into a P12 file](#)” on page 6.

You can now create a simple Hello World application. See “[Creating a Hello World iPhone application with Flash Professional CS5](#)” on page 8.

Generating a certificate signing request

To obtain a developer certificate, you generate a certificate signing request file, which you submit at the Apple iPhone Dev Center site.

Generate a certificate signing request on Mac OS

On Mac OS, you can use the Keychain Access application to generate a code signing request. The Keychain Access application is in the Utilities subdirectory of the Applications directory. On the Keychain Access menu, select Certificate Assistant > Request a Certificate from a Certificate Authority.

- 1 Open Keychain Access.
- 2 On the Keychain Access menu, select Preferences.
- 3 In the Preferences dialog box, click Certificates. Then set Online Certificate Status Protocol and Certificate Revocation List to Off. Close the dialog box.
- 4 On the Keychain Access menu, select Certificate Assistant > Request a Certificate from a Certificate Authority.
- 5 Enter the e-mail address and name that matches your iPhone developer account ID. Do not enter a CA e-mail address. Select Request is Saved to Disk and then click the Continue button.
- 6 Save the file (CertificateSigningRequest.certSigningRequest).
- 7 Upload the CSR file to Apple at the [iPhone developer site](#). (See “Apply for an iPhone developer certificate and create a provisioning profile”.)

Generate a certificate signing request on Windows

For Windows developers, it may be easiest to obtain the iPhone developer certificate on a Mac computer. However, it is possible to obtain a certificate on a Windows computer. First, you create a certificate signing request (a CSR file) using OpenSSL:

- 1 Install OpenSSL on your Windows computer. (Go to <http://www.openssl.org/related/binaries.html>)
You may also need to install the Visual C++ 2008 Redistributable files, listed on the Open SSL download page. (You do *not* need Visual C++ installed on your computer.)

- 2 Open a Windows command session, and CD to the OpenSSL bin directory (such as c:\OpenSSL\bin\).
- 3 Create the private key by entering the following in the command line:

```
openssl genrsa -out mykey.key 2048
```

Save this private key file. You will use it later.

When using OpenSSL, do not ignore error messages. If OpenSSL generates an error message, it may still output files. However, those files may not be usable. If you see errors, check your syntax and run the command again.

- 4 Create the CSR file by entering the following in the command line:

```
openssl req -new -key mykey.key -out CertificateSigningRequest.certSigningRequest -subj  
"/emailAddress=yourAddress@example.com, CN=John Doe, C=US"
```

Replace the e-mail address, CN (certificate name), and C (country) values with your own.

- 5 Upload the CSR file to Apple at the [iPhone developer site](#). (See “Apply for an iPhone developer certificate and create a provisioning profile”.)

Converting a developer certificate into a P12 file

To develop iPhone applications using Flash Professional CS5, you must use a P12 certificate file. You generate this certificate based on the Apple iPhone developer certificate file you receive from Apple.

Convert the iPhone developer certificate to a P12 file on Mac OS

Once you have downloaded the Apple iPhone certificate from Apple, export it to the P12 certificate format. To do this on Mac® OS:

- 1 Open the Keychain Access application (in the Applications/Utilities folder).
- 2 If you have not already added the certificate to Keychain, select File > Import. Then navigate to the certificate file (the .cer file) you obtained from Apple.
- 3 Select the Keys category in Keychain Access.
- 4 Select the private key associated with your iPhone Development Certificate.
The private key is identified by the iPhone Developer: <First Name> <Last Name> public certificate that is paired with it.
- 5 Select File > Export Items.
- 6 Save your key in the Personal Information Exchange (.p12) file format.
- 7 You will be prompted to create a password that is used when you attempt to import this key on another computer.

Convert an Apple developer certificate to a P12 file on Windows

To develop iPhone applications using Flash CS5, you must use a P12 certificate file. You generate this certificate based on the Apple iPhone developer certificate file you receive from Apple.

- 1 Convert the developer certificate file you receive from Apple into a PEM certificate file. Run the following command-line statement from the OpenSSL bin directory:

```
openssl x509 -in developer_identity.cer -inform DER -out developer_identity.pem -outform PEM
```
- 2 If you are using the private key from the keychain on a Mac computer, convert it into a PEM key:

```
openssl pkcs12 -nocerts -in mykey.p12 -out mykey.pem
```
- 3 You can now generate a valid P12 file, based on the key and the PEM version of the iPhone developer certificate:

```
openssl pkcs12 -export -inkey mykey.key -in developer_identity.pem -out iphone_dev.p12
```

If you are using a key from the Mac OS keychain, use the PEM version you generated in the previous step. Otherwise, use the OpenSSL key you generated earlier (on Windows).

Managing certificates, device IDs, application IDs, and provisioning profiles

You can manage certificates, device IDs, app IDs, and provisioning profiles at the Apple iPhone Dev Center site (<http://developer.apple.com/iphone/>). Go to the iPhone Developer Program Portal section of the site.

- Click the Certificates link to manage your development certificates. You can create a certificate, download a certificate, or revoke a certificate. To create a certificate, you must first create a certificate signing request. See “Generating a certificate signing request” on page 5.
- Click the Devices link to manage the list of devices your development application can be installed on.
- Click the App IDs link to manage your app IDs. When you create a provisioning profile, it is bound to an app ID.
- Click the Provisioning link to manage your provisioning profiles. You can also use the Development Provisioning Assistant to create development provisioning profiles.
- Click the Distribution link when you want to submit your application to the App Store or create an Ad Hoc version of your application. This section has a link to the iTunes Connect site, which you use to submit an application to the App Store.

Creating a Hello World iPhone application with Flash Professional CS5

Important: Before you create the application, download the required developer applications and files. See [“Obtaining developer tools from Adobe”](#) on page 4 and [“Obtaining developer files from Apple”](#) on page 4.

Create a Flash Professional CS5 project

You can generate an iPhone application directly in Flash Professional CS5:

- 1 Open Flash CS5.
- 2 Select File > New.
- 3 Select iPhone OS.
- 4 Click the OK button.

Add content to the application

Next, add the text “Hello world!” to the application:

- 1 Select the Text Tool and click the stage.
- 2 In the Properties settings for the text field, select Classic Text (not TLF Text).
This is a basic application, and Classic Text is adequate. To use TLF Text, you need to apply some other settings. See [“Fonts and text input”](#) on page 36.
- 3 In the new text field, type “Hello World!”
- 4 Select the text field with the Selection Tool.
- 5 Then open the Properties inspector, make the following settings:
 - Character > Family: `_sans`
 - Character > Size: 50
 - Position > X: 20
 - Position > Y: 20
- 6 Save the file.
- 7 Select Control > Test Movie > In AIR Debug Launcher (Mobile).
Flash Professional CS5 compiles the SWF content and displays a version of the application in the AIR Debug Launcher (ADL). This gives you a quick preview of your application.

Create icon art and initial screen art for the application

All iPhone applications have icons that appear in the user interface of the iTunes application and on the iPhone screen.

- 1 Create a directory within your project directory, and name it icons.
- 2 Create three PNG files in the icons directory. Name them Icon29.png, Icon57.png, and Icon512.png.
- 3 Edit the PNG files to create appropriate art for your application. The files must be 29-by-29 pixels, 57-by-57 pixels, and 512-by-512 pixels. For this test, you can simply use solid color squares as the art.

All iPhone applications display an initial image while the application loads on the iPhone. You define the initial image in a PNG file:

- 1 In the main development directory, create a PNG file named Default.png. (Do *not* put this file in the icons subdirectory. Be sure to name the file Default.png, with an uppercase D.)
- 2 Edit the file so that it is 320 pixels wide and 480 pixels high. For now, the content can be a plain white rectangle. (You will change this later.)

Note: When you submit an application to the Apple App Store, you use a JPG version (not a PNG version) of the 512-pixel file. You use the PNG version while testing development versions of an application.

For detailed information on these graphics, see “[iPhone icon and initial screen images](#)” on page 12.

Edit application settings

Important: If you have not already done so, download the required developer applications and files for iPhone development. See “[Obtaining developer files from Apple](#)” on page 4.

In the Flash Professional CS5 iPhone Settings dialog box, define many basic properties of the iPhone application.

- 1 Choose File > iPhone OS Settings.
- 2 In the General tab, make the following settings:
 - Output file: HelloWorld.ipa
This is the filename of the iPhone installer file to be generated.
 - App name: Hello World
This is the name of the application displayed under the application icon in the iPhone.
 - Version: 1.0
The version of the application.
 - Aspect ratio: portrait
 - Full screen: selected.
 - Auto orientation: deselected.
 - Rendering: CPU
The other options, GPU and Auto, use hardware acceleration for rendering. This feature can help improve performance for graphics-intensive applications (such as games) that have been designed to take advantage of hardware acceleration. For more information, see “[Hardware acceleration](#)” on page 33.
 - Included files: Add the initial screen art file (Default.png) to the Included Files list.

Note: For this Hello World example, do not modify the settings from those provided in these instructions. Some settings, such as the Version setting, have specific restrictions. These restrictions are described in “[iPhone application settings](#)” on page 14.

- 3 In the Deployment tab, make the following settings:
 - Certificate: Browse and select the .p12 certificate based on the developer certificate you obtained from Apple.
This certificate is used to sign the file. You must convert the Apple iPhone certificate to the .p12 format. For more information, see “[Obtaining developer tools from Adobe](#)” on page 4.
 - Password: enter the password for the certificate.

- Provisioning file: Browse and select the developer provisioning file that you obtained from Apple. See [“Obtaining developer files from Apple”](#) on page 4.
- App ID: If this field is selectable, you can enter an application ID that matches the application ID you provided to Apple (such as com.example.as3.HelloWorld).

The application ID uniquely identifies your application.

If the field is not selectable, then the provisioning profile is bound a specific application ID. The application ID is displayed in the field.

For details on specifying an app ID, see the “Deployment tab” section of [“Setting iPhone application properties in Flash Professional CS5”](#) on page 14.

- 4 In the Icons tab, click the Icon 29 x 29 in the Icons list. Then specify the location of the 29 x 29-pixel PNG file you created earlier (see [“Create icon art and initial screen art for the application”](#) on page 8). Then specify PNG files for the 57 x 57-pixel icon and the 512 x 512-pixel icon.
- 5 Click the OK button.
- 6 Save the file.

For details on the application settings, see [“iPhone application settings”](#) on page 14.

Compiling the IPA file

You can now compile the IPA installer file:

- 1 Select File > Publish.
- 2 In the iPhone Settings dialog box, click the OK button.

The Packager for iPhone generates the iPhone application installer file, HelloWorld.ipa file, in the project directory. Compiling the IPA file can take a few minutes.

Install the application on the iPhone

To install the iPhone application for testing on an iPhone:

- 1 Open the iTunes application.
- 2 If you have not already done so, add the provisioning profile for this application to iTunes. In iTunes, select File > Add File To Library. Then select the provisioning profile file (which has mobileprovision as the file type).

For now, to test the application on your developer iPhone, use the development provisioning profile.

Later, when distributing an application to the iTunes Store, use the distribution profile. To distribute the application ad-hoc (to multiple devices without going through the iTunes Store), use the ad-hoc provisioning profile.

For more information on provisioning profiles, see [“Obtaining developer files from Apple”](#) on page 4.

- 3 Some versions of iTunes do not replace the application if the same version of the application is already installed. In this case, delete the application from your device and from the list of applications in iTunes.
- 4 Double-click the IPA file for your application. It should appear in the list of applications.
- 5 Connect your iPhone to the USB port on your computer.
- 6 In iTunes, check the Application tab for the device, and ensure that the application is selected in the list of applications to be installed.

- 7 Select the device in the left-hand list of the iTunes application. Then click the Sync button. When the sync completes, the Hello World application appears on your iPhone.

If the new version is not installed, delete it from your iPhone and from the list of applications in iTunes, and then redo this procedure. This may be the case if the currently installed version uses the same application ID and version.

If iTunes displays an error when you try to install your application, see “Troubleshooting application installation problems” in “[Installing an iPhone application](#)” on page 21.

Edit the initial screen graphic

Before you compiled your application, you created a Default.png file (see “[Create icon art and initial screen art for the application](#)” on page 8). This PNG file serves as the startup image while the application loads. When you tested the application on your iPhone, you may have noticed this blank screen at startup.

You should change this image to match the startup screen of your application (“Hello World!”):

- 1 Open your application on your device. When the first “Hello World” text appears, press and hold the Home button (below the screen). While holding the Home button, press the Power/Sleep button (at the top of the iPhone). This takes a screenshot and sends it to the Camera Roll.
- 2 Transfer the image to your development computer by transferring photos from iPhoto or another photo transfer application. (On Mac OS, you can also use the Image Capture application.)

You can also e-mail the photo to your development computer:

- Open the Photos application.
- Open the Camera Roll.
- Open the screenshot image you captured.
- Tap the image and then tap the “forward” (arrow) button in the bottom-left-hand corner. Then click the Email Photo button and send the image to yourself.

- 3 Replace the Default.png file (in your development directory) with a PNG version of the screen capture image.
- 4 Recompile the application (see “[Compiling the IPA file](#)” on page 10) and reinstall it on your iPhone.

The application now uses the new startup screen as it loads.

Note: You can create any art you’d like for the Default.png file, as long as it is the correct dimensions (320 by 480 pixels). However, it is often best to have the Default.png image match the initial state of your application.

Chapter 2: Compiling and debugging iPhone applications

You can compile an iPhone application using the Packager for iPhone. The Packager for iPhone is included with Flash Professional CS5.

You can debug the application on the development computer. You can also install a debug version on the iPhone and receive `trace()` output in Flash Professional CS5.

For a tutorial on how to build an iPhone application from start to finish, see [“Creating a Hello World iPhone application with Flash Professional CS5”](#) on page 8.

iPhone icon and initial screen images

All iPhone applications have icons which appear in the user interface of the iTunes application and on the iPhone.

iPhone application icons

You define the following icons for an iPhone application:

- A 29-by-29-pixel icon—Spotlight search results on the iPhone and iPod touch use this icon.
- A 48-by-48-pixel icon—Spotlight search results on the iPad use this icon.
- A 57-by-57-pixel icon—The iPhone and iPod touch home screens display this icon.
- A 72-by-72-pixel icon (optional)—The iPad home screen displays this icon.
- A 512-by-512-pixel icon—iTunes displays this icon. The 512-pixel PNG file is used only for testing development versions of your application. When you submit the final application to the Apple App Store, you submit the 512 image separately, as a JPG file. It is not included in the IPA.

In Flash Professional CS5, add these icons in Icons tab of the the iPhone Settings dialog box. See [“Setting iPhone application properties in Flash Professional CS5”](#) on page 14.

You can also add the locations of the icons to the application descriptor file:

```
<icon>
  <image29x29>icons/icon29.png</image29x29>
  <image57x57>icons/icon57.png</image57x57>
  <image72x72>icons/icon72.png</image72x72>
  <image512x512>icons/icon512.png</image512x512>
</icon>
```

The iPhone adds a glare effect to the icon. You do not need to include it in your source image. To remove this default glare effect, add the following to the `InfoAdditions` element in the application descriptor file:

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

See [“Setting iPhone application properties in the application descriptor file”](#) on page 16.

The initial screen art (Default.png)

All iPhone applications display an initial image while the application loads on the iPhone. You define the initial image in a PNG file named `Default.png`. In the main development directory, create a PNG file named `Default.png`. (Do *not* put this file in a subdirectory. Be sure to name the file `Default.png`, with an uppercase D.)

The `Default.png` file is 320 pixels wide by 480 pixels tall, regardless of the initial orientation of the application or whether it is full-screen or not.

If the initial orientation of your application is landscape, use the same dimensions that a portrait application uses: 320 pixels wide by 480 pixels high. However, rotate the artwork 90° counter-clockwise in the PNG file. The left side of the PNG art corresponds to the top of the iPhone screen in landscape mode. (For information on setting the initial application orientation, see [“iPhone application settings”](#) on page 14.)

For an application that is not full-screen, the top 20 pixels of the default image art are ignored. The iPhone displays its status bar over the 20 pixel-wide rectangle at the top of the default image. In a landscape-orientation application, this region corresponds to the left 20 pixel-wide rectangle of the `Default.png` file (which displays on the top in landscape mode). In a portrait-orientation application, this region is the top 20 pixel-wide rectangle of the `Default.png` file.

For most applications, the `Default.png` image should match the startup screen of the application. To take a screenshot of the startup screen of your application:

- 1 Open your application on the iPhone. When the first screen of the user interface appears, press and hold the Home button (below the screen). While holding the Home button, press the Power/Sleep button (at the top of the device). This takes a screenshot and sends it to the Camera Roll.
- 2 Transfer the image to your development computer by transferring photos from iPhoto or another photo transfer application. (On Mac OS, you can also use the Image Capture application.)

You can also e-mail the photo to your development computer:

- Open the Photos application.
- Open the Camera Roll.
- Open the screenshot image you captured.
- Tap the image and then tap the “forward” (arrow) button in the bottom-left-hand corner. Then click the Email Photo button and send the image to yourself.

Note: You can create any art you’d like for the `Default.png` file, as long as it is the correct dimensions. However, it is often best to have the `Default.png` image match the initial state of your application.

Do not include text in the `Default.png` image if your application is localized into multiple languages. The `Default.png` is static, and the text would not match other languages.

In Flash Professional CS5, be sure to add the `Default.png` file to the Included Files list in the iPhone settings dialog box. See [“Setting iPhone application properties in Flash Professional CS5”](#) on page 14.

When compiling using the PFI application on the command line, be sure to reference this file in the list of included assets. See [“Creating an iPhone application installer file from the command line”](#) on page 20.

iPhone application settings

Application settings include:

- The application name
- The IPA filename
- The application version
- The initial screen orientation of the application, and whether the screen orientation rotates automatically when the iPhone is rotated
- Whether the initial view is full-screen or not
- Information about the application's icons
- Information about hardware acceleration

You can edit application settings in Flash Professional CS5.

You can also edit them in the application descriptor file. The application descriptor file is an XML file that contains settings for the application.

Setting iPhone application properties in Flash Professional CS5

The Flash Professional CS5 iPhone Settings dialog box lets you define many basic properties of the iPhone application.

To open the iPhone Settings dialog box:

- ❖ Choose File > iPhone Settings.

General tab

The General tab includes the following iPhone-related settings:

- Output file—The name of the application displayed under the application icon in the iPhone. Do not include a plus sign (+) character in the output filename.
- App name—The name of the application displayed under the application icon in the iPhone. Do not include a plus sign (+) character in the app name.
- Version—Helps users determine which version of your application they are installing. The version is used as the `CFBundleVersion` of the iPhone application. It must be in a format similar to `nnnnn[.nn[.nn]]` where `n` is a digit 0-9 and brackets indicate optional components, such as 1, 1.0, or 1.0.1. iPhone versions must contain only digits and decimal points. iPhone versions can contain up to two decimal points.
- Aspect ratio—The initial aspect ratio of the application (portrait or landscape).
- Full screen—Whether the application uses the full screen, or if it displays the iPhone status bar.
- Auto orientation—Select this application to have the application's display contents reorient when the iPhone is reoriented.

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

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

- Rendering—How display objects are rendered on the iPhone:
 - CPU—The application uses the CPU to render all display objects. No hardware acceleration is used.

- GPU—The application uses the iPhone GPU to composite bitmaps.
- Auto—This feature has not been implemented.

For more information, see “[Hardware acceleration](#)” on page 33.

- Included files—Add all files and directories to package in the iPhone application. The main SWF file and the application descriptor file are included by default. Add any other required assets to the Included Files list. Be sure to add the initial screen art file (Default.png) to the Included Files list.

Deployment tab

The Deployment tab includes signing and compilation settings for the application:

- iPhone digital signature—Specify a P12 certificate file and the password for the certificate. You must convert the Apple iPhone certificate to the .p12 format. For more information, see “[Obtaining developer files from Apple](#)” on page 4.
- Provisioning file—Point to the provisioning file for this application, which you obtained from Apple. For more information, see “[Obtaining developer files from Apple](#)” on page 4.
- App ID—The app ID uniquely identifies your application. If the provisioning file is tied to a specific application ID, Flash Professional CS5 sets this field, and you cannot edit it. Otherwise, the provisioning profile allows multiple (wildcard) app IDs. Provide an application ID that matches the application ID wildcard pattern you provided to Apple:

- If your Apple app ID is com.myDomain.*, the app ID in the iPhone Settings dialog box must start with com.myDomain. (such as com.myDomain.myApp or com.myDomain.app22).
- If your Apple app ID is *, the App ID in the iPhone Settings dialog box can be any string of valid characters.

You can find the Apple app ID (or wildcard app ID pattern) associated with your provisioning profile at the iPhone Dev Center (<http://developer.apple.com/iphone>). Go to the iPhone Developer Program Portal and then go to the Provisioning section.

Important: Disregard the characters at the front of the Apple app ID. Apple calls this string the Bundle Seed ID. For example, if Apple lists your app ID as 96LPVWEASL.com.example.bob.myApp, disregard 96LPVWEASL—use com.example.bob.myApp as the app ID. If Apple lists your app ID as 5RM86Z4DJM.*, disregard 5RM86Z4DJM—this is a wildcard app ID.

- iPhone deployment type:
 - Quick publishing for device testing—Choose this option to quickly compile a version of the application for testing on your developer iPhone.
 - Quick publishing for device debugging—Choose this option to quickly compile a debug version of the application for testing on your developer iPhone. With this option, the Flash Professional CS5 debugger can receive `trace()` output from the iPhone application. (See “[Debugging an iPhone application](#)” on page 23.)
 - Deployment - Ad Hoc—Choose this option to create an application for ad hoc deployment. See the Apple iPhone developer center
 - Deployment - Apple App Store—Choose this option to create a final version of the IPA file for deployment to the Apple App Store.

Icons tab

On the Icons tab, specify the location of the 29 x 29-pixel icon image, the 48 x 48-pixel icon image, the 57 x 57-pixel icon image, the 72 x 72-pixel icon image, and the 512 x 512-pixel icon image. See “[iPhone icon and initial screen images](#)” on page 12.

Note: Options for the 48 x 48-pixel and 72 x 72-pixel are not included in the version of the Packager for iPhone Preview included with Flash Professional CS5. In Flash Professional CS5 select Help > Updates to add these options.

Setting iPhone application properties in the application descriptor file

The application descriptor file is an XML file containing properties for the entire application, such as its name, version, copyright, and other settings.

Flash Professional CS5 generates an application descriptor file based on the settings in the iPhone Settings dialog box. However, you can also edit the application descriptor file in a text editor. 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. Edit the application descriptor file if you want to define settings not supported in the Flash Professional CS5 iPhone Settings dialog box. For example, you can define the `InfoAdditions` element to define info.plist settings for the application.

Important: Do not edit the application descriptor file while the Flash Professional CS5 dialog box is open. Save changes to the application descriptor file before you open the iPhone Settings dialog box.

Here's an example application descriptor file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.0">
  <id>com.example.HelloWorld</id>
  <filename>HelloWorld</filename>
  <name>Hello World</name>
  <version>v1</version>
  <initialWindow>
    <renderMode>gpu</renderMode>
    <content>HelloWorld.swf</content>
    <fullScreen>true</fullScreen>
    <aspectRatio>portrait</aspectRatio>
    <autoOrients>true</autoOrients>
  </initialWindow>
  <supportedProfiles>mobileDevice desktop</supportedProfiles>
  <icon>
    <image29x29>icons/icon29.png</image29x29>
    <image57x57>icons/icon57.png</image57x57>
    <image512x512>icons/icon512.png</image512x512>
  </icon>
  <iPhone>
    <InfoAdditions>
      <![CDATA[
        <key>UIStatusBarStyle</key>
        <string>UIStatusBarStyleBlackOpaque</string>
        <key>UIRequiresPersistentWiFi</key>
        <string>NO</string>
      ]]>
    </InfoAdditions>
  </iPhone>
</application>
```

Here are details on the settings in this application descriptor file:

- In the `<application>` element, the AIR 2.0 namespace is required for building iPhone applications:
`<application xmlns="http://ns.adobe.com/air/application/2.0">`
- The `<id>` element:

`<id>com.example.as3.HelloWorld</id>` The application ID uniquely identifies your application. The recommended form is a dot-delimited, reverse-DNS-style string, such as "com.company.AppName". The compiler uses this value as the bundle ID for the iPhone application.

If the provisioning file is tied to a specific app ID, use that app ID in this element. Disregard the characters Apple assigns at the start of the Apple app ID (known as the bundle seed ID). For example, if the app ID for the provisioning profile is 96LPVWEASL.com.example.bob.myApp, use com.example.bob.myApp as the app ID in the application descriptor file.

If the provisioning profile allows multiple (wildcard) app IDs, its app ID ends in an asterisk (such as 5RM86Z4DJM.*). Provide an application ID that matches the app ID wildcard pattern you provided to Apple:

- If your Apple app ID is com.myDomain.*, the app ID in the application descriptor file must start with com.myDomain. You can specify an app ID such as com.myDomain.myApp or com.myDomain.app22.
- If your Apple app ID is *, the App ID in the app descriptor file can be any string of valid characters.

You can find the Apple app ID (or wildcard app ID pattern) associated with your provisioning profile at the iPhone Dev Center (<http://developer.apple.com/iphone>). Go to the iPhone Developer Program Portal and then go to the Provisioning section.

Important: Disregard the characters at the front of the Apple app ID. Apple calls this string the Bundle Seed ID. For example, if Apple lists your app ID as 5RM86Z4DJM.*, disregard 5RM86Z4DJM—this is a wildcard app ID. If Apple lists your app ID as 96LPVWEASL.com.example.bob.myApp, disregard 96LPVWEASL—use com.example.bob.myApp as the app ID.

- The `<filename>` element:

`<filename>HelloWorld</filename>` The name used for the iPhone installer file. Do not include a plus sign (+) character in the filename.

- The `<name>` element:

`<name>Hello World</name>` The name of the application displayed in the iTunes application and in the iPhone. Do not include a plus sign (+) character in the name.

- The `<version>` element:

`<version>1.0</version>` Helps users to determine which version of your application they are installing. The version is used as the CFBundleVersion of the iPhone application. It must be in a format similar to nnnnn[.nn[.nn]] where n is a digit 0-9 and brackets indicate optional components, such as 1, 1.0, or 1.0.1. iPhone versions must contain only digits and decimal points. iPhone versions can contain up to two decimal points.

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

`<content>HelloWorld.swf</content>` Identifies the root SWF file to compile into the iPhone application.

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

`<fullScreen>>true</fullScreen>` Specifies that the application uses the entire screen of the iPhone.

`<aspectRatio>portrait</aspectRatio>` Specifies that the initial aspect ratio of the application is in portrait mode (rather than landscape). Note the Default.png file used to define the initial window of the application should be 320 pixels wide and 480 pixels high, regardless of this setting. (See “[iPhone icon and initial screen images](#)” on page 12.)

`<autoOrients>true</autoOrients>` (Optional) 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. For more information, see [Setting and detecting screen orientation](#).

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;
```

<renderMode>gpu</renderMode> (Optional) The rendering mode used by the application. There are three possible settings:

- `cpu`—The application uses the CPU to render all display objects. No hardware acceleration is used.
- `gpu`—The application uses the iPhone GPU to composite bitmaps.
- `auto`—This feature has not been implemented.

For more information, see “[Hardware acceleration](#)” on page 33.

- The `<profiles>` element:

<profiles>mobileDevice</profiles> Limits the application to be compiled into the mobile device profile. This profile currently only supports iPhone applications. 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. Currently, iPhone is the only supported mobile device.

Limiting the application to a specific profile prevents it from being compiled into other profiles. If you specify no profile, then you can compile an application for any of these profiles. You can specify multiple profiles by listing them each, separated by spaces, in the `<profiles>` element.

Be sure to include `mobileDevice` as a supported profile (or leave the `<profiles>` element empty).

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

<image29x29>icons/icon29.png</image29x29> This is the image used in Spotlight search results.

<image48x48>icons/icon48.png</image48x48> This is the image used in Spotlight search results on the iPad.

<image57x57>icons/icon57.png</image57x57> This is the image used on the iPhone and iPod Touch home screen.

<image72x72>icons/icon72.png</image72x72> This is the image used on the iPad home screen.

<image512x512>icons/icon512.png</image512x512> This is the image used in the iTunes application.

The Packager for iPhone tool uses the 29, 57, and 512 icons referenced in the application descriptor file. The tool copies them to files called `Icon-Small.png`, `Icon.png`, and `iTunesArtwork` respectively. To avoid having this copy made, you can package those files directly. Package them directly by placing them in the directory that contains the application descriptor file and list the correct names and paths.

The 512 image is for internal testing only. When submitting an application to Apple you submit the 512 image separately. It is not included in the IPA. Specify it so you can make sure that your 512 image looks good in iTunes before submitting it.

- The `<iPhone>` element contains the following child elements to specify iPhone-specific settings:

<InfoAdditions></InfoAdditions> contains the child elements specifying key-value pairs to use as Info.plist settings for the application:

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

In this example, the values set the status bar style of the application and state that the application does not require persistent Wi-Fi access.

The InfoAdditions settings are enclosed in a CDATA tag.

For iPad support, include key-value settings for `UIDeviceFamily`. The `UIDeviceFamily` setting is an array of strings. Each string defines supported devices. The `<string>1</string>` setting defines support for the iPhone and iPod Touch. The `<string>2</string>` setting defines support for the iPad. If you specify only one of these strings, only that device family is supported. For example, the following setting limits support to the iPad:

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

The following sets support for both device families (iPhone/iPod Touch and iPad):

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

For information on other Info.plist settings, see the Apple developer documentation.

Compiling an iPhone application installer (IPA) file

Use the Packager for iPhone to compile an ActionScript-3.0 application into an IPA installer file.

Creating an iPhone application installer file using the Packager for iPhone included with in Flash Professional CS5

To use the Packager for iPhone included with Flash Professional CS5:

- 1 Select File > Publish.
- 2 In the iPhone Settings dialog box, make sure that you have provided values for all settings. Be sure that you have selected the correct options in the Deployment tab. See [“Setting iPhone application properties in Flash Professional CS5”](#) on page 14.
- 3 Click the Publish button.

The Packager for iPhone generates the iPhone application installer (IPA) file. Compiling the IPA file can take a few minutes.

You can also run the Packager for iPhone from the command line. See [“Creating an iPhone application installer file from the command line”](#) on page 20.

Creating an iPhone application installer file from the command line

You can run the Packager for iPhone from the command line. The Packager for iPhone converts the SWF file byte code and other source files into a native iPhone application.

- 1 Open a command shell or a terminal and navigate to the project folder of your iPhone application.
- 2 Next, use the pfi tool to create the IPA file, using the following syntax:

```
pfi -package -target [ipa-test ipa-debug ipa-app-store ipa-ad-hoc] -provisioning-profile  
PROFILE_PATH SIGNING_OPTIONS TARGET_IPA_FILE APP_DESCRIPTOR SOURCE_FILES
```

Change the reference `pfi` to include the full path to the pfi application. The pfi application is installed in the `pfi/bin` subdirectory of the Flash Professional CS5 installation directory

Select the `-target` option that corresponds to the type of iPhone application you want to create:

- `-target ipa-test`—Choose this option to quickly compile a version of the application for testing on your developer iPhone.
- `-target ipa-debug`—Choose this option to compile a debug version of the application for testing on your developer iPhone. With this option, you can use a debug session to receive `trace()` output from the iPhone application.

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

```
-target ipa-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:

```
-target ipa-debug -connect bobroberts-mac.example.com
```

Note: The `-connect` option is not included in the Packager for iPhone Preview included Flash Professional CS5. Update the Packager for iPhone by selecting *Help > Updates* in Flash Professional CS5.

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

If a debug connection attempts fail, the application presents a dialog asking the user to enter the IP address of the debugging host machine. A connection attempt can fail if the device is not connected to wifi. It can also occur if the device is connected but not behind the firewall of the debugging host machine.

For more information, see “[Debugging an iPhone application](#)” on page 23.

You can also include the `-renderingdiagnostics` option to enable the GPU rendering diagnostics feature. For more information, see “[Debugging with GPU rendering diagnostics](#)” in “[Debugging an iPhone application](#)” on page 23.

- `-target ipa-ad-hoc`—Choose this option to create an application for ad hoc deployment. See the Apple iPhone developer center
- `-target ipa-app-store`—Choose this option to create a final version of the IPA file for deployment to the Apple App Store.

Replace the *PROFILE_PATH* with the path to the provisioning profile file for your application. For more information on provisioning profiles, see [“Obtaining developer files from Apple”](#) on page 4.

Replace the *SIGNING_OPTIONS* to reference your iPhone developer certificate and password. Use the following syntax:

```
-storetype pkcs12 -keystore P12_FILE_PATH -storepass PASSWORD
```

Replace *P12_FILE_PATH* with the path to your P12 certificate file. Replace *PASSWORD* with the certificate password. (See the example below.) For more information on the P12 certificate file, see [“Converting a developer certificate into a P12 file”](#) on page 6.

Replace the *APP_DESCRIPTOR* to reference the application descriptor file.

Replace the *SOURCE_FILES* to reference the main SWF file of your project followed by any other assets to include. Include the paths to all icon files you defined in the application settings dialog box in Flash CS5 or in a custom application descriptor file. Also, add the initial screen art file, Default.png.

Consider the following example:

```
pfi -package -target ipa-test -storetype pkcs12 -keystore  
"/Users/Jeff/iPhoneCerts/iPhoneDeveloper_Jeff.p12" -storepass dfb7VKL19 "HelloWorld.ipa"  
"HelloWorld-app.xml" "HelloWorld.swf" "Default.png" "icons/icon29.png" "icons/icon57.png"  
"icons/icon512.png"
```

It compiles a HelloWorld.ipa file using the following:

- A specific PKCS#12 certificate using the certificate password dfb7VKL19
- The HelloWorld-app.xml application descriptor file
- A source HelloWorld.swf file
- Specific Default.png and icon files

The pfi application compiles the application, based on the application descriptor file, the SWF file, and the other assets, into an IPA file.

On Mac OS, you can use a certificate stored in the keychain by adding the following options to the pfi command:

```
-alias ALIAS_NAME -storetype KeychainStore -providerName Apple
```

Replace *ALIAS_NAME* with the alias of the certificate you want to use. When you point to a certificate stored in the Mac keychain, you do specify the alias instead of pointing to a certificate file location.

Installing an iPhone application

To install your development application on an iPhone, you add the provisioning profile to the iPhone, and then you install the application on the iPhone.

Add the provisioning profile to the iPhone

To add the provisioning profile to the iPhone:

- 1 In iTunes, select File > Add To Library. Then select the provisioning profile file (which has mobileprovision as the filename extension).

Make sure that your iPhone is added to the provisioning profile. You can manage provisioning profiles at the Apple iPhone Dev Center site (<http://developer.apple.com/iphone/>). Go to the iPhone Developer Program Portal section of the site. Click the Devices link to manage the list of devices your development application can be installed on. Click the Provisioning link to manage your provisioning profiles.

- 2 Connect your iPhone to your computer and sync.

For information on obtaining a provisioning profile, see “[Obtaining developer files from Apple](#)” on page 4.

Install the application

You install your development application in the same way that you install any other IPA file:

- 1 If you have previously installed a version of the application, delete the application from your device and from the list of applications in iTunes.
- 2 Add the application in iTunes, in any of the following ways:
 - On the File menu (in iTunes), choose the Add to Library command. Then select the IPA file and click the Open button.
 - Double-click the IPA file.
 - Drag the IPA file into the iTunes library.
- 3 Connect your iPhone to the USB port on your computer.
- 4 In iTunes, check the Application tab for the device, and ensure that the application is selected in the list of applications to be installed.
- 5 Sync your iPhone.

Troubleshooting application installation problems

If iTunes displays an error when you try to install the application, check the following:

- Make sure the device ID is added to the provisioning profile.
- To make sure that the provisioning profile is installed, you can drag it to iTunes or use the File > Add to Library command.

Also, check that your application’s app ID matches the Apple app ID:

- If your Apple app ID is com.myDomain.*, the app ID in the app descriptor file or the App ID in the iPhone Settings dialog box must start with com.myDomain (such as com.myDomain.anythinghere).
- If your Apple app ID is com.myDomain.myApp, the app ID in the app descriptor file or the Flash Professional CS5 user interface must be com.myDomain.myApp.
- If your Apple app ID is *, the App ID in the app descriptor file or the Flash Professional CS5 user interface can be anything.

You set the application’s app ID in the iPhone Settings dialog box in Flash Professional CS5 or in the application descriptor file.

Debugging an iPhone application

You can debug the application on the development computer, with the application running in ADL. You can also debug the application on the iPhone.

Some AIR functionality that is not supported on the iPhone is still available when testing an application using ADL (on the development computer). Be aware of these differences when testing content on the desktop. For more information, see [“ActionScript 3.0 APIs unsupported on mobile devices”](#) on page 26.

Debugging the application on the development computer

To debug the application on the development computer using Flash Professional CS5:

- ❖ Choose Debug > Debug Movie > In AIR Debug Launcher (Mobile).

You can also debug the application by calling ADL from the command line. This is the syntax:

```
adl -profile mobileDevice appDescriptorFile
```

Replace *appDescriptorFile* with the path to the application descriptor file.

Be sure to include the `-profile mobileDevice` option.

Debugging the application on the iPhone

To debug the application on the iPhone:

- 1 Compile the application with debug support:
 - In Flash Professional CS5, compile using the “Quick publishing for device debugging” setting. (See [“Creating an iPhone application installer file using the Packager for iPhone included with in Flash Professional CS5”](#) on page 19.)
 - Using the PFI command-line application, compile the application with the `target ipa-debug` option. (See [“Creating an iPhone application installer file from the command line”](#) on page 20.)
- 2 Install the application on the iPhone.
- 3 On the iPhone, turn Wi-Fi on and connect to the same network as that of the development computer.
- 4 Start a debug session on your development computer. In Flash Professional CS5, choose Debug > Begin Remote Debug Session > ActionScript 3.0.
- 5 Run the application on the iPhone.

The debug version of the application will prompt you for the IP address of the developer computer. Enter the IP address and tap the OK button. To obtain the IP address of the development computer.

- On Mac OS, on the Apple menu, choose System Preference. In the System Preferences window, click the Network icon. The Network preferences window lists the IP address.
- On Windows, start a command-line session and run the `ipconfig` command.

The debug session displays any `trace()` output from the application.

When debugging an application installed on the iPhone, Flash Professional CS5 supports all debugging features, including breakpoint control, stepping through code, and variable monitoring.

Debugging with GPU rendering diagnostics

The GPU rendering diagnostics feature lets you see how the application uses hardware acceleration (for applications that use GPU rendering mode). To use this feature, compile the application using the PFI tool on the command line, and include the `-renderingdiagnostics` option:

```
pfi -package -renderingdiagnostics -target ipa-debug -connect ...
```

The `-renderingdiagnostics` flag must directly follow the `-package` flag.

The GPU rendering diagnostic feature displays colored rectangles for all display objects:

- Blue—The display object is not a bitmap or cached as a bitmap, and it is being rendered.

If blue appears repeatedly for a display object that is not changing, it could be because it intersects with moving display objects. For example, the display object may be a background for moving display objects. Consider caching the display object as a bitmap.

If blue appears for an object that you think should be cached, it may be because the object is using an effect that the GPU cannot apply. These effects include certain blend modes, color transforms, the `scrollRect` property, and masks.

The application also displays blue if display objects uploaded to the GPU exceed the memory limits.

The application logs messages for each blue rectangle. The application outputs these messages along with other `trace()` and debug output messages.

- Green—The display object is a bitmap or cached as a bitmap, and it is being uploaded to the GPU for the first time.

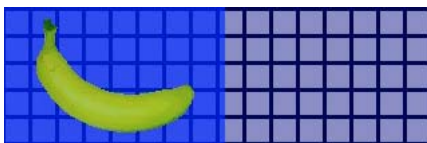
If green appears repeatedly for a display object, then the code in the application is recreating the display object. For example, this can occur if the timeline returns to a frame that creates the display object. Consider modifying the content to prevent re-creation of identical objects.

- Red—The display object is a bitmap or cached as a bitmap, and it is being re-uploaded to the GPU.

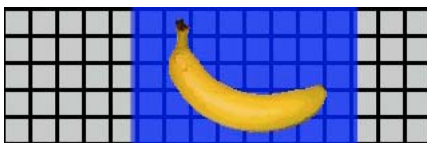
Red appears every time such a display object changes in a way that requires the application to re-render the bitmap representation. For example, if a 2D object that does not have the `cacheAsBitmapMatrix` property set, it is re-rendered when it is scaled or rotated. Re-rendering also occurs when child display objects move or change.

Each colored rectangle fades after four screen redraw cycles, provided the reason for the coloration does not occur again during those cycles. However, if there are no changes onscreen, the diagnostic coloring does not change.

For example, consider a bitmap display object (a banana) in front of a vector background that is not cached as a bitmap. When the banana first renders, it is colored green. When the background first renders, it is colored blue:

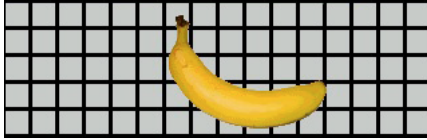


As the banana moves, the CPU must re-render the background, causing the blue shading to appear over the background:



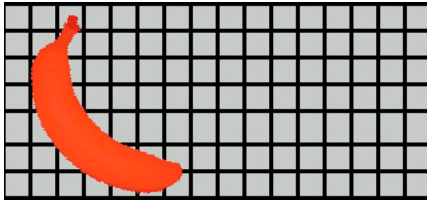
The blue shading over the background reflects redrawn regions that need to be sent to the GPU.

However, if the background is cached as a bitmap, when the banana moves, the rendering diagnostic feature displays no colored tints:



The diagnostic feature displays no colored tints since the GPU retains the background bitmap. The GPU can composite the banana with the background without involving the CPU.

Suppose the banana is a 2D display object that does not have its `cacheAsBitmapMatrix` property set. Whenever the display object rotates (or scales), the rendering diagnostic feature displays red. This indicates that the application has to upload a new version of the display object to the GPU:



Submitting your iPhone application to the App Store

To submit your application to the App Store:

- 1 Obtain a distribution certificate and provisioning profile from the iPhone Dev Center website (<http://developer.apple.com/iphone/>).

A distribution certificate is named “iPhone Developer: XXX.cer,” where XXX is your name.

For more information, see “[Obtaining developer files from Apple](#)” on page 4.

- 2 Convert the distribution certificate into a P12 file.

For more information, see “[Converting a developer certificate into a P12 file](#)” on page 6.

- 3 Compile your application using the P12 file and provisioning profile.

Use the P12 file you made based on the distribution certificate. Use the app ID associated with the distribution provisioning profile.

For more information, see “[Compiling an iPhone application installer \(IPA\) file](#)” on page 19.

- 4 Submit the application at the iPhone Dev Center website (<http://developer.apple.com/iphone/>).

Important: Apple requires that you use the Apple Application Loader program in order to upload applications to the App Store. Apple only publishes Application Loader for Mac OS X. While you can develop an AIR application for the iPhone using a Windows computer, you must have access to a computer running OS X (version 10.5.3, or later) to submit the application to the App Store. You can get the Application Loader program from the Apple iOS Dev Center.

Chapter 3: ActionScript 3.0 API support for mobile devices

In creating mobile AIR applications, you can use the same ActionScript 3.0 APIs available for developing Adobe Flash Player 10.1 and AIR 2 desktop applications. There are, however, exceptions and additions.

ActionScript 3.0 APIs unsupported on mobile devices

Some ActionScript 3.0 APIs that are not supported for applications running in the mobile device profile (such as applications running on the iPhone).

When using the same ActionScript code to develop for multiple profiles (such as desktop and mobile), use code to test if an API is supported. For example, the `NativeWindow` class is not supported in iPhone applications. (iPhone applications cannot use or create native windows.) To test if an application is running on a profile that supports native windows (such as the desktop profile), check the `NativeWindow.isSupported` property.

The following table lists APIs that are not supported in the mobile device profile. It also lists properties you can check to determine when an application is running on a platform that offers support for an API.

| API | Test for support |
|---|--|
| Accessibility | Capabilities.hasAccessibility |
| Camera | Camera.isSupported |
| DatagramSocket | DatagramSocket.isSupported |
| DNSResolver | DNSResolver.isSupported |
| DockIcon | NativeApplication.supportsDockIcon |
| DRMManager | DRMManager.isSupported |
| EncryptedLocalStore | EncryptedLocalStore.isSupported |
| HTMLLoader | HTMLLoader.isSupported |
| LocalConnection | LocalConnection.isSupported |
| Microphone | Microphone.isSupported |
| NativeApplication.exit() | — |
| NativeApplication.menu | NativeApplication.supportsMenu |
| NativeApplication.isSetAsDefaultApplication() | NativeApplication.supportsDefaultApplication |
| NativeApplication.startAtLogin | NativeApplication.supportsStartAtLogin |
| NativeMenu | NativeMenu.isSupported |
| NativeProcess | NativeProcess.isSupported |
| NativeWindow | NativeWindow.isSupported |
| NativeWindow.notifyUser() | NativeWindow.supportsNotification |

| API | Test for support |
|-----------------------|-----------------------------------|
| NetworkInfo | NetworkInfo.isSupported |
| PDF support | HTMLLoader.pdfCapability |
| PrintJob | PrintJob.isSupported |
| SecureSocket | SecureSocket.isSupported |
| ServerSocket | ServerSocket.isSupported |
| Shader | — |
| ShaderFilter | — |
| StorageVolumeInfo | StorageVolumeInfo.isSupported |
| XMLSignatureValidator | XMLSignatureValidator.isSupported |

You cannot write HTML- and JavaScript-based AIR applications for the mobile device profile.

Some ActionScript 3.0 classes are only partially supported:

File

iPhone applications only have access to the application directory and the application storage directory. You can also call `File.createTempFile()` and `File.createTempDirectory()`. Calling an operation to access another directory (such as a `FileStream` read or write method) results in an `IOError` exception.

iPhone applications do not support native file browser dialog boxes, such as the one provided by the `File.browseForOpen()` method.

Loader

In an iPhone application, you cannot use the `Loader.load()` method. However, you cannot run any ActionScript code in SWF content loaded with the `Loader.load()` method. However, you can use assets in the SWF file (such as movie clips, images, fonts, and sounds in the library). You can also use the `Loader.load()` method to load image files.

Video

Only Sorensen video and ON2 VP6 video are supported within an AIR application on the iPhone.

You can use the `navigateToURL()` method to open an H.264 video outside the application. As the `request` parameter, pass a `URLRequest` object with a URL pointing to the video. The video launches in the video player of the iPhone.

Text fields

There are limitations for fonts and other settings of text fields on the iPhone. See “[Fonts and text input](#)” on page 36.

Unsupported APIs and debugging using ADL

Some AIR functionality that is not supported on the iPhone is still available when testing an application using ADL (on the development computer). Be aware of these differences when testing content using ADL.

This functionality includes the following video and audio codecs: Speex (audio), H.264/AVC (video), and AAC (audio). These codecs are unavailable to AIR applications running on the iPhone. However, they continue to function normally on the desktop.

Accessibility and screen reader support works in ADL on Windows. However, these APIs are not supported on the iPhone.

The RTMPE protocol works normally when used from ADL on the desktop. However a NetConnection that attempts to connect using the RTMPE protocol do not succeed on the iPhone.

The Loader class works without additional restrictions when content is executed with ADL. However, when executing on the iPhone, attempts to load SWF content that contains ActionScript byte code results in an error message.

Shader instances execute in ADL. However, on the iPhone pixel bender bytecode is not interpreted and shaders have no graphical effect.

For more information, see “[Debugging an iPhone application](#)” on page 23.

ActionScript APIs specific to mobile AIR applications

The following APIs are available only in AIR applications on mobile devices. They are not currently functional in Flash Player or desktop versions of AIR.

Screen orientation API

The screen orientation API lets you work with the orientation of the stage and the iPhone:

- `Stage.autoOrients`— Whether the application is set to have the stage automatically reorient when the device is rotated. This property is set to `true` when the Auto Orientation option is selected in the Flash Professional CS5 iPhone Settings dialog box. (You can also set the `autoOrients` element to `true` in the application descriptor file.) See “[iPhone application settings](#)” on page 14. You can cancel the automatic reorientation by adding an `orientationChanging` event listener for the Stage object. Calling the `preventDefault()` method of this event object cancels the automatic reorientation.

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;
```

- `Stage.deviceOrientation`—The physical orientation of the device. The `StageOrientation` class defines values for this property.
- `Stage.orientation`—The current orientation of the stage. The `StageOrientation` class defines values for this property.
- `Stage.supportsOrientationChange`—Set to `true` on the iPhone, and `false` in an AIR application.
- `Stage.setOrientation()`—Sets the orientation of the stage. This method has one parameter, which is a string defining the new stage orientation. The constants in the `StageOrientation` class define possible values for the parameter.
- `StageOrientation`—Defines stage orientation values. For example, `StageOrientation.ROTATED_RIGHT` indicates a stage that is rotated right relative to the default orientation of the device.
- `StageOrientationEvent`—Defines events that the Stage dispatches when the orientation of the screen changes. This event occurs when the user rotates the iPhone. There are two types of events. The Stage dispatches the `orientationChanging` event as the device is being rotated. To prevent the stage from reorienting, call the `preventDefault()` method of the `orientationChanging` event object. The Stage dispatches the `orientationChange` event once the stage reorientation is complete.

Currently, the screen orientation API is only useful in AIR applications on mobile devices. If a mobile AIR application and a desktop AIR application share source code, use the `Stage.supportsOrientationChange` property to check if the API is supported.

The following example shows how to respond to the user rotating the device:

```
stage.addEventListener(StageOrientationEvent.ORIENTATION_CHANGE,
    onOrientationChange);

function onOrientationChange(event:StageOrientationEvent):void
{
    switch (event.afterOrientation) {
        case StageOrientation.DEFAULT:
            // re-orient display objects based on
            // the default (right-side up) orientation.
            break;
        case StageOrientation.ROTATED_RIGHT:
            // Re-orient display objects based on
            // right-hand orientation.
            break;
        case StageOrientation.ROTATED_LEFT:
            // Re-orient display objects based on
            // left-hand orientation.
            break;
        case StageOrientation.UPSIDE_DOWN:
            // Re-orient display objects based on
            // upside-down orientation.
            break;
    }
}
```

In this example, in the case of different stage orientations, there are comments instead of functional code.

You can change the orientation of the stage by calling the `setOrientation()` method of the `Stage` object. Setting the orientation is an asynchronous operation. You can check when the orientation is complete by listening for the `orientationChange` event. The following code shows how to set the stage to the right-hand orientation:

```
stage.addEventListener(StageOrientationEvent.ORIENTATION_CHANGE,
    onOrientationChange);
stage.setOrientation(StageOrientation.ROTATED_RIGHT);

function onOrientationChange(event:StageOrientationEvent):void
{
    // Code to handle the new Stage orientation
}
```

As the `Stage` rotates it resizes, and the `Stage` object dispatches a `resize` event. You can resize and reposition display objects on the `Stage` in response to the `resize` event.

NativeApplication.systemIdleMode and SystemIdleMode

The `NativeApplication.systemIdleMode` property lets you prevent the iPhone from going into idle mode. By default, an iPhone goes into idle mode if there is no touch screen interaction for some period. Idle mode can cause the screen to dim. It can also cause the iPhone to go into lock mode. This property can be set to one of two values:

- `SystemIdleMode.NORMAL`—The iPhone follows the normal idle mode behavior.
- `SystemIdleMode.KEEP_AWAKE`—The application attempts to prevent the iPhone from going into idle mode.

This functionality is only supported on mobile devices. It is not supported in AIR applications running on desktop operating systems. In an application running on the desktop, setting the `NativeApplication.systemIdleMode` property has no effect.

The following code shows how to disable the iPhone idle mode:

```
NativeApplication.nativeApplication.systemIdleMode = SystemIdleMode.KEEP_AWAKE;
```

CameraRoll

The `CameraRoll` class lets you add an image to the iPhone camera roll. The `addBitmapData()` method adds an image to the iPhone camera roll. The method has one parameter, `bitmapData`. This parameter is the `BitmapData` object containing the image to add to the camera roll.

`CameraRoll` functionality is only supported on mobile devices. It is not supported in AIR applications running on desktop operating systems. To check at runtime whether your application supports the `CameraRoll` functionality, check the static `CameraRoll.supportsAddBitmapData` property.

After you call the `addBitmapData()` method, the `CameraRoll` object dispatches one of two events:

- `complete`—The operation completed successfully.
- `error`—There was an error. For example, perhaps there is not enough free space on the iPhone to store the image.

The following code adds an image of the stage (a screen capture) to the camera roll:

```
if (CameraRoll.supportsAddBitmapData)
{
    var cameraRoll:CameraRoll = new CameraRoll();
    cameraRoll.addEventListener(ErrorEvent.ERROR, onCrError);
    cameraRoll.addEventListener(Event.COMPLETE, onCrComplete);
    var bitmapData:BitmapData = new BitmapData(stage.stageWidth, stage.stageHeight);
    bitmapData.draw(stage);
    cameraRoll.addBitmapData(bitmapData);
}
else
{
    trace("not supported.");
}

function onCrError(event:ErrorEvent):void
{
    // Notify user.
}

function onCrComplete(event:Event):void
{
    // Notify user.
}
```

DisplayObject.cacheAsBitmapMatrix

The `cacheAsBitmapMatrix` property is a `Matrix` object that defines how a display object is rendered when `cacheAsBitmap` is set to `true`. The application uses this matrix as a transformation matrix when rendering the bitmap version of the display object.

With `cacheAsBitmapMatrix` set, the application retains a cached bitmap image rendered using that matrix, instead of the display matrix. (The display matrix is the value of the `transform.concatenatedMatrix` of the display object.) If this matrix does not match the display matrix, the bitmap is scaled and rotated as necessary.

A display object with `cacheAsBitmapMatrix` set is only rerendered when the value of `cacheAsBitmapMatrix` changes. The bitmap is scaled or rotated as appropriate to conform to the display matrix.

Both CPU- and GPU-based rendering benefit from the use of the `cacheAsBitmapMatrix` property, although GPU rendering typically benefits more.

Note: To use the hardware acceleration, set the *Rendering to GPU* in the *General* tab of the *iPhone Settings* dialog box in *Flash Professional CS5*. (Or set the `renderMode` property to `gpu` in the application descriptor file.)

For example, the following code uses an untransformed bitmap representation of the display object:

```
matrix:Matrix = new Matrix(); // creates an identity matrix
mySprite.cacheAsBitmapMatrix = matrix;
mySprite.cacheAsBitmap = true;
```

The following code uses a bitmap representation that matches the current rendering:

```
mySprite.cacheAsBitmapMatrix = mySprite.transform.concatenatedMatrix;
mySprite.cacheAsBitmap = true;
```

Usually, the identity matrix (`new Matrix()`) or `transform.concatenatedMatrix` suffices. However, you can use another matrix, such as a scaled-down matrix, to upload a different bitmap to the GPU. For example, the following example applies a `cacheAsBitmapMatrix` matrix that is scaled by 0.5 on the x- and y-axes. The bitmap object that the GPU uses is smaller, however the GPU adjusts its size to match the `transform.matrix` property of the display object.:

```
matrix:Matrix = new Matrix(); // creates an identity matrix
matrix.scale(0.5, 0.5); // scales the matrix
mySprite.cacheAsBitmapMatrix = matrix;
mySprite.cacheAsBitmap = true;
```

Generally, choose a matrix that transforms the display object to the size that it will appear in the application. For example, if your application displays the bitmap version of the sprite scaled down by a half, use a matrix that scales down by a half. If your application will display the sprite larger than its current dimensions, use a matrix that scales up by that factor.

There is a practical limit to the size of display objects for which the `cacheAsBitmapMatrix` property is set. The limit is 1020 by 1020 pixels. There is a practical limit for the total number of pixels for all display objects for which the `cacheAsBitmapMatrix` property is set. That limit is about four million pixels.

There are many considerations when using `cacheAsBitmapMatrix` and hardware acceleration. It is important to know which display objects should have that property set, and which ones should not. For important information on using this property, see [“Hardware acceleration”](#) on page 33.

You can use the GPU rendering diagnostics feature to diagnose GPU usage in debug builds of your application. For more information, see [“Debugging an iPhone application”](#) on page 23.

Networking notes

Using the following URL schemes with the `navigateToURL()` function causes a document to open in an external application:

| URL scheme | Result of call to nativeToURL() | Example |
|------------|---|---|
| mailto: | Opens a new message in the mail application. | <pre>str = "mailto:test@example.com"; var urlReq:URLRequest = new URLRequest(str); navigateToURL(urlReq);</pre> |
| sms: | Opens a message in the text message application. | <pre>str = "sms:1-415-555-1212"; var urlReq:URLRequest = new URLRequest(str); navigateToURL(urlReq);</pre> |
| tel: | Dials a phone number on the phone (with user approval). | <pre>str = "tel:1-415-555-1212"; var urlReq:URLRequest = new URLRequest(str); navigateToURL(urlReq);</pre> |

An iPhone application may rely on installed self-signed root certificates for server authentication during a secure transaction, such as an https request. A server should send not just the leaf certificate but also all intermediate certificates chaining to the root certificate.

ActionScript 3.0 APIs of special interest to mobile application developers

The following ActionScript 3.0 APIs define functionality that is useful on mobile devices.

Accelerometer API

The following classes allow the application get events from the device's accelerometer:

- Accelerometer
- AccelerometerEvent

For more information, see [Accelerometer input](#).

Geolocation API

The following classes allow the application get events from the device's location sensor:

- Geolocation
- GeolocationEvent

For more information, see [Geolocation](#).

Touch, multi-touch, and gesture API

The following classes allow the application to receive touch and gesture events:

- GestureEvent
- GesturePhase
- MultiTouch
- MultitouchInputMode
- TouchEvent
- TransformGestureEvent

For more information, see [Touch, multi-touch and gesture input](#).

Chapter 4: iPhone application design considerations

The processing speed and screen size of the iPhone 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](#). 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 the iPhone.

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

Hardware acceleration

You can use OpenGL ES 1.1 hardware acceleration to enhance graphics performance in some applications. Games and other applications in which display objects are animated can benefit from hardware acceleration. Applications that use hardware acceleration can off-load some graphics processes to from the CPU to the iPhone GPU, which can greatly improve performance.

When designing an application to use the GPU it is important to follow rules that ensure the content is GPU-accelerated effectively.

To use the hardware acceleration, set the Rendering to GPU in the General tab of the iPhone Settings dialog box in Flash Professional CS5. You can also set the `renderMode` property to `gpu` in the application descriptor file:

```
<initialWindow>
  <renderMode>gpu</renderMode>
  ...
```

See “[Setting iPhone application properties in Flash Professional CS5](#)” on page 14 and “[Setting iPhone application properties in the application descriptor file](#)” on page 16.

There are four classes of display objects that can render quickly in hardware acceleration if their content changes infrequently:

- Bitmap objects
- 2D Display objects for which the `cacheAsBitmap` property is set to `true`, and optionally have their `cacheAsBitmapMatrix` property set (see below)
- 3D Display objects (i.e. have their `z` property set)
- Display objects that have a single solid-color rectangular fill and that have edges that align with the pixels on the screen.

Vector-based objects re-render whenever another sprite animates over or under them. So, any object serving as a backdrop or foreground to an animation should also belong to one of these categories.

For display objects that have `cacheAsBitmap` set to `true`, setting `cacheAsBitmapMatrix` causes the GPU to use the bitmap that results from the matrix transformation. The GPU uses the bitmap representation even if the object is rotated or scaled. The GPU can composite and animate this bitmap much more quickly than the CPU can redraw a vector-rendered object.

Setting `cacheAsBitmap` to `true` alone causes a display object (and any children) to be cached. The display object is not redrawn when new regions are exposed, or the whole combined graphic is translated.

When a display object's `cacheAsBitmapMatrix` property is set, the application can build a representation of the display object even when it is not visible. The application builds a cached representation of the display object at the start of the next frame. Then when you add the display object to the stage, the application renders it quickly. Also, the application can quickly animate rotate or scale the object. For objects that rotate or scale, do not set the `cacheAsBitmap` property without also setting the `cacheAsBitmapMatrix` property.

The application can also quickly perform alpha transformations on an object that is cached as a bitmap. However only alpha values between 0 and 1.0 are supported for hardware-accelerated alpha transformations. That corresponds to a `colorTransform.alphaMultiplier` setting between 0 to 256.

Do not set the `cacheAsBitmap` property to `true` for frequently updated objects, such as text fields.

Display objects that have frequently changing graphical content are generally poor candidates for GPU rendering. This is especially true on earlier devices with less powerful GPUs. The overhead of uploading the graphics to the GPU can make CPU rendering a better choice.

Restructure display objects that contain child display objects that move relative to the parent. Change them so that the child display objects become siblings of the parent. This ensures they will each have their own bitmap representation. Also, each display object will be able to move relative to the others without any need for new graphics to be uploaded to the GPU.

Set the `cacheAsBitmap` property to `true` at the highest level of the display list where child display objects do not animate. In other words, set it for display object containers that contain no moving parts. Do not set it on the child display objects. Do *not* set it for sprites containing other display objects that animate.

When you set the `z` property of a display object, the application always uses a cached bitmap representation. Also, after you set the `z` property of a display object, the application uses the cached bitmap representation, even if you rotate or scale the object. The application does not use the `cacheAsBitmapMatrix` property for display objects that have a set `z` property. The same rules apply when you set any three-dimensional display object properties, including the `rotationX`, `rotationY`, `rotationZ`, and `transform.matrix3D` properties.

Do not set the `scrollRect` or `mask` property of a display object container containing content for which you want to use hardware acceleration. Setting these properties disables hardware acceleration for the display object container and its child objects. As an alternative to setting the `mask` property, layer a mask display object over the display object being masked.

There are limits to the size of display objects available for hardware acceleration. On older devices, the limit is 1024 pixels or less in both width and height. On newer devices, the limit is 2048 pixels or less. You can use the GPU rendering diagnostic tool to test performance on a device.

The GPU also uses iPhone RAM to store bitmap images. It uses as least as much memory as is needed for the bitmap images.

The GPU uses memory allocations that are powers of 2 for each dimension of the bitmap image. For example, the GPU can reserve memory in 512 x 1024 or 8 x 32 sizes. So a 9 x 15-pixel image takes up the same amount of memory as a 16 x 16-pixel image. For cached display objects, you may want to use dimensions that are close to powers of 2 (but not more) in each direction. For example, it is more efficient to use a 32 x 16-pixel display object than a 33 x 17-pixel display object.

Do not rely on a resized Stage to scale down assets that were sized for other platforms (such as the desktop). Rather, use the `cacheAsBitmapMatrix` property, or resize the assets before publishing for the iPhone. 3D objects ignore the `cacheAsBitmapMatrix` property when caching a surface image. For this reason, it is better to resize display objects before publishing if they will be rendered on a 3D surface.

There is a trade-off between the benefits of hardware acceleration and RAM usage. As memory fills up, iPhone OS notifies other running, native iPhone applications to free up memory. As these applications process this notification and work to free memory, they may compete with your application for CPU cycles. This can momentarily degrade the performance of your application. Be sure to test your application on older devices, as they may have substantially less memory available for your running process.

When debugging the application on the iPhone, you can enable the GPU rendering diagnostics feature. This feature aids you in seeing how your application uses GPU rendering. For more information, see “Debugging with GPU rendering diagnostics” in “[Debugging an iPhone application](#)” on page 23.

For information on using the `cacheAsBitmapMatrix` property, see the “`DisplayObject.cacheAsBitmapMatrix`” section in “[ActionScript APIs specific to mobile AIR applications](#)” on page 28.

Other ways to improve display object performance

Hardware acceleration 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.

- 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. This is particularly true for display objects that are not hardware accelerated.
- 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. This is particularly true for display objects that are not hardware accelerated.
- Make bitmaps in sizes that are close to, but less than, 2^n by 2^m 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.) Such images also use memory more efficiently.
- Limit the size of display objects to 1024 x 1024 pixels (or 2048 x 2048 on newer devices).

Information density

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 display 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

For best appearance, use device fonts. For example, the following fonts are device fonts on the iPhone:

- Serif: Times New Roman, Georgia, and `_serif`
- Sans-serif: Helvetica, Arial, Verdana, Trebuchet, Tahoma, and `_sans`
- Fixed-width: Courier New, Courier, and `_typewriter`

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.

Do not use underlined text for input text fields. Also, do not set the alignment of the text field. Input text fields on the iPhone only support left alignment (the default).

If you use TLF Text setting for a text field in Flash Professional CS5, turn off runtime shared library in the default linkage in ActionScript 3.0 Settings. Otherwise the application will not work on the iPhone because it would try to use the runtime shared library SWF file:

- 1 Select File > Publish Settings.
- 2 In the Publish Settings dialog box, click the Flash tab.
- 3 Click the Script button the right of the Script (ActionScript 3.0) drop-down list.
- 4 Click the Library Path tab.
- 5 In the Default Linkage drop-down list, select Merged Into Code.

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.

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.

When the user clicks a text field to edit it (and the virtual keyboard is displayed), the `TextField` object dispatches a `focusIn` event. You can add an event listener for this event 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 Done key on the keyboard.

After editing text in a multi-line text field, the user dismisses the virtual keyboard by tapping outside the text field. This removes focus from the text field. Make sure that your design includes area outside 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 the iPhone, the virtual keyboard is only present when the user edits a text field. An iPhone application only dispatches keyboard events when the virtual keyboard is present.

Saving application state

Your application may quit at any time (for example when the phone rings). Consider saving the state of your application anytime it changes. 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 it relaunches. If a phone call interrupts an application, it will restart when the call ends.

Do not rely on the `NativeApplication` object dispatching an `exitting` event when the application quits; it may not.

Screen orientation changes

iPhone content can be viewed in portrait or landscape orientation. Consider how your application will deal with screen orientation changes. For more information, see [Detecting device 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.

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](#).

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 video is 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.

Flex and Flash components

Many Flex components and Flash components are designed for use in desktop applications. These components, especially display components, may perform slowly on mobile devices. In addition to performing slowly, desktop components may have inappropriate interaction models for mobile devices.

Adobe is developing a mobile-optimized version of the Flex framework. For more information, see <http://labs.adobe.com/technologies/flex/mobile/>.

Reducing application file size

Here are some tips on reducing the file size of your IPA 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.