iPhone Development Guide

Development Environments: Xcode

2009-01-06

Contents

Figures, Tables, and Listings

Introduction

To develop iPhone applications, you use Xcode, Apple's first-class integrated development environment (IDE). Xcode provides all the tools you need to design your application's user interface and write the code that brings it to life. As you develop your application, you run it on your computer, an iPhone, or an iPod touch.

This document describes the iPhone application development process. It also provides information about becoming a member of the iPhone Developer Program, which is required to run applications on devices for testing.

After you finish developing your iPhone application, you submit it to the App Store, the secure marketplace where iPhone OS users obtain their applications. However, you should test your application on a small set of users before publishing it to cover a wide variety of usage patterns and get feedback about your product. This document describes how to create a group of testers for your application and how to distribute it to them.

To take advantage of this document, you should be familiar with the iPhone application architecture, described in *iPhone Application Programming Guide*. You should also be familiar with basic programming concepts.

After reading this document, you'll have a basic understanding of the iPhone application development process. To enhance that knowledge, you should read the documents listed later in this introduction.

Organization of This Document

This document contains the following chapters:

- "The Development Process" (page 11) provides an overview of the major development tasks you follow to design, build, and run an application using Xcode.
- "Tutorial: Hello, World" (page 19) guides you through the creation of a simple project, Hello World, that prints text on the iPhone screen.
- "Running Applications" (page 25) describes each of the steps required to run or debug your iPhone applications.
- "Using iPhone Simulator" (page 31) describes the ways in which you use your computer's input devices to simulate the interaction between users and their devices.
- "Managing Devices" (page 35) shows how to configure your computer and your device for development; how to use the Xcode Organizer window to view console logs or crash information, and take screen shots of applications running on your device; and how to safeguard the digital identifications required to install applications in development on devices.
- "Debugging Applications" (page 45) describes the Xcode debugging facilities.
- "Tuning Applications" (page 49) describes Instruments and Shark, the tools you use to measure and tune your application's performance.

- "Publishing Applications for Testing" (page 51) describes the steps you need to perform to add testers to your team and shows how to add symbol information to their crash logs, also known as symbolicating.
- "Conditional Compilation and Linkage" (page 55) shows how to target code to iPhone Simulator or a device and how to link to frameworks or libraries depending on that choice.
- "iPhone Simulator FAQ" (page 59) lists common questions developers ask about iPhone Simulator.
- "Instructions for Application Testers" (page 61) provides instructions to testers about the procedures to follow to test your iPhone applications on their devices.

See Also

These documents describe the essential concepts you need to know about developing iPhone applications:

- *iPhone OS Technology Overview* introduces iPhone OS and its technologies.
- *Cocoa Fundamentals Guide* introduces the basic concepts, terminology, architectures, and design patterns of the Cocoa frameworks and development environment.
- *The Objective-C 2.0 Programming Language* introduces object-oriented programming and describes the main programming language used for iPhone development.

The Development Process

Developing iPhone applications is a pleasant and rewarding endeavor. To convert your ideas into products you use Xcode, the integrated development environment (IDE) used to develop iPhone applications. With Xcode you organize and edit your source files, view documentation, build your application, debug your code, and optimize your application's performance.

This chapter provides an overview of the major development tasks you follow to design, build, and run an application using Xcode.

Essential Development Tasks

The iPhone-application development process is divided into these major steps:

1. Create your project.

   Xcode provides several project templates that get you started. You choose the template that implements the type of application you want to develop. See "Creating an iPhone Project" (page 12) for details.

2. Design the user interface.

   The **Interface Builder application** lets you design your application's user interface graphically and save those designs as resource files that you load into your application at runtime. If you do not want to use Interface Builder, you can layout your user interface programmatically. See "User Interface Design Considerations" in *iPhone Application Programming Guide* for more information.

3. Write code.

   Xcode provides several features that help you write code fast, including class and data modeling, code completion, direct access to documentation, and refactoring. See "Editing Code" (page 13) for details.

4. Build and run your application.

   You build your application on your computer and run it in the iPhone Simulator application or on your device. See "Building and Running Your Application" (page 17) for more information.

5. Measure and tune application performance.

   After you have a running application, you should measure its performance to ensure that it uses a device's resources as efficiently as possible and that it provides adequate responses to the user's gestures. See "Measuring Application Performance" (page 18) for more information.

The rest of this chapter gives more details about these steps.

Creating an iPhone Project

Xcode provides several iPhone-application project templates to get you up and running developing your application. You can choose from these types of application:

- **Navigation-Based Application.** An application that presents data hierarchically, using multiple screens. The Contacts application is an example of a navigation-based application.
- **OpenGL ES Application.** An application that uses an OpenGL ES–based view to present images or animation.
- **Tab Bar Application.** An application that presents a radio interface that lets the user choose from several screens. The Clock application is an example of a tab bar application.
- **View-Based Application.** An application that uses a single view to implement its user interface.
- **Utility Application.** An application that implements a main view and lets the user access a flipside view to perform simple customizations. The Stocks application is an example of a utility application.
- **Window-Based Application.** This template serves as a starting point for any application, containing an application delegate and a window. Use this template when you want to implement your own view hierarchy.

If you need to develop a static library for use in an iPhone application, you can add a static library target to your project by choosing Project > New Target and selecting the Static Library target template in the iPhone OS/Cocoa Touch list.

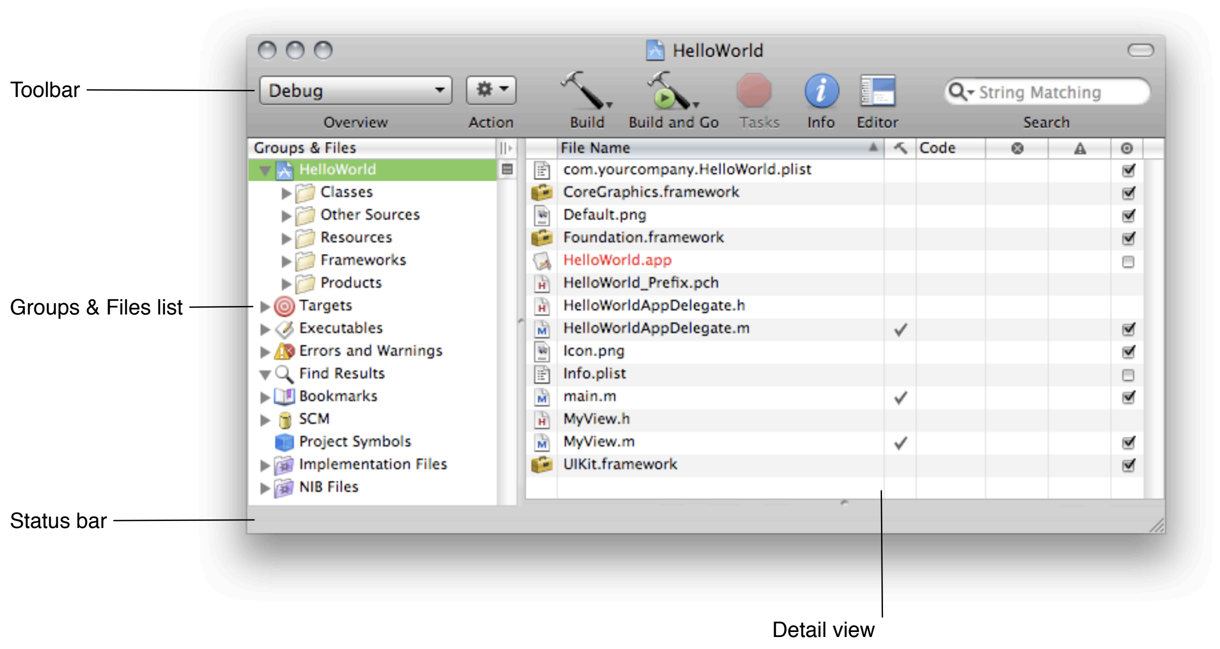
Static libraries used in iPhone applications do not need to be code signed. Therefore, you should remove the Code Signing Identity build setting definition from the static library targets you create. To do so:

1. Open the static library target's Info window and display the Build pane.
2. In the Code Signing category, select the Any iPhone OS Device conditional definition for the Code Signing Identity build setting.
3. Change the conditional definition's value from iPhone Developer to Don't Code Sign.

To learn more about the iPhone application architecture, see *iPhone Application Programming Guide*.

To develop an iPhone application, you work on an Xcode project. And you do most of your work on projects through the **project window**, which displays and organizes your source files and other resources needed to build your application. It allows you to access and edit all the pieces of your project. Figure 1-1 shows the project window.

Figure 1-1 Project window



The project window contains the following key areas for navigating your project:

- **Groups & Files list.** Provides an outline view of your project contents. You can move files and folders around and organize your project contents in this list. The current selection in the Groups & Files list controls the contents displayed in the detail view.
- **Detail view.** Shows the item or items selected in the Groups & Files list. You can browse your project's contents in the detail view, search them using the Search field, or sort them according to column. The detail view helps you rapidly find and access your project's contents.
- **Toolbar.** Provides quick access to the most common Xcode commands.
- **Status bar.** Displays status messages for the project. During an operation—such as building or indexing—Xcode displays a progress indicator in the status bar to show the progress of the current task.
- **Favorites bar.** Lets you store and quickly return to commonly accessed locations in your project. The favorites bar is not displayed by default. To display the favorites bar, choose View > Layout > Show Favorites Bar.

To learn more about creating projects, see "Creating Projects".

Editing Code

The main tool you use to write your code is the Xcode text editor. This advanced text editor provides several convenient features:

- **Header-file lookup.** By Command–double-clicking a symbol, you can view the header file that declares the symbol.
- **API reference lookup.** By Option–double-clicking a symbol, you get access to API reference that provides information about the symbol's usage.
- **Code completion.** As you type code, you can have the editor help out by inserting text for you that completes the name of the symbol Xcode thinks you're going to enter. Xcode does this in an unobtrusive and overridable manner.
- **Code folding.** With code folding, you can collapse code that you're not working on and display only the code that requires your attention.

For details about these and other text editor features, see "The Text Editor".

Using Code Completion

The text editor helps you type code faster with code completion. When code completion is active, Xcode uses both text you have typed and the context into which you have typed it to provide suggestions for completing the token it thinks you intend to type. Code completion is not active by default.

To activate code completion:

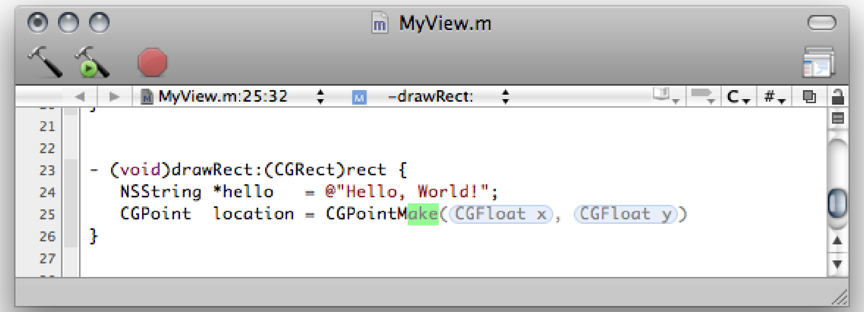
1. Open the Xcode Preferences window.

   Choose Xcode > Preferences (or press Command–,).

2. In the Code Completion section of the Code Sense pane, choose Immediate from the Automatically Suggest pop-up menu.
3. Click OK.

As you type the name of a symbol, Xcode recognizes that symbol and offers a suggestion, as shown in Figure 1-2. You can accept suggestions by pressing Tab or Return. You may also display a list of completions by pressing Escape.

Figure 1-2 Using code completion



```
- (void)drawRect:(CGRect)rect {
    NSString *hello    = @"Hello, World!";
    CGPoint  location = CGPointMake(CGFloat x, CGFloat y)
}
```

To learn more about code completion, see “Completing Code” in *Xcode Workspace Guide*.

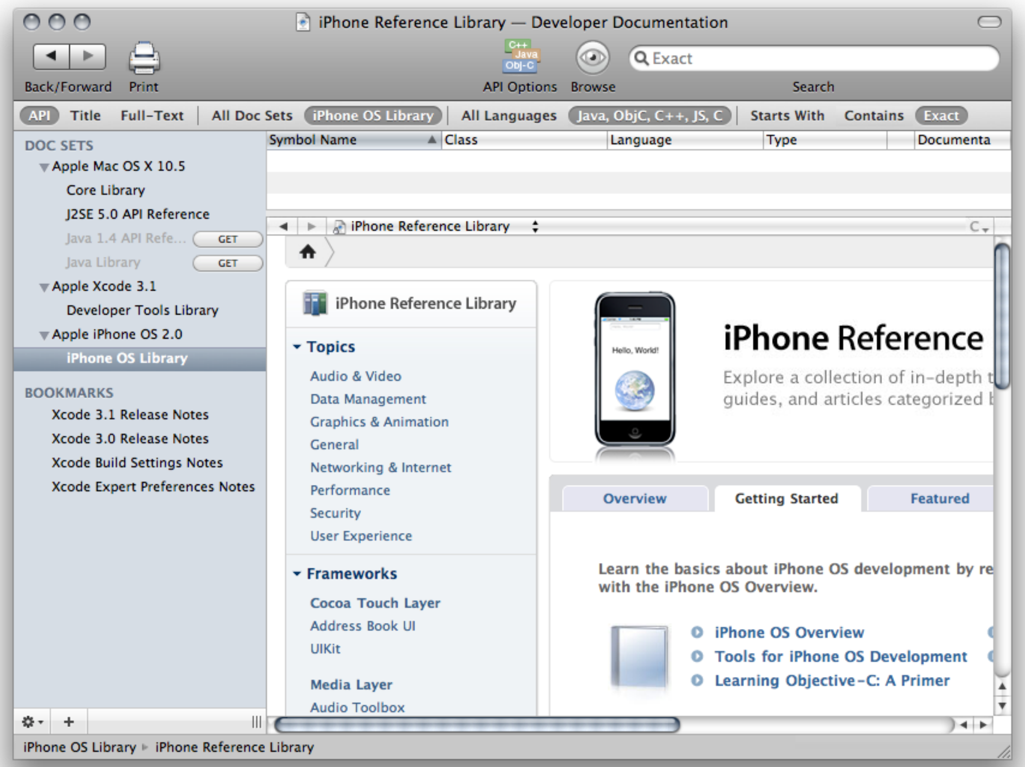
Accessing Documentation

During development, you may need fast access to reference for a particular symbol or high-level documentation about API usage or an iPhone OS technology. Xcode gives you easy access to such resources through the Research Assistant and the Documentation window.

The **Research Assistant** is a lightweight window, shown in Figure 1-5 (page 17), that provides a condensed view of the API reference for the selected item, without taking your focus away from the editor in which the item is located. This window provides an unobtrusive way to consult API reference. However, when you need to delve deeper into the reference, the Documentation window is just a click away.

The **Documentation window** (Figure 1-3) lets you browse and search the developer documentation (which includes API reference, guides, and article collections about particular tools or technologies) installed on your computer. It provides access to a wider and more detailed view of the documentation than the Research Assistant, for the times when you need additional help.

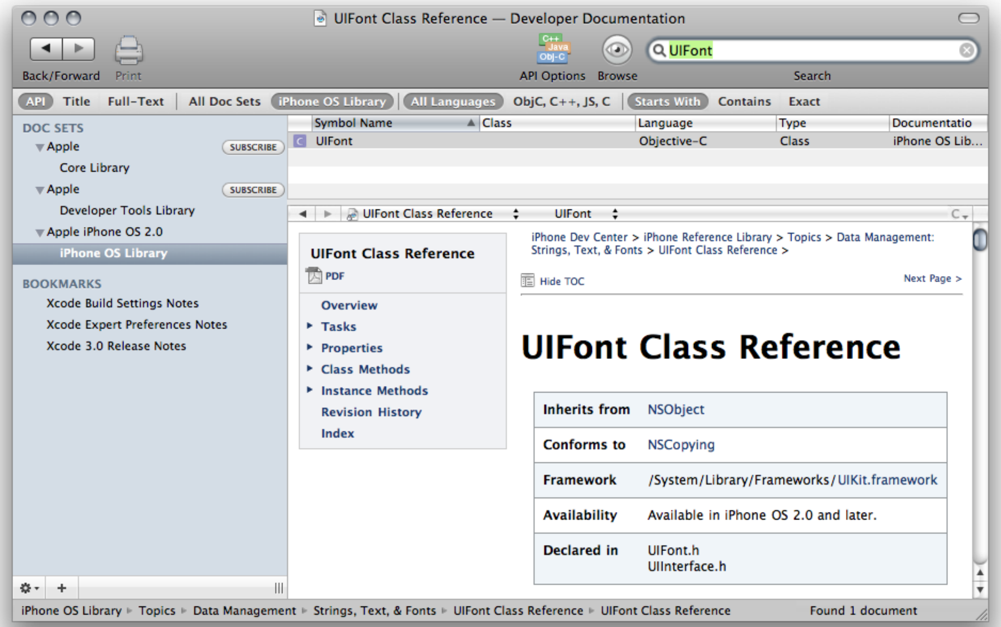
Figure 1-3 The Documentation window



To display the API reference for a symbol in a source file, you select the symbol in the text editor and choose Help > Find Selected Text in API Reference (you can also Option–double-click the symbol name). This command searches for the selected symbol in the API reference for your project’s SDK and displays it in the

Documentation window. For example, if you select the `UIFont` class name in a source file and execute the Find Selected Text in API Reference command, Xcode displays the Documentation window with the API reference for the `UIFont` class, as shown in Figure 1-4.

Figure 1-4 Viewing API reference in the Documentation window



Symbol Name	Class	Language	Type	Documentatio
UIFont		Objective-C	Class	iPhone OS Lib...

UIFont Class Reference

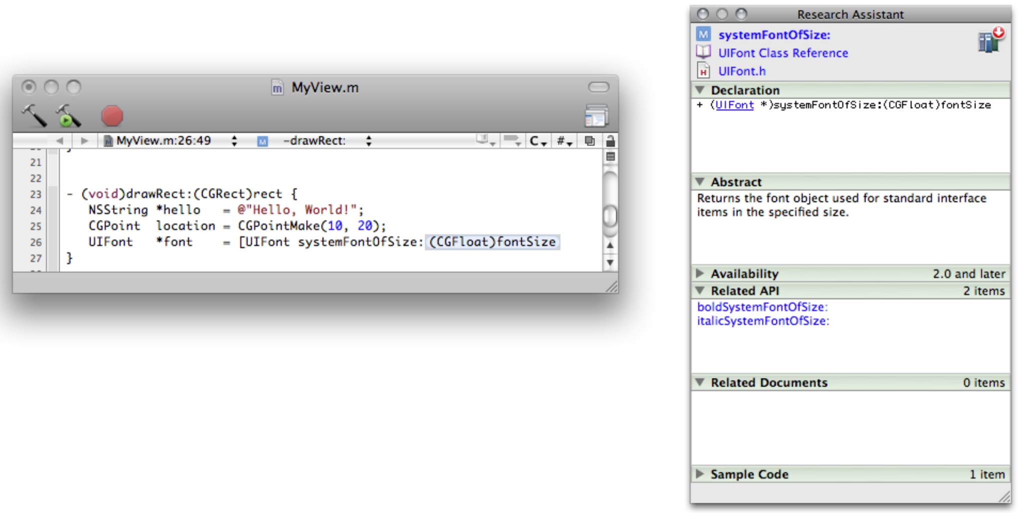
Inherits from	NSObject
Conforms to	NSCopying
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iPhone OS 2.0 and later.
Declared in	UIFont.h UIInterface.h

While the Documentation window is a great tool to browse the iPhone documentation library, sometimes you may not want to take your focus away from the text editor while you write code, but need basic information about a symbol in a condensed way. The Research Assistant provides such information in a small and unobtrusive window.

The Research Assistant actively follows you as you move the cursor around a source file. When it recognizes a symbol for which it finds API reference, the Research Assistant displays that reference, as shown in Figure 1-5. All you have to do is glance at the Research Assistant to get essential details about the symbol.

To display the Research Assistant, choose Help > Show Research Assistant.

Figure 1-5 Viewing API reference in the Research Assistant



```
- (void)drawRect:(CGRect)rect {
   NSString *hello   = @"Hello, World!";
   CGPoint  location = CGPointMake(10, 20);
   UIFont   *font    = [UIFont systemFontOfSize:(CGFloat)fontSize
}
```

systemFontOfSize:
UIFont Class Reference
UIFont.h

Declaration

```
+ (UIFont *)systemFontOfSize:(CGFloat)fontSize
```

Abstract

Returns the font object used for standard interface items in the specified size.

Availability 2.0 and later

Related API 2 items

boldSystemFontOfSize:
italicSystemFontOfSize:

Related Documents 0 items

Sample Code 1 item

From the Research Assistant you can quickly jump to more comprehensive reference for the symbol, or even view the header that declares it.

For more information about accessing documentation in Xcode, see Documentation Access.

Building and Running Your Application

iPhone Simulator implements the iPhone OS API, providing an environment that closely resembles the environment devices provide. It allows you to run your applications in Mac OS X, letting you quickly test your application's functionality when you don't have a device available. However, running applications in iPhone Simulator is not the same as running them in actual devices. iPhone Simulator does not emulate device performance: It doesn't implement the memory constraints or processor performance of an actual device. To get an accurate idea of how your application will perform on a user's device, you must run the application on a device and gather performance data using Instruments and other performance-measuring tools.

First, the simulator uses Mac OS X versions of the low-level system frameworks instead of the versions that run on the devices. Secondly, there may be hardware-based functionality that's unavailable on the simulator. But, in general, the simulator is a great tool to perform initial testing of your applications.

To compile and debug your code, Xcode relies on open-source tools, such as GCC and GDB. Xcode also supports team-based development with source control systems, such as Subversion, CVS, and Perforce.

Building your application involves the following steps:

- Compiling your source files and generating your application binary.
- Placing the binary in iPhone Simulator or on your device.

Xcode performs these tasks for you when you execute the Build command. See "Running Applications" (page 25) for details.

Measuring Application Performance

After you have tested your application's functionality, you must ensure that it performs well on a device. This means that the application uses the device's resources as efficiently as possible. For example, memory is a scarce resource; therefore, your application should maintain a small memory footprint not to impair the performance of iPhone OS. Your application should also use efficient algorithms to consume as little power as possible not to reduce battery life. Xcode provides two major tools to measure and tune application performance: Instruments and Shark.

The **Instruments application** is a dynamic performance analysis tool that lets you peer into your code as it's running and gather important metrics about what it is doing. You can view and analyze the data Instruments collects in real time, or you can save that data and analyze it later. You can collect data about your application's use of the CPU, memory, the file system, and the network, among other resources.

The **Shark application** is another tool that helps you find performance bottlenecks in your code. It produces profiles of hardware and software performance events and shows how your code works as a whole and its interaction with iPhone OS.

With Instruments and Shark, you can find and eliminate performance bottlenecks in your code. For more information, see "Tuning Applications" (page 49).

Further Exploration

To learn more about the Xcode development process, see *Xcode Project Management Guide*.

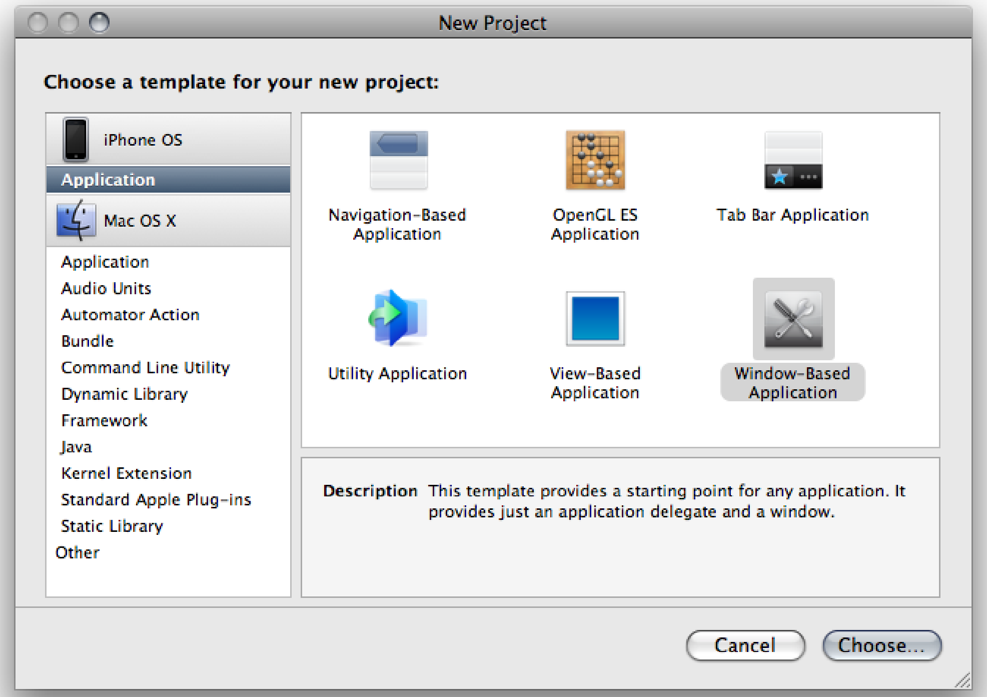
Tutorial: Hello, World!

This chapter guides you through the creation of a simple project that prints text on the iPhone screen.

Create the Project

To create the Hello World project, follow these steps:

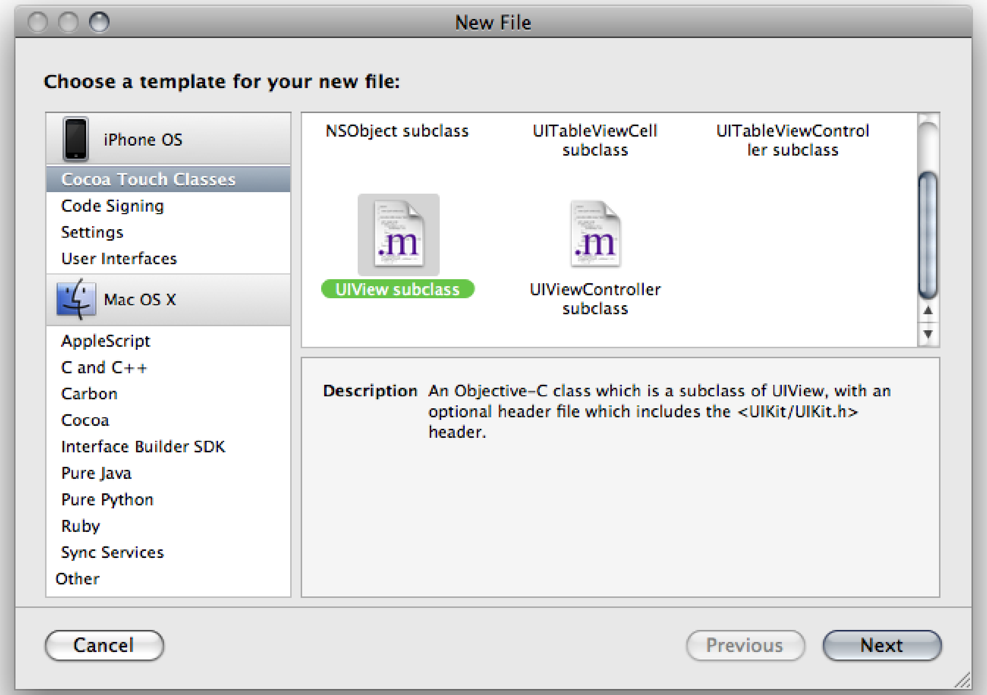
1. Launch the Xcode application, located in `<Xcode>/Applications`.
2. Choose File > New Project.
3. Select the Window-Based Application template and click Choose.



Description This template provides a starting point for any application. It provides just an application delegate and a window.

4. Name the project `HelloWorld` and choose a location for it in your file system.
5. Add the `MyView` class to the project.
   a. Choose File > New File.

b. Select the Cocoa Touch UIView subclass template and click Next.



Description An Objective-C class which is a subclass of UIView, with an optional header file which includes the <UIKit/UIKit.h> header.

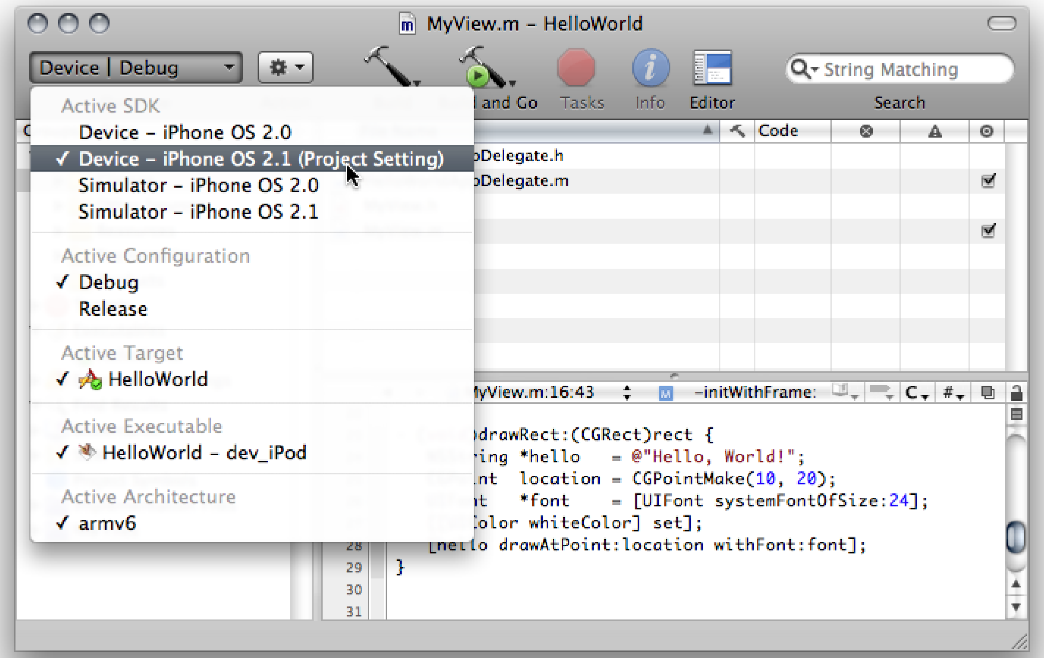
c. In the File Name text field, enter `MyView.m`.

d. Select the “Also create "MyView.h"” option and click Finish.

6. Choose the active SDK for your project.

If you have a development device plugged in at the time you create the project, Xcode sets the active SDK to build for your device. Otherwise, it sets it to build for iPhone Simulator.

To set the active SDK, chose an item from the Project > Set Active SDK submenu or the Overview toolbar menu in the project window.



```
drawRect:(CGRect)rect {
ing *hello    = @"Hello, World!";
nt  location = CGPointMake(10, 20);
t   *font     = [UIFont systemFontOfSize:24];
olor whiteColor] set];
[hello drawAtPoint:location withFont:font];
}
```

Write the Code

In Xcode, the text editor is where you spend most of your time. You can write code, build your application, and debug your code. Let's see how Xcode assists you in the first task.

First, modify the `HelloWorldAppDelegate` class to use the `MyView` class:

1. In the Groups & Files list, select the HelloWorld group.
2. In the detail view, double-click `HelloWorldAppDelegate.m`.
3. In the `HelloWorldAppDelegate` editor window:

   a. Add the following code line below the existing `#import` line.

   ```
   #import "MyView.h"
   ```

   b. Add the following code lines to the `applicationDidFinishLaunching:` method, below the override-point comment.

   ```
   MyView *view = [[MyView alloc] initWithFrame:[window frame]];
   [window addSubview:view];
   ```

```
       [view release];
```

After making these changes, the code in the `HelloWorldAppDelegate.m` file should look like this:

```
#import "HelloWorldAppDelegate.h"
#import "MyView.h"

@implementation HelloWorldAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

   // Override point for customization after app launch
   MyView *view = [[MyView alloc] initWithFrame:[window frame]];
   [window addSubview:view];
   [view release];

   [window makeKeyAndVisible];
}

- (void)dealloc {
    [window release];
    [super dealloc];
}

@end
```

Listing 2-1 shows the code that draws “Hello, World!” in the window. Add the highlighted code lines to the `drawRect:` method in the `MyView.m` file.

Listing 2-1 Method to draw “Hello, World!” in a view

```
- (void) drawRect:(CGRect) rect {
    NSString *hello   = @"Hello, World!";
    CGPoint  location = CGPointMake(10, 20);
    UIFont   *font    = [UIFont systemFontOfSize:24];
    [[UIColor whiteColor] set];
    [hello drawAtPoint:location withFont:font];
}
```

If you turned on code completion (as described in “Using Code Completion” (page 14)), as you type symbol names the text editor suggests completions for the symbol names it recognizes. For example, as you type `CGPointM`, the text editor suggests the completion shown in Figure 1-2 (page 14). You can take advantage of completion here by accepting the suggested completion and jumping to the parameter placeholders:

1. Jump to the first parameter by choosing Edit > Select Next Placeholder (or press Control–/), and type `10`.

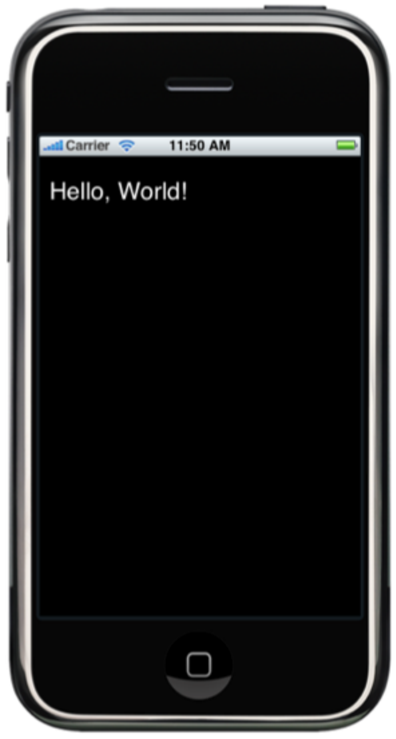
   The Select Next Placeholder command moves you among the arguments in function or method calls that the text editor suggests as completions to the text you're typing.

2. Jump to the second parameter and type `20`.

3. Enter the semicolon (`;`) at the end of the line and press Return.

Run the Application

To build and run the Hello World application, choose Build > Build and Run (or click the Build and Go toolbar item in the project window). If there are no build errors, Xcode installs the application in iPhone Simulator or your device (depending on the active SDK setting).



Further Exploration

Now that you learned how to write the standard Hello, World! application for iPhone OS, you can experiment with *HelloWorld*, the Cocoa Touch version of this ubiquitous application.

For a step-by-step tutorial in developing a more complex application, see *Your First iPhone Application*.

Running Applications

When you're ready to run or debug your application, you build it using the Xcode build system. If there are no build errors, you can run it in iPhone Simulator or on a device.

The iPhone SDK comprises two SDK families: The iPhone Simulator SDK and the iPhone Device SDK.

- **iPhone Simulator SDK:** These SDKs build applications that run in the iPhone Simulator.
- **iPhone Device SDK:** These SDKs build applications that run in a device.

These are steps you follow to build and run applications:

1. Set the application's target iPhone OS release.
2. Set the active SDK.
3. Set the active build configuration.
4. Build the application.
5. Run the application.

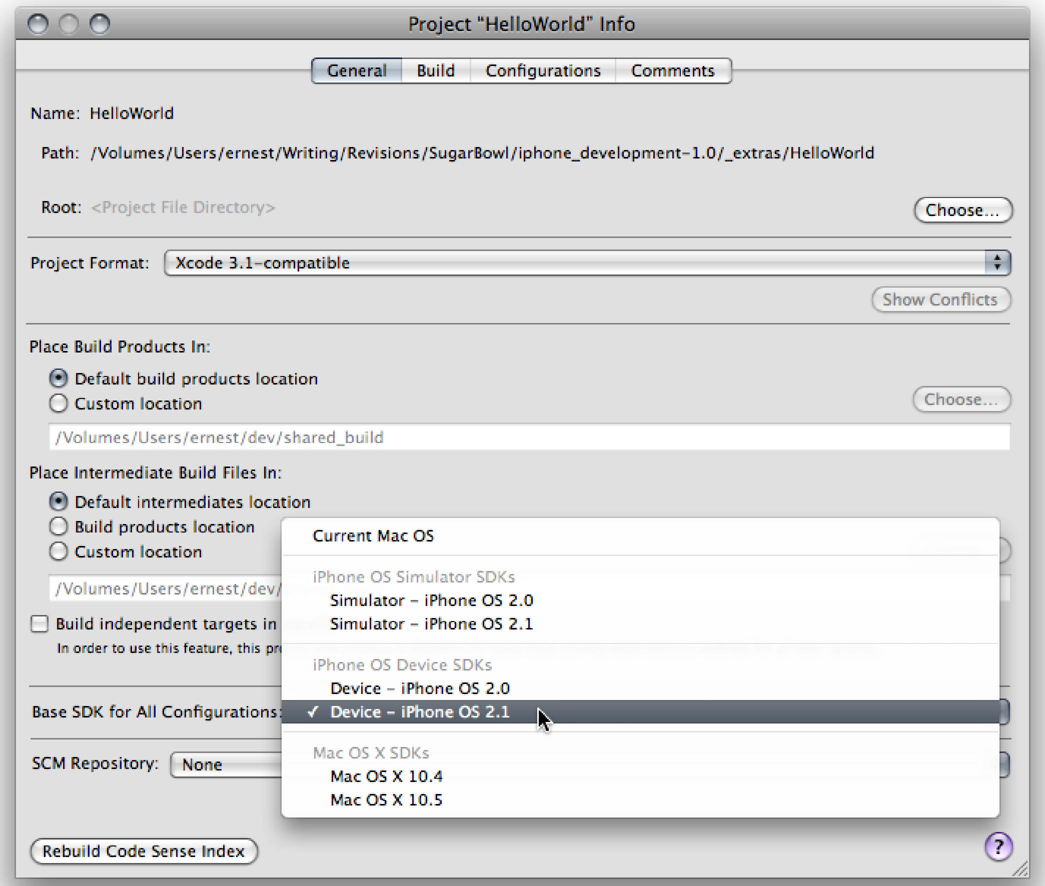
This chapter describes each of the steps required to run or debug your application.

Setting Your Application's Target iPhone OS Release

Each release of iPhone OS (and its corresponding iPhone SDK) includes features and capabilities not present in earlier releases. As new releases of the OS are published, some users may upgrade immediately while other users may wait before moving to the latest release. You can take one of two strategies, depending on the needs of your application and your user base:

- **Target the latest iPhone OS release.** Targeting the latest release allows you to take advantage of all the features available in the latest version of iPhone OS. However, this approach may offer a smaller set of users capable of installing your application on their devices because your application cannot run on iPhone OS releases that are earlier than the target release.
- **Target an earlier iPhone OS release.** Targeting a noncurrent release lets you publish your application to a wide user base (because your application runs on the target OS release and later releases), but may limit the iPhone OS features your application can use.

You specify the default target iPhone OS release for your project by choosing the corresponding SDK from the Base SDK for All Configurations pop-up menu in the General pane of the project Info window.



Name: HelloWorld

Path: /Volumes/Users/ernest/Writing/Revisions/SugarBowl/iphone_development-1.0/_extras/HelloWorld

Root: <Project File Directory>

Project Format: Xcode 3.1-compatible

Place Build Products In:
[● Default build products location]
○ Custom location
/Volumes/Users/ernest/dev/shared_build

Place Intermediate Build Files In:
[● Default intermediates location]
○ Build products location
○ Custom location
/Volumes/Users/ernest/dev/

☐ Build independent targets in
In order to use this feature, this pr

Base SDK for All Configurations: Device – iPhone OS 2.1

SCM Repository: None

You can override the the base SDK setting for each build configuration (such as Release or Debug) with the iPhone OS Deployment Target build setting.

When you build your application, your target iPhone OS–release selection is reflected in the `MinimimOSVersion` entry in the application's `Info.plist` file.

When you publish your application to the App Store, the store indicates the iPhone OS release on which your application can run based on your application's `MinimumOSVersion` property.

To learn more about build settings, see "Editing Build Settings" in *Xcode Project Management Guide*.

Setting the Active SDK

The **active SDK** tells Xcode what release of the iPhone SDK to use to build your application the next time you perform a build operation and whether to run the application in iPhone Simulator or on a device.

There are two places you can set the active SDK:

- In the Set Active SDK submenu in the Project menu
- In the Overview pop-up menu in the toolbar

Important: If the iPhone OS release is earlier than the application's target iPhone OS release, Xcode displays build warnings when it detects that your application is using a feature that's not available in the target OS release. See "Setting Your Application's Target iPhone OS Release" (page 25) for more information.

You must also ensure that the symbols you use are available in the application's runtime environment using cross-development techniques. These techniques are described—in Mac OS X terms—in *Cross-Development Programming Guide*.

After testing your application in iPhone Simulator, you must test it on an iPhone OS–based device to measure and tune its performance. To be able to run your application on a device, you must be a member of the iPhone Developer Program; see "Preparing Devices for Development" (page 35) for details.

When changing the active SDK from iPhone Simulator SDK to iPhone Device SDK, keep in mind that the former provides a runtime environment for Mac OS X. Some of the key frameworks iPhone Simulator uses are tailored for Mac OS X. Moving to the iPhone Device SDK means that instead of Mac OS X–based frameworks and libraries, your application links against device-specific frameworks and libraries. Some of these frameworks and libraries are implemented differently or are not available for the device; for example, the ApplicationServices framework.

If your code imports iPhone Simulator SDK headers—the headers in the `<Xcode>/Platforms/iPhoneSimulator.platform/SDKs/.../usr/include` directory—directly, you may encounter build errors after switching to the iPhone Device SDK. You may need to conditionalize your code so that the `#import`/`#include` statements that use SDK headers are compiled only when building for the appropriate platform.

See "Conditional Compilation and Linkage" (page 55) for details about building your application so that it can run unchanged both in iPhone Simulator and on your device.

Setting the Active Build Configuration

A **build configuration** tells Xcode the purpose of the built product. Out of the box, Xcode lets you perform Debug and Release builds of your application. Debug builds include information and features that aid in debugging your application. Release builds produce smaller and faster binaries. During early and midlevel development, you should use the Debug configuration because it provides the best debugging experience. You should use the Release configuration in the last stages of development to measure and analyze your application's performance.

When you start a build process, Xcode uses the **active build configuration** to build your application. There are two places you can set the active build configuration:

- In the Set Active Build Configuration submenu in the Project menu
- In the Overview pop-up menu in the toolbar

To learn more about the Release and Debug configurations, see Building a Product.

Building Your Application

To start the build process, choose Build > Build.

The status bar in the project window indicates that the build was successful or that there are build errors or warnings. You can view build errors and warnings in the text editor or the project window.

Important: If you want your application to run in iPhone Simulator and on your device, you may need to conditionalize code compilation and framework linkage to adapt to the differences between the iPhone Simulator SDK and the iPhone Device SDK. See "Conditional Compilation and Linkage" (page 55) for details.

When building for iPhone Simulator, the generated binary runs only on the targeted iPhone Simulator release. It doesn't run on earlier or later releases of the simulator.

iPhone OS–based devices support two instruction sets, ARM and Thumb. Xcode uses Thumb instructions by default because using Thumb typically reduces code size by about 35 percent relative to ARM. Applications that have extensive floating point code might perform better if they use ARM instructions rather than Thumb. You can turn off Thumb for your application by turning off the Compile for Thumb build setting.

If the build completes successfully, you can proceed to run your application as described in "Running Your Application" (page 28).

If you encounter compilation errors, use the techniques described in "Viewing Errors and Warnings" in *Xcode Project Management Guide* to fix them.

When building for a device, if Xcode has trouble installing your application onto your device due to a problem with your provisioning profile, ensure that your provisioning profile is properly configured in the Program Portal. If necessary reinstall it on your computer and device, as described in "Preparing Devices for Development" (page 35).

To learn about the structure of iPhone application binaries, see "The Application Bundle" in *iPhone Application Programming Guide*.

Running Your Application

When you run your application, Xcode installs it on the iPhone Simulator or a device (depending on the active SDK) and launches it.

If you have more than one device attached to your computer, you can choose the device onto which Xcode installs the built application using the Project > Set Active Executable menu.

Once running, you can test that your application performs as you intend using all the capabilities of your device. You should especially ensure that your application uses the device's resources—CPU, memory, battery, and so on—as efficiently as possible. See "Tuning Applications" (page 49) for more information.

To run your application, choose Run > Run or Run > Debug.

Xcode launches your application on the iPhone Simulator or a device.

Troubleshooting: If you get the "Failed to start remote debugserver" while trying to debug your application on your device, your device may not be running the iPhone OS release that corresponds to your iPhone SDK. For more information, see "Restoring System Software" (page 40).

Streamlining the Build-and-Run Workflow

In addition to the Build, Run, and Debug commands, Xcode provides convenience commands that perform these operations as a single task. These commands are Build and Run and Build and Debug.

You can also repeat the last build task with the Build and Go command, which you can execute from the Build menu.

Managing Application Data

As you develop your application, you might need to rely on user settings and application data to remain on the iPhone Simulator or your development device between builds. Xcode doesn't remove any user settings or application data as you build your project and install the application on its host. But you may need to erase that information as part of testing your application the way users will use it. To do so, remove the application from the Home screen. See "Uninstalling Applications" (page 32) for details.

Further Exploration

To learn more about using Xcode to build and run applications, see Building a Product.

Using iPhone Simulator

The iPhone simulation environment lets you build and run your iPhone application on your computer. You use the simulator environment to:

- Find and fix major problems in your application during design and early testing.
- Learn about the Xcode development experience and the iPhone development environment before becoming a member of the iPhone Developer Program.
- Lay out and test your application's user interface.
- Measure your application's memory usage before carrying out detailed performance analysis on iPhone OS–based devices.

A major part of the iPhone simulation environment is the **iPhone Simulator application**. This application presents the iPhone user interface in a window on your computer. The application provides several ways of interacting with it using your keyboard and mouse to simulate taps and device rotation.

iPhone OS supports the Objective-C runtime introduced in Mac OS X v10.5 except for access to Objective-C class metadata. This means that, if your application accesses Objective-C class metadata, it may not run on the iPhone Simulator. See *Objective-C 2.0 Runtime Reference* for more information.

The following sections describe the ways in which you use your computer's input devices to simulate the interaction between users and their devices. They also show how to uninstall applications from the simulator and how to reset the contents of the simulator.

Manipulating the Hardware

The iPhone Simulator lets you simulate most of the actions a user performs on their device. You can carry out these hardware interactions through the iPhone Simulator Hardware menu:

- **Rotate Left.** Rotates the simulator to the left.
- **Rotate Right.** Rotates the simulator the right.
- **Home.** Takes the simulator to the Home screen.
- **Lock.** Locks the simulator.
- **Simulate Memory Warning.** Sends the frontmost application low-memory warnings. For information on how to handle low-memory situations, see "Observing Low-Memory Warnings" in *iPhone Application Programming Guide*.
- **Toggle In-Call Status Bar.** Toggles the status bar between its normal state and its in-call state. The status bar is taller in its in-call state than in its normal state. This command shows you how your application's user interface looks when the user launches your application while a phone call is in progress.

Performing Gestures

Table 4-1 lists gestures you can perform on the simulator (see *iPhone Human Interface Guidelines* for gesture information).

Table 4-1 Performing gestures in iPhone Simulator

Gesture	Desktop action
Tap	Click.
Touch and hold	Hold down the mouse button.
Double tap	Double click.
Swipe	1. Place the pointer at the start position. 2. Hold the mouse button. 3. Move the pointer in the swipe direction and release the mouse button.
Flick	1. Place the pointer at the start position. 2. Hold the mouse button. 3. Move the pointer quickly in the flick direction and release the mouse button.
Drag	1. Place the pointer at the start position. 2. Hold down the mouse button. 3. Move the pointer in the drag direction.
Pinch	1. Hold down the Option key. 2. Move the circles that represent finger touches to the start position. 3. Hold down the mouse button and move the circles to the end position.

Installing Applications

Xcode installs applications in iPhone Simulator automatically when you build your application using the iPhone Simulator SDK. See "Running Applications" (page 25) for details.

In the iPhone Simulator, you can install only applications that you build using the iPhone Simulator SDK. You cannot install applications from the App Store in the simulator.

Uninstalling Applications

To uninstall applications you have installed on the simulator use the same method used to uninstall applications from devices:

1. Place the pointer over the icon of the application you want to uninstall and hold down the mouse button until the icon starts to wiggle.
2. Click the icon's close button.
3. Click the Home button to stop the icon wiggling.

Resetting Content and Settings

To set the user content and settings of the simulator to their factory state and remove the applications you have installed, choose iPhone Simulator > Reset Content and Settings.

Core Location Functionality

The relocation reported by the CoreLocation framework in the simulator is fixed at the following coordinates (accuracy 100 meters), which correspond to 1 Infinite Loop, Cupertino, CA 95014.

- Latitude: 37.3317 North
- Longitude: 122.0307 West

iPhone Simulator File System on Host

iPhone Simulator stores the User domain of the iPhone OS file system in your home directory at:

```
~/Library/Application Support/iPhone Simulator/User
```

The User domain comprises all user data iPhone OS stores including third-party application binaries, system databases (such as the Address Book and Calendar databases), application preferences, and so on. This location is also known as `<iPhoneUserDomain>`.

iPhone Simulator stores system application preferences files in `<iPhoneUserDomain>/Library/Preferences`.

Third-party–application preferences files are stored in `<iPhoneUserDomain>/Applications/<app_UUID>Library/Preferences`.

Further Exploration

To learn about the frameworks available in iPhone Simulator, see iPhone OS Frameworks.

Managing Devices

With iPhone Simulator you can start developing iPhone applications without using iPhone OS–based devices. This way you can familiarize yourself with the API and development workflows used to develop applications. However, you must always test your applications on actual devices before publishing them to ensure that they run as intended and to tune them for performance on actual hardware.

As a member of the **Apple Developer Connection (ADC)** you can log in to the iPhone Dev Center, which provides access to iPhone developer documentation and lets you build iPhone applications that run in iPhone Simulator. (To become an ADC member, visit http://connect.apple.com.) An ADC membership, however, doesn't allow you to run applications on iPhone OS–based devices. To do so you must be a member of the iPhone Developer Program.

The **iPhone Developer Program** provides the tools and resources you need to run applications on development devices and distribute them to iPhone OS users. To become a member of the iPhone Developer Program, visit http://developer.apple.com/iphone/program.

After becoming an iPhone Developer Program member, you'll have access to the Program Portal in the iPhone Dev Center. The **Program Portal** is a restricted-access area of the iPhone Dev Center that allows you to configure devices to test your iPhone applications on them.

This chapter shows how to add your development devices to the iPhone Developer Program and how to configure your computer for development. It also shows how to use the Xcode Organizer window to view your application's console logs or crash information, or to take screenshots of your application as it runs. The chapter also describes how to safeguard the digital identifications required to install applications in development in devices.

Preparing Devices for Development

In order to test your application on a device, you must configure your computer and your device for iPhone OS development. This chapter presents an overview of the process and provides detailed information in the sections that follow.

In preparing your device for development, you create or obtain the following digital assets:

- **Certificate signing request.** A certificate signing request (CSR) contains personal information used to generate your development certificate. You submit this request to the iPhone Developer Program Portal.

  The Program Portal is visible only to members of the iPhone Developer Program.

  Note: The Program Portal is visible only to members of the iPhone Developer Program. To become an iPhone Developer Program member, visit http://developer.apple.com/iphone/program.

- **Development certificate.** A development certificate identifies an iPhone application developer. After the CSR is approved, you download your developer certificate from the portal and add it to your Keychain.

When you build your iPhone application with Xcode, it looks for your development certificate in your keychain; if it finds the certificate, Xcode signs your application, otherwise, it reports a build error.

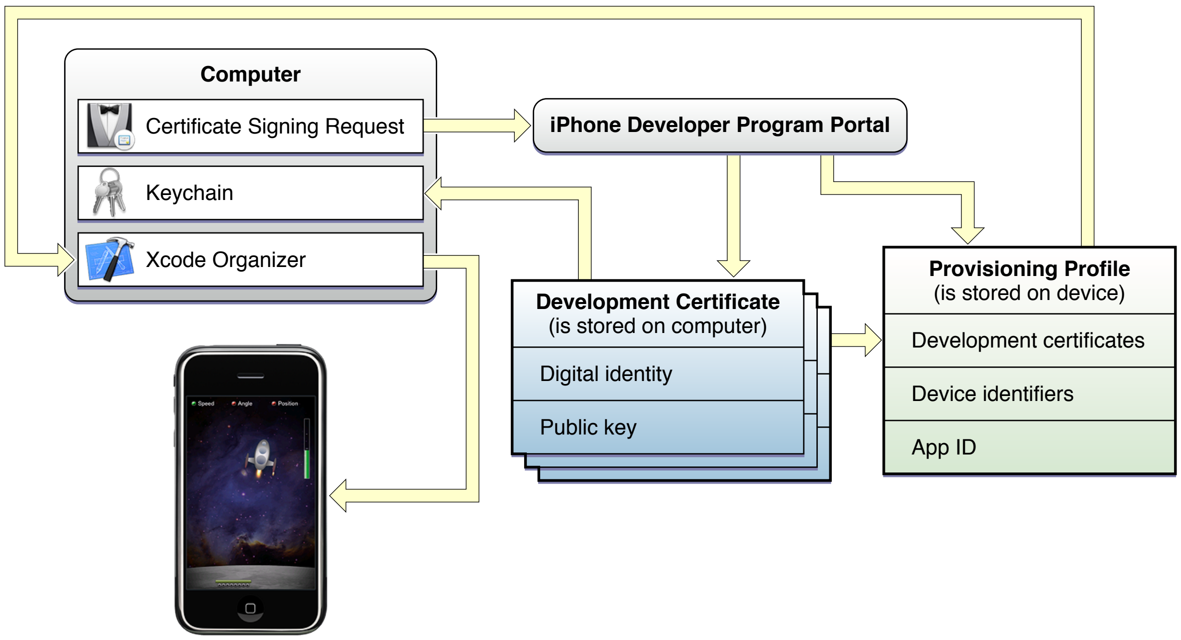
If your development certificate is missing from your keychain, you download it again from the Program Portal. Then, follow the instructions in "Adding Your Development Certificate to Your Keychain" (page 39).

- **Provisioning profile.** A provisioning profile associates one or more development certificates, devices, and an iPhone application ID (a unique identifier for the iPhone applications you or your organization develop under an iPhone Developer Program contract).

  To be able to install iPhone applications signed with your development certificate on a device, you must install at least one provisioning profile on the device. This provisioning profile must identify you (through your development certificate) and your device (by listing its unique device identifier). If you're part of an iPhone developer team, other members of your team, with appropriately defined provisioning profiles, may run applications you build on their devices.

Figure 5-1 illustrates the relationship between these digital assets.

Figure 5-1 Preparing computers and devices for iPhone development



These are the requirements your computer and your development device must meet so that you can build iPhone applications that run on your device:

- Your computer must have your development certificate in your keychain.
- Your device must contain at least one provisioning profile that contains your developer certificate and identifies your device.
- Your development device must have iPhone OS 2.0 or later installed.

These are the steps you must follow to configure your computer and development device for iPhone development:

1. Specify your application ID.
2. Register your device with the Program Portal.
3. Install iPhone OS on your device.
4. Obtain your development certificate.
5. Add your development certificate to your keychain.
6. Obtain your provisioning profile.
7. Add your provisioning profile to Xcode.
8. Install your provisioning profile on your device.

The following sections describe these tasks in detail.

Setting Your Application ID

After becoming a member of the iPhone Developer Program, you must set your **application ID** in the Program Portal. iPhone OS uses application IDs to identify the applications you create. An iPhone application ID is made up of a ten-character bundle seed identifier and a bundle identifier. The bundle identifier can identify one application or a group of applications.

This is an example of an iPhone application ID that identifies a single application, named MyApp:

```
GFWOTNXFIY.com.mycompany.MyApp
```

Using an asterisk instead of an application name in the bundle identifier, as shown below, lets you share a single application ID between a set of related applications.

```
GFWOTNXFIY.com.mycompany.myappsuite.*
```

Note: Regardless of the format used by the iPhone application ID, application bundles must always include the application name in their bundle identifier. To learn more about the bundle identifier, see "The Application Bundle" in *iPhone Application Programming Guide*.

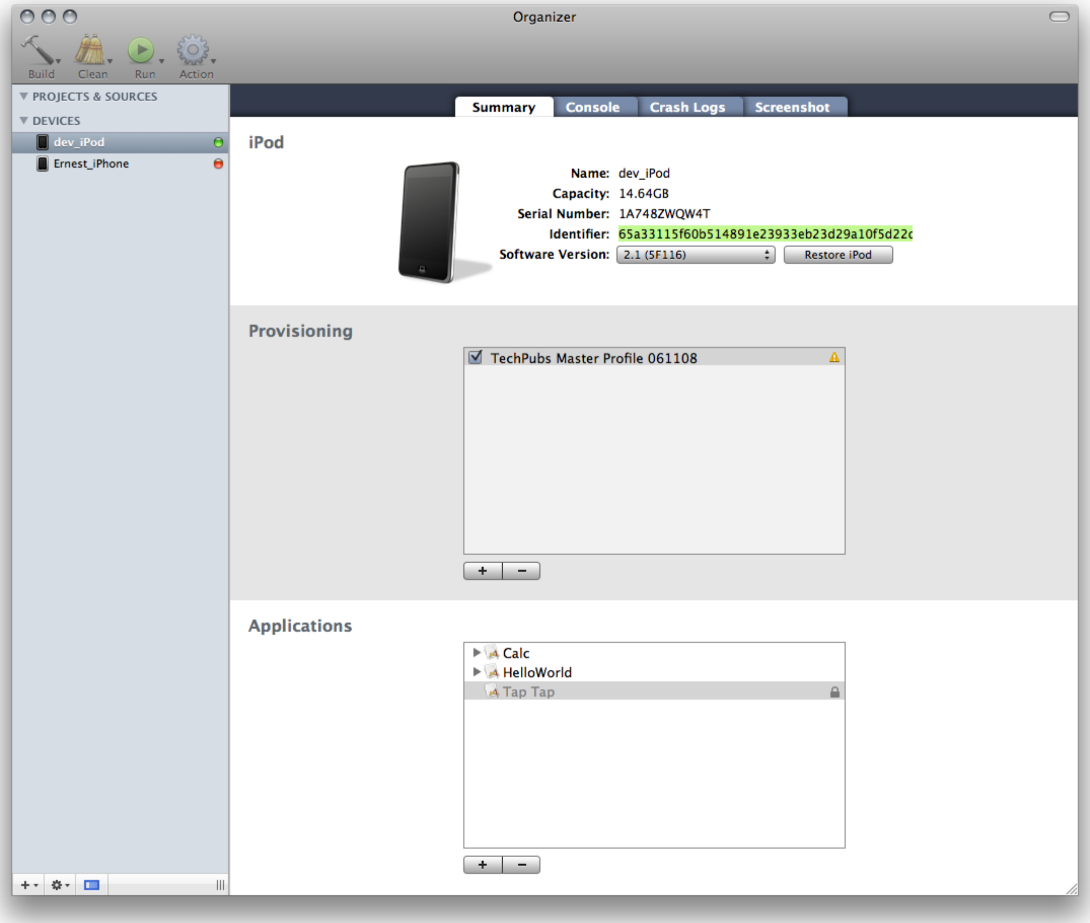
Registering Your Device with the Program Portal

To register your development device with the portal:

1. Launch Xcode.
2. Choose Window > Organizer to open the Organizer window.
3. Plug-in your device and select it in the devices list.

4. Copy your device UDID from the Identifier text field in the Summary pane, as shown in Figure 5-2 (page 38).

Figure 5-2 Organizer: Copying your device identifier



Name: dev_iPod
Capacity: 14.64GB
Serial Number: 1A748ZWQW4T
Identifier: 65a33115f60b514891e23933eb23d29a10f5d22c
Software Version: 2.1 (5F116)

☑ TechPubs Master Profile 061108

5. Go to the portal to register your device or have your team admin register your device into the program.

Installing iPhone OS on Your Device

To run applications you develop using the iPhone SDK, your device must be running iPhone OS 2.0 or later.

To learn how to install iPhone OS on your device, see "Restoring System Software" (page 40).

Obtaining Your Development Certificate

Xcode uses your development certificate to code-sign your application before it uploads it to your device for testing.

Start by generating a certificate signing request (CSR) on your computer:

1. Launch Keychain Access, located in `/Applications/Utilities`.
2. Choose Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority.
3. In the Certificate Information window:
    a. In the User Email Address field, enter your email address.
    b. In the Common Name field, enter your name.
    c. In the "Request is" group, select the "Saved to disk" option.
    d. Select "Let me specify key pair information."
    e. Click Continue.
    f. Choose your desktop as the location for the CSR file.
    g. In the Key Pair Information pane, choose 2048 as the key size and RSA as the algorithm.

    The Certificate Assistant saves a CSR file to your desktop.

    This process creates a public/private key pair. The public key is stored in your development certificate. Your private key is stored in your keychain. You must ensure that you don't lose your private key and that only you have access to it. Therefore, it's a good idea to backup your private key. Backing up your private key may also help if you need to use more than one computer to develop iPhone applications. See "Managing Your Digital Identifications" (page 43) for more information.
4. Open the CSR file in a text editor and copy the entire text, including the enclosing tags.
5. Submit the CSR to the Program Portal.

After the CSR is approved by your team admin, you can download your development certificate from the Program Portal. As with your private key, you should backup your development certificate in case you need to develop iPhone applications on another computer. See "Managing Your Digital Identifications" (page 43) for details.

Adding Your Development Certificate to Your Keychain

Your development certificate must be in your keychain so that Xcode can digitally sign your iPhone applications.

To add your development certificate to your keychain, in your computer:

1. Open your development certificate with the Keychain Access application by double-clicking it or dragging it to the Keychain Access application icon.
2. In the Add Certificates dialog, ensure Keychain is set to "login" and click OK.

Obtaining Your Provisioning Profile

To obtain your provisioning profile:

1. Have your team admin create your provisioning profile in the Program Portal.
2. Download your provisioning profile from the Program Portal.

Adding Your Provisioning Profile to the Xcode Organizer

You use the Organizer to add provisioning profiles to your development device.

To add a provisioning profile to Xcode:

1. Drag the provisioning profile file to the Xcode icon in the Dock.
2. Restart Xcode.

After this operation, the `~/Library/MobileDevice/Provisions` directory should contain your provisioning profile and it should also appear in the Provisioning section of the Summary pane in the Organizer.

Installing Your Provisioning Profile on Your Device

After adding your provisioning profile to the Organizer, you can add it to your device:

1. Open the Organizer window.

   Your provisioning profile should appear in the Provisioning section of the Summary pane. If you don't see it there, follow the instructions in "Adding Your Provisioning Profile to the Xcode Organizer" (page 40).

2. Plug in your device and select it in the devices list.
3. Click the checkbox next to the provisioning profile to install it on your device.

   Once installed, a checkmark should appear in the checkbox next to the provisioning profile. If the checkmark doesn't appear, ensure that the provisioning profile includes your device UDID, your development certificate, and a valid application ID. Go to the Program Portal or contact your team admin to verify that the provisioning profile contains this information. You need to go back to the "Obtaining Your Provisioning Profile" (page 40) step if changes are made to your profile.

Restoring System Software

When you develop applications using a particular version of the iPhone SDK, such as iPhone SDK 2.0, you should test those applications on devices running the iPhone OS version the SDK targets, such as iPhone OS 2.0.

You can download seed releases of iPhone OS that correspond to seed releases of the iPhone SDK from the iPhone Dev Center.

To restore a device:

1. Launch Xcode and open the Organizer window.
2. Plug the device into your computer.
3. Select the device in the Devices list.
4. From the Software Version pop-up menu, choose the version of iPhone OS you want to place on the device.

   If the version you want to install is not listed in the Software Version pop-up menu:

   a. Download the iPhone OS release you want to install on the device from http://developer.apple.com.

      Important: You must be a member of the iPhone Developer Program to be able to download iPhone OS.

   b. From the Software Version pop-up menu, choose Other Version.
   c. Navigate to the disk image containing the iPhone OS developer software and click Open.

      Xcode extracts the iPhone OS software from the disk image. You can dispose of the disk image you downloaded.

   d. From the Software Version pop-up menu, choose the newly downloaded iPhone OS version.

5. Click Reset iPhone or Reset iPod, depending on your device's type.
6. Use iTunes to name your device.

Running Applications on a Device

After following the instructions in "Preparing Devices for Development" (page 35) and "Restoring System Software" (page 40) (if necessary) you can run your application on your development device.

You tell Xcode that you want to build your application for a device and that you want to run it on a connected device by setting the active SDK to an iPhone Device SDK release. See "Setting the Active SDK" (page 26) for details.

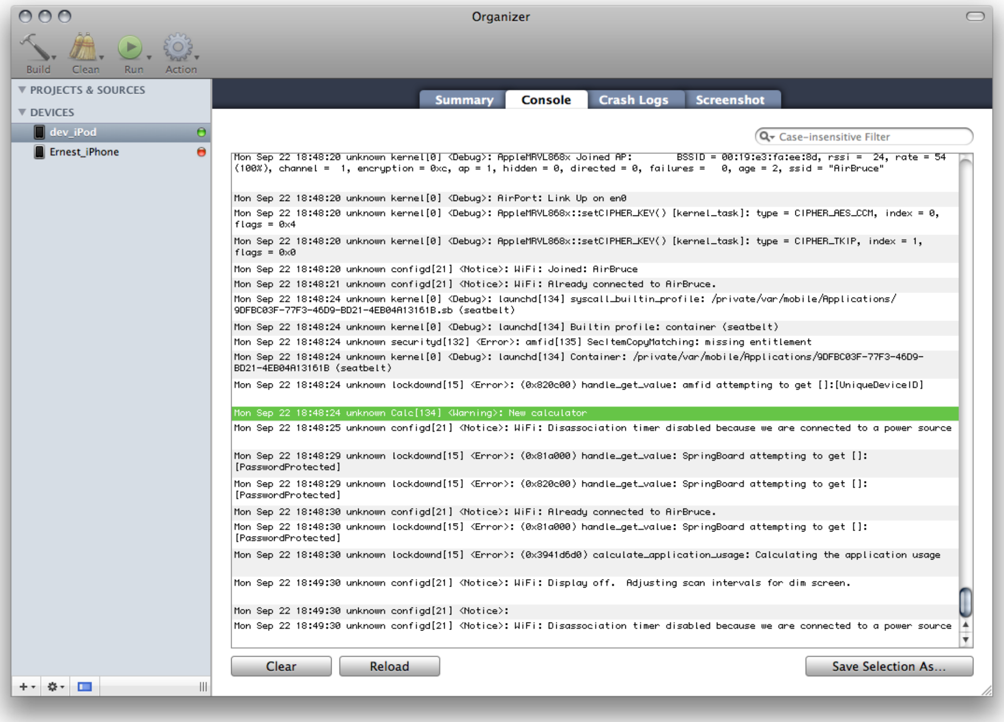
Viewing Console and Crash Logs

The iPhone OS frameworks, such as UIKit, produce log entries to the console to indicate, among other things, when an unexpected event occurs. You can read those messages using the Console application in `/Application/Utilities`. You can produce console messages in your iPhone applications, too. One way to produce console logs is to use the `NSLog` class. In addition to the Xcode debugger, console logs may help you analyze your application's logic and track down bugs.

When running your application on the iPhone Simulator, you can access its console logs in the Xcode Console window. But when you run the application on your development device, you must use the Xcode Organizer to access log entries.

To view a device's console output:

1. Open the Organizer window.
2. Select the device whose console log you want to view.
3. Click Console.



```
Mon Sep 22 18:48:20 unknown kernel[0] <Debug>: AppleMRVL868x Joined AP:        BSSID = 00:19:e3:fa:ee:8d, rssi =  24, rate = 54
(100%), channel =  1, encryption = 0xc, ap = 1, hidden = 0, directed = 0, failures =   0, age = 2, ssid = "AirBruce"

Mon Sep 22 18:48:20 unknown kernel[0] <Debug>: AirPort: Link Up on en0
Mon Sep 22 18:48:20 unknown kernel[0] <Debug>: AppleMRVL868x::setCIPHER_KEY() [kernel_task]: type = CIPHER_AES_CCM, index = 0,
flags = 0x4
Mon Sep 22 18:48:20 unknown kernel[0] <Debug>: AppleMRVL868x::setCIPHER_KEY() [kernel_task]: type = CIPHER_TKIP, index = 1,
flags = 0x0
Mon Sep 22 18:48:20 unknown configd[21] <Notice>: WiFi: Joined: AirBruce
Mon Sep 22 18:48:21 unknown configd[21] <Notice>: WiFi: Already connected to AirBruce.
Mon Sep 22 18:48:24 unknown kernel[0] <Debug>: launchd[134] syscall_builtin_profile: /private/var/mobile/Applications/
9DFBC03F-77F3-46D9-BD21-4EB04A13161B.sb (seatbelt)
Mon Sep 22 18:48:24 unknown kernel[0] <Debug>: launchd[134] Builtin profile: container (seatbelt)
Mon Sep 22 18:48:24 unknown securityd[132] <Error>: amfid[135] SecItemCopyMatching: missing entitlement
Mon Sep 22 18:48:24 unknown kernel[0] <Debug>: launchd[134] Container: /private/var/mobile/Applications/9DFBC03F-77F3-46D9-
BD21-4EB04A13161B (seatbelt)
Mon Sep 22 18:48:24 unknown lockdownd[15] <Error>: (0x820c00) handle_get_value: amfid attempting to get []:[UniqueDeviceID]

Mon Sep 22 18:48:24 unknown Calc[134] <Warning>: New calculator
Mon Sep 22 18:48:25 unknown configd[21] <Notice>: WiFi: Disassociation timer disabled because we are connected to a power source

Mon Sep 22 18:48:29 unknown lockdownd[15] <Error>: (0x81a000) handle_get_value: SpringBoard attempting to get []:
[PasswordProtected]
Mon Sep 22 18:48:29 unknown lockdownd[15] <Error>: (0x820c00) handle_get_value: SpringBoard attempting to get []:
[PasswordProtected]
Mon Sep 22 18:48:30 unknown configd[21] <Notice>: WiFi: Already connected to AirBruce.
Mon Sep 22 18:48:30 unknown lockdownd[15] <Error>: (0x81a000) handle_get_value: SpringBoard attempting to get []:
[PasswordProtected]
Mon Sep 22 18:48:30 unknown lockdownd[15] <Error>: (0x3941d6d0) calculate_application_usage: Calculating the application usage

Mon Sep 22 18:49:30 unknown configd[21] <Notice>: WiFi: Display off.  Adjusting scan intervals for dim screen.

Mon Sep 22 18:49:30 unknown configd[21] <Notice>:
Mon Sep 22 18:49:30 unknown configd[21] <Notice>: WiFi: Disassociation timer disabled because we are connected to a power source
```

You can use the search field to filter log entries. You can also save the log to a file.

The Crash Log pane in the Organizer contains information about application crashes. You may have to unplug your device and plug it in again to refresh the crash list.

Capturing Screen Shots

Screen shots help to document your application. This is also how you create your application's default image, which iPhone OS displays when the user taps your application's icon. You can capture screen shots of your device's screen from the Organizer or directly on your device.

To capture a screen shot from the Organizer:

1. Configure your application's screen for the screen shot.

   Depending on your application's workflow, you may need to place breakpoint in your code and run your application until it reaches that point.

2. Open the Organizer window, select your device, and click Screenshots.

3. Click Capture.

   To make that screen shot your application's default image, click Save As Default Image.

If you have iPhoto installed on your computer, you may capture screen shots directly on your device and import them into your iPhoto library.

To capture a screen shot on your device, press the the Lock and Home buttons simultaneously. Your screen shot is saved in the Saved Photos album in the Photos application.

Note: Although the default image includes the status bar as it looked when the screen shot was captured, iPhone OS replaces it with the current status bar when your application launches.

Managing Your Digital Identifications

When you create a certificate signing request (CSR) to obtain your development certificate, you generated a public/private key pair. The public key is included in your development certificate. The private key is stored in your keychain. With these two items in your computer, Xcode can code-sign the iPhone applications you build with it. If you need to use another computer to develop iPhone applications, you must transfer these items to the other computer and add them to your keychain.

This section shows how to export your private key from your keychain in your development computer, store your private key and development certificate in a protected disk image, and add both items to a second computer for iPhone development.

To export your private key from your keychain:

1. Launch Keychain Access.

2. In the category list, select Keys.

3. Select the private key you use for iPhone development.

4. Choose Export from the private key shortcut menu.

(To display the private key shortcut menu, Control-click the selected row.)

5. Enter a password to protect the private key.

6. Select a location for the private key and use the Personal Information Exchange (`.p12`) format for the file.

To generate a protected disk image containing your private key and development certificate:

1. Place your private key and development certificate file in a newly created directory, named `iPhone Developer Identifications`.

2. Launch the Disk Utility application, located in `/Applications/Utilities`.

3. Choose File > New > Disk Image from Folder.

4. Choose the `iPhone Developer Identifications` directory you created earlier.

5. Select a location for the new protected disk image.

6. From the Encryption pop-up menu, choose "256-AES encryption".

7. In the dialog that appears, enter a password for the disk image.

   You should deselect the "Remember password in my keychain" option. Otherwise, anybody with access to your account may open the disk image.

8. Place the protected disk image in a secure location.

Now, when you need to develop iPhone applications on another computer:

1. Copy the `iPhone Developer Identifications.dmg` disk-image file to the second computer.

2. On the second computer, open the disk image.

3. Import the private key into your keychain:

   a. Launch Keychain Access.

   b. Choose File > Import Items.

   c. Choose the private key file to import.

4. Add the development certificate to your keychain. See "Adding Your Development Certificate to Your Keychain" (page 39) for details.

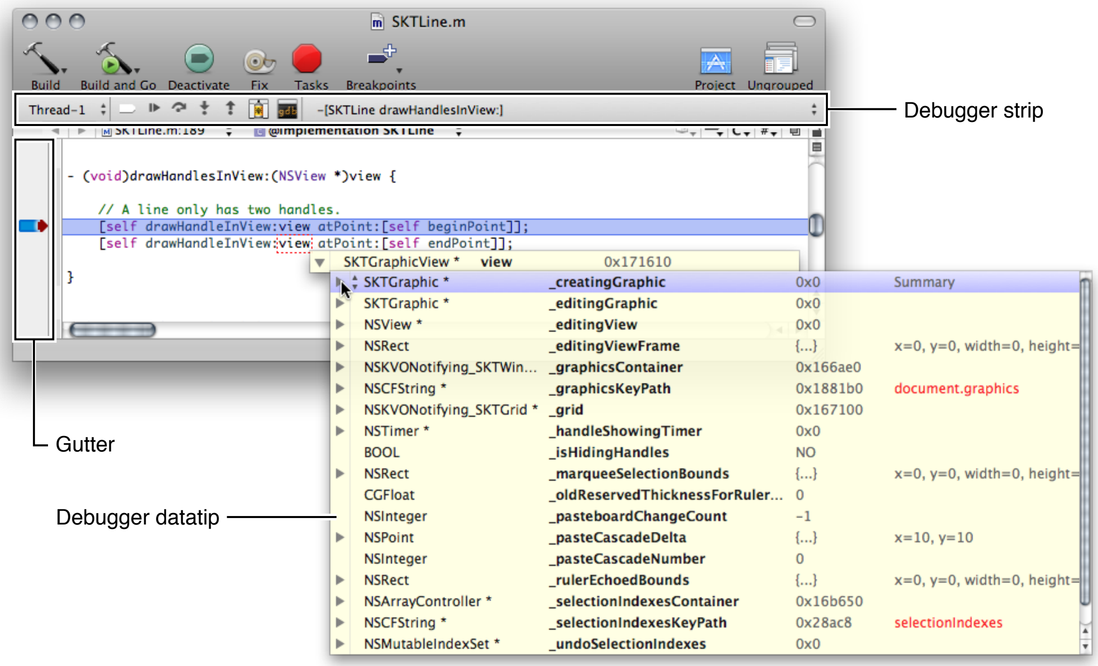
Debugging Applications

This chapter describes the Xcode debugging facilities.

General Debugging Tasks

Xcode provides several debugging environments you can use to find and squash bugs in your code:

- **The text editor.** The text editor allows you to debug your code right *in* your code. It provides most of the debugging features you need. You can
    - Add and set breakpoints
    - View your call stack per thread
    - View the value of variables by hovering the mouse pointer over them
    - Execute a single line of code
    - Step in to, out of, or over function or method calls



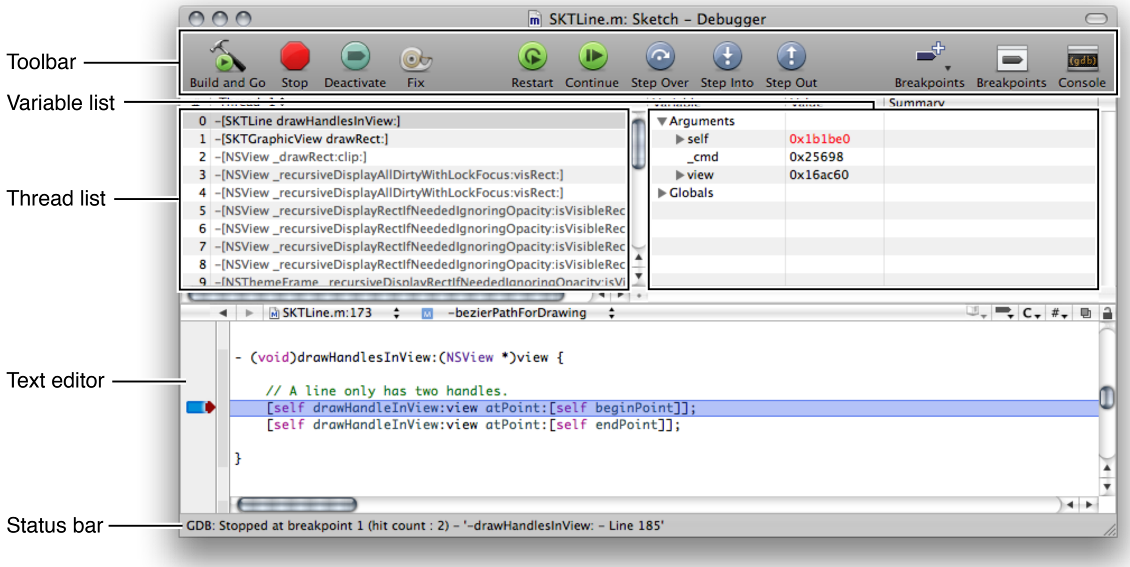
```
- (void)drawHandlesInView:(NSView *)view {

    // A line only has two handles.
    [self drawHandleInView:view atPoint:[self beginPoint]];
    [self drawHandleInView:view atPoint:[self endPoint]];

}
```

```
SKTGraphicView *    view                               0x171610
  SKTGraphic *                 _creatingGraphic                 0x0        Summary
  SKTGraphic *                 _editingGraphic                  0x0
  NSView *                     _editingView                     0x0
  NSRect                       _editingViewFrame                {...}      x=0, y=0, width=0, height=
  NSKVONotifying_SKTWin...     _graphicsContainer               0x166ae0
  NSCFString *                 _graphicsKeyPath                 0x1881b0   document.graphics
  NSKVONotifying_SKTGrid *     _grid                            0x167100
  NSTimer *                    _handleShowingTimer              0x0
  BOOL                         _isHidingHandles                 NO
  NSRect                       _marqueeSelectionBounds          {...}      x=0, y=0, width=0, height=
  CGFloat                      _oldReservedThicknessForRuler... 0
  NSInteger                    _pasteboardChangeCount           -1
  NSPoint                      _pasteCascadeDelta               {...}      x=10, y=10
  NSInteger                    _pasteCascadeNumber              0
  NSRect                       _rulerEchoedBounds               {...}      x=0, y=0, width=0, height=
  NSArrayController *          _selectionIndexesContainer       0x16b650
  NSCFString *                 _selectionIndexesKeyPath         0x28ac8    selectionIndexes
  NSMutableIndexSet *          _undoSelectionIndexes            0x0
```

- **The Debugger window.** When you need to perform more focused debugging, the Debugger window provides all the debugging features the text editor provides using a traditional interface. This window provides lists that allow you to see your call stack and the variables in scope at a glance.



```
0 -[SKTLine drawHandlesInView:]
1 -[SKTGraphicView drawRect:]
2 -[NSView _drawRect:clip:]
3 -[NSView _recursiveDisplayAllDirtyWithLockFocus:visRect:]
4 -[NSView _recursiveDisplayAllDirtyWithLockFocus:visRect:]
5 -[NSView _recursiveDisplayRectIfNeededIgnoringOpacity:isVisibleRec
6 -[NSView _recursiveDisplayRectIfNeededIgnoringOpacity:isVisibleRec
7 -[NSView _recursiveDisplayRectIfNeededIgnoringOpacity:isVisibleRec
8 -[NSView _recursiveDisplayRectIfNeededIgnoringOpacity:isVisibleRec
9 -[NSThemeFrame _recursiveDisplayRectIfNeededIgnoringOpacity:isVi
```

```
▼ Arguments
  ▶ self     0x1b1be0
    _cmd     0x25698
  ▶ view     0x16ac60
▶ Globals
```

```
- (void)drawHandlesInView:(NSView *)view {

    // A line only has two handles.
    [self drawHandleInView:view atPoint:[self beginPoint]];
    [self drawHandleInView:view atPoint:[self endPoint]];

}
```

GDB: Stopped at breakpoint 1 (hit count : 2) – '-drawHandlesInView: – Line 185'

- **The GDB console.** A GDB console window is available for text-based debugging.

For more information about the Xcode debugging facilities, see *Xcode Debugging Guide*.

Memory Leaks

If you fix a leak and your program starts crashing, your code is probably trying to use an already-freed object or memory buffer. To learn more about memory leaks, see Finding Memory Leaks.

You can use the NSZombieEnabled facility to find the code that accesses freed objects. When you turn on NSZombieEnabled, your application logs accesses to deallocated memory, as shown here:

```
2008-10-03 18:10:39.933 HelloWorld[1026:20b] *** -[GSFont ascender]: message sent to
deallocated instance 0x126550
```

To activate the NSZombieEnabled facility in your application:

1. Choose Project > Edit Active Executable to open the executable Info window.
2. Click Arguments.
3. Click the add (+) button in the "Variables to be set in the environment" section.
4. Enter `NSZombieEnabled` in the Name column and `YES` in the Value column.
5. Make sure that the checkmark for the `NSZombieEnabled` entry is selected.

For more information about configuring executable environments, see “Configuring Executable Environments” in *Xcode Project Management Guide*.

Tuning Applications

Optimizing your application's performance is an important phase of the development, more so in iPhone OS–based devices, which, although powerful computing devices, do not have the memory or CPU power that desktop or portable computers possess. You also have to pay attention to your application's battery use, as it directly impacts your customer's battery-life experience.

These chapter describes Instruments and Shark, the tools you use to measure and tune your application's performance.

The Instruments Application

The Instruments application lets you gather a variety of application performance metrics, such as memory and network use. You can gather data from iPhone applications running in iPhone Simulator or on devices.

It is important that your iPhone applications use the resources of iPhone OS–based devices as efficiently as possible to provide a compelling experience for you customers. For example, your application should not use resources in a way that makes the application feel sluggish to users or drains their batteries too quickly. Applications that use too much memory run slowly. Applications that rely on the network for their operation must use it as sparingly as possible because powering up the radios for network communications is a significant drag on the battery.

The Instruments application provides an advanced data gathering interface that lets you know exactly how your application uses resources, such as the CPU, memory, file system, and so on.

Instruments uses software-based data-gathering tools, known as instruments, to collect performance data. An **instrument** collects a specific type of data, such as network activity or memory usage. You find which instruments are available for iPhone OS in the Instruments Library.

Although most iPhone applications run in iPhone Simulator and you can test most design decisions there, the simulator does not emulate a device, in particular it doesn't attempt to replicate a device's performance characteristics such as CPU speed or memory throughput. To effectively measure your application's performance as users may use it on their devices, you must use an iPhone or iPod touch. That's because only on a device can you can get an accurate representation of the runtime environment (in terms of processor speed, memory limitations, specialized hardware, and the like).

These are some limitations of iPhone Simulator:

- **Maximum of two fingers.** If your application's user interface can respond to touch events involving more than two fingers, you can test that capability only on actual devices.
- **Accelerometer.** Although you can access your computer's accelerometer (if it has one) through the UIKit framework, its readings will differ from the accelerometer readings on a device. This discrepancy stems largely from the different positioning of the screen in relation to the rest of the hardware between computers and iPhone OS–based devices.

- **OpenGL ES.** OpenGL ES uses renderers on devices that are slightly different from those it uses in iPhone Simulator. For this reason, a scene on the simulator and the same scene on a device may not be identical at the pixel level. See "Drawing with OpenGL ES" in *iPhone Application Programming Guide* for details.

To measure your application's performance on a device:

1. Build and run your application on the device as described in "Running Applications" (page 25).
2. Launch Instruments.

   The Instruments application is located at `<Xcode>/Applications`. (`<Xcode>` refers to the installation location of the development tools.)
3. Choose a template, such as Activity Monitor, to create the trace document.

   A **trace document** contains one or more instruments that collects data about a process.
4. From the Default Target pop-up menu in the toolbar, select the iPhone OS–based device containing the application from which you want to collect performance data.
5. Add or remove instruments from the trace document to collect the desired data.
6. Use the Default Target pop-up menu, to launch or attach to the target application.
7. Click Record to start collecting data and use your application, exercising the areas you want to examine.

To learn more about measuring and analyzing application performance, see *Instruments User Guide*. This document provides general information about using Instruments.

The Shark Application

To complement the performance data Instruments collects, the Shark application lets you view system-level events, such as system calls, thread-scheduling decisions, interrupts, and virtual memory faults. You can see how your code's threads interact with each other and how your code interacts with iPhone OS.

When performance problems in your code are more related to the interaction between your code, iPhone OS, and the hardware architecture of the device, you can use Shark to get information about those interactions and find performance bottlenecks.

For information about using Shark with your iPhone applications, see *Shark User Guide*.

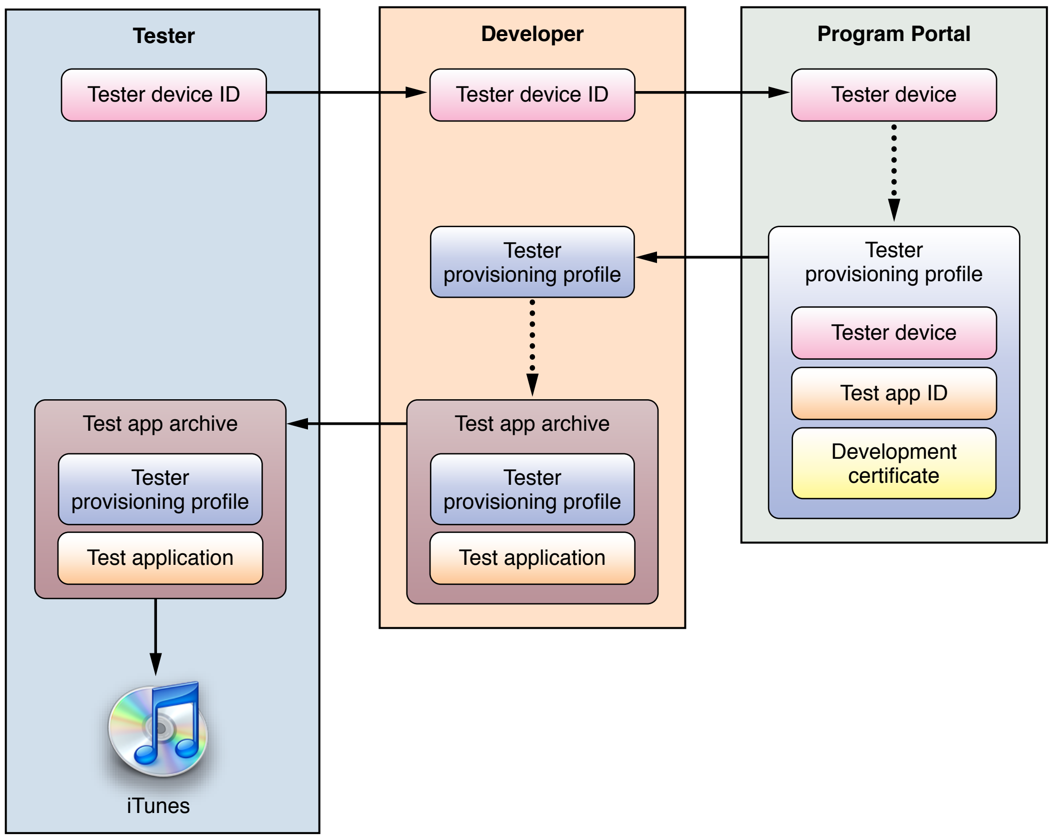
Publishing Applications for Testing

After testing and tuning your application yourself or with the assistance of your teammates, it's always a good idea to perform wider testing with a representative sample of your application's potential users. Such testing may reveal issues that surface only with particular usage patterns. Incorporating a few nondeveloper users in your testing strategy lets you expose your application to a variety of usage styles, and, if such usage produces crashes in your application, allows you to collect the crash reports (also known as crash logs) from those users to help you resolve those execution problems.

An iPhone application in development can run only on devices with provisioning profiles generated by the application developer. As iPhone Developer Program members, you and your fellow team members install these files on your devices as part of your development process. To include users that are not part of your team (also known as testers) in your testing strategy, you must add them as part of your team in the Program Portal and issue them **test provisioning profiles** (also known as ad-hoc provisioning profiles), which allow them to install on their devices applications that have not been published to the App Store.

Figure 8-1 illustrates the process of adding users as testers and delivering your test application to them.

Figure 8-1 Adding testers to your team



To help testers obtain the information you need to add them to your testing program and to show them how to send you crash logs, you can send them the information in "Instructions for Application Testers" (page 61).

Important: To add testers to your team, you must be a member of the iPhone Developer Program. See "Managing Devices" (page 35) for details.

The remainder of this chapter describes the steps you need to perform to add testers to your team and shows how to add symbol information to their crash logs, also known as symbolicating.

Adding Testers to Your Team

To add an iPhone OS user to your team as a tester:

1. Obtain the tester's device ID.

   The easiest way to obtain this information is through email. Have your tester follow the instructions for sending their device ID to developers in "Sending Your Device ID to Developers" (page 61).

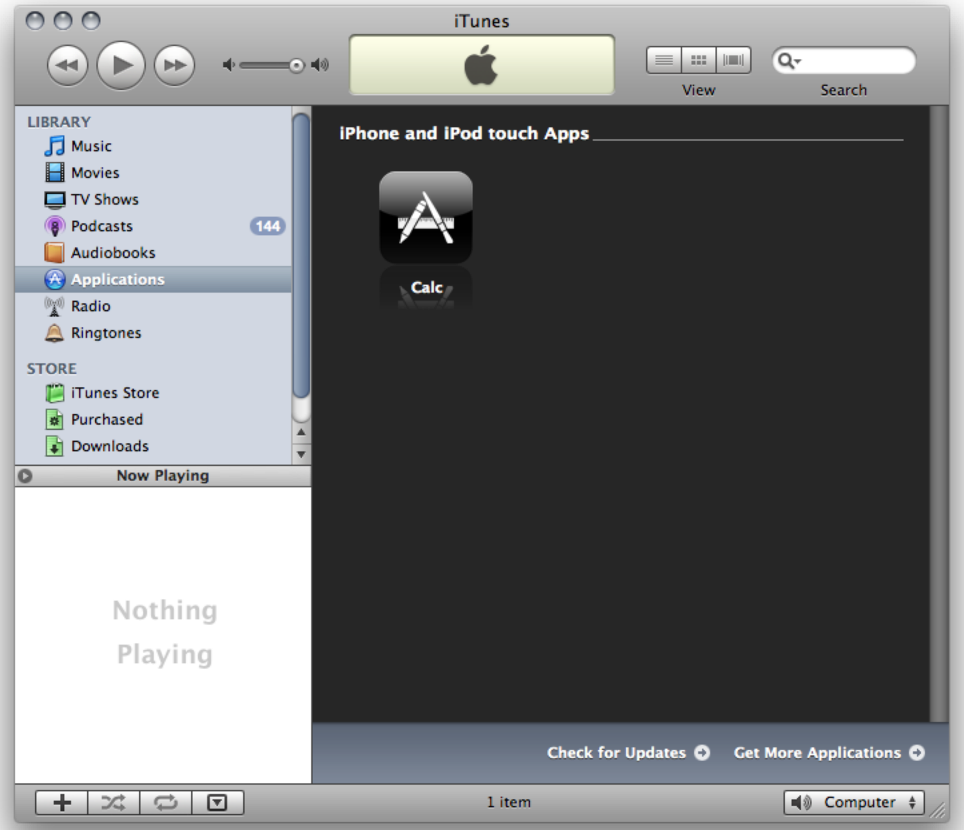
2. Add the tester's device ID to the Program Portal.
3. Generate the tester's provisioning profile in the Program Portal.

   You must select a development certificate, application ID, and only the tester's device name.

Adding the iTunes Artwork to Your Application

Test versions of your application should contain artwork iTunes uses to identify your application. Otherwise, when users add your application to their iTunes library, iTunes uses generic artwork for it, as shown in Figure 8-2.

Figure 8-2 Generic iTunes artwork for test applications



The iTunes artwork your testers see should be your application's icon. This artwork must be a 512 x 512 JPEG or PNG file named `iTunesArtwork`. Note that the file must not have an extension.

After generating the file of your application's icon, follow these steps to add it to your application:

1. Open your project in Xcode.
2. In the Groups & Files list, select the Resources group.
3. Choose Project > Add to Project, navigate to your `iTunesArtwork` file, and click Add.
4. In the dialog that appears, select the "Copy items" option and click Add.

Distributing Your Application to Testers

To send your application to a tester:

1. Build your application using the Release build configuration.

Remember to keep the binary and its corresponding dSYM file on your file system (copy them to a directory that contains a subdirectory for each build you've released to your teammates or to testers) so that they're indexed by Spotlight.

2. Create an archive containing the application binary and the tester's provisioning profile.

   In the Finder, select the two files and choose File > Compress.

   Name the archive `<application_name> for <device_name>.zip`. For example, `MyTestApp for Anna Haro.zip`.

3. Send the archive to your tester.

   In the Finder, select the archive and choose Finder > Services > Mail > Send File.

   In the body of the message make sure to include your application's target iPhone OS release. If the tester's iPhone OS version is earlier than the one for which your application was built, they will not be able to install the application on their device.

Adding Symbol Information to Crash Logs from Testers

To add symbol information to crash logs (also known as symbolicating) after receiving them from testers:

1. In the Organizer, select a development device in the Devices list.

2. Drag the crash logs to the Crash Logs pane.

Conditional Compilation and Linkage

The two iPhone OS runtime environments are the iPhone simulation environment and the iPhone device environment. You use the former to test your application on your Mac, and the latter to test it on a device. These environments are fundamentally different; therefore, when using technology that's implemented differently in the two environments, such as OpenGL ES, you need to tweak your code so that some of it runs in iPhone Simulator application but not on a device. You may also need to link to different frameworks to obtain the same functionality in the simulator and a device, such as when you use the CFNetwork framework.

This chapter shows how to target code to iPhone Simulator or a device and how to link to frameworks or libraries depending on whether the active SDK belongs to the iPhone Simulator SDK family or the iPhone Device SDK family.

Compiling Source Code Conditionally for iPhone Applications

There may be times when you need to run code on the simulator but not on a device, and the other way around. On those occasions, you can use the preprocessor macros `TARGET_OS_IPHONE` and `TARGET_IPHONE_SIMULATOR` to conditionally compile code.

Listing 9-1 shows how to use the `TARGET_IPHONE_SIMULATOR` macro to determine whether code meant for iPhone OS is being compiled for the simulator or devices.

Listing 9-1 Determining whether you're compiling for the simulator

```
// Set hello to "Hello, <device or simulator>"!
#if TARGET_IPHONE_SIMULATOR
   NSString *hello = @"Hello, iPhone simulator!";
#else
   NSString *hello = @"Hello, device!";
#endif
```

Listing 9-2 shows how to use the `TARGET_OS_IPHONE` macro in a source to be shared between Mac OS X and iPhone OS.

Listing 9-2 Determining whether you're compiling for iPhone OS

```
#if TARGET_OS_IPHONE
   #import <UIKit/UIKit.h>
#else
   #import <Cocoa/Cocoa.h>
#endif
```

The `TARGET_OS_IPHONE` and `TARGET_IPHONE_SIMULATOR` macros are defined in the `TargetConditionals.h` header file.

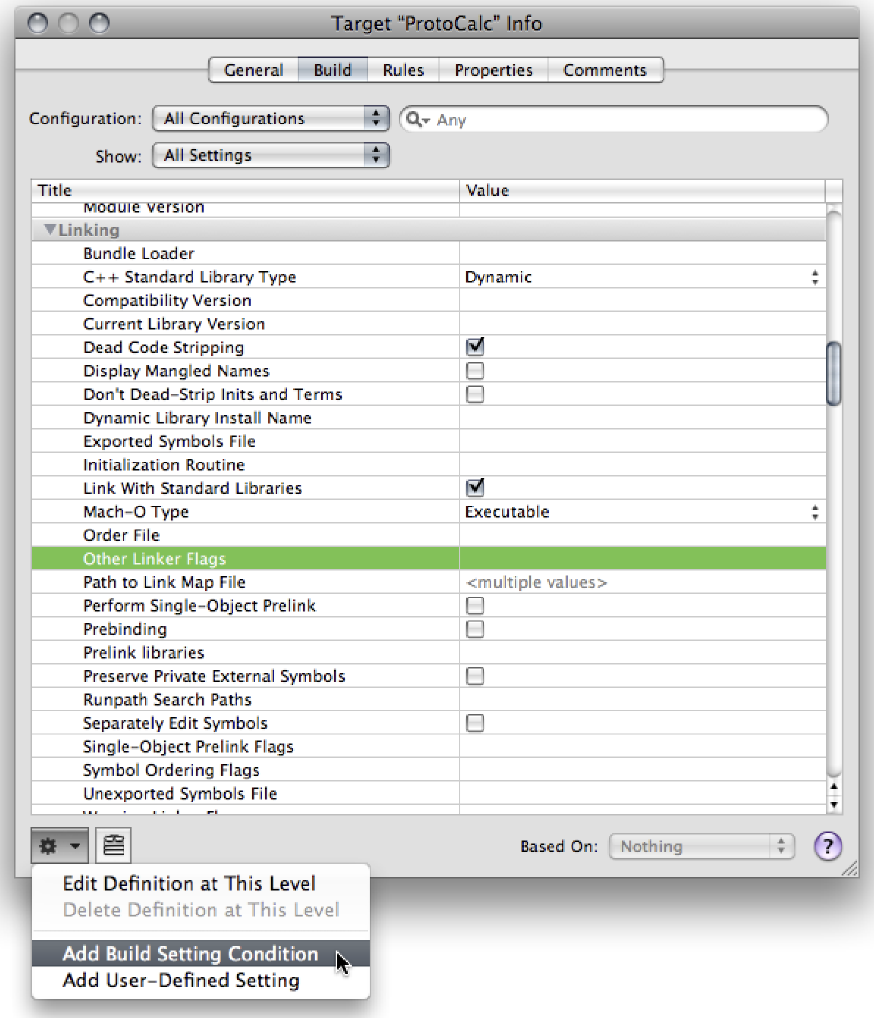
Linking Frameworks Conditionally for iPhone Applications

There may be occasions when you need to configure your application target so that it links against one framework to run on the simulator and a different framework to run on a device. For example, if you need to use the OpenAL framework to deliver 3D sound in your iPhone application, you link against it when building using the iPhone Device SDK family; when building using the iPhone Simulator SDK family, you don't link against the OpenAL framework because this SDK family doesn't include it.

To link a framework only when when using a particular SDK:

1. In your project's Group & Files list, double-click your application target to open the target Info window.
2. Click Build to display the Build pane.
3. From the Configuration pop-up menu, choose All Configurations.
4. From the Show pop-up menu, choose All Settings.
5. In the build setting lists, scroll to the Linking group.
6. Select the Other Linker Flags build setting.

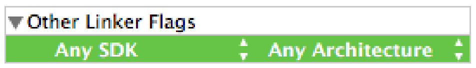
7. From the Action menu (the gear icon at the bottom-left corner of the window), choose Add Build Setting Condition.



Configuration: All Configurations
Show: All Settings
Based On: Nothing

Title	Value
Module Version	
Linking	
Bundle Loader	
C++ Standard Library Type	Dynamic
Compatibility Version	
Current Library Version	
Dead Code Stripping	☑
Display Mangled Names	☐
Don't Dead-Strip Inits and Terms	☐
Dynamic Library Install Name	
Exported Symbols File	
Initialization Routine	
Link With Standard Libraries	☑
Mach-O Type	Executable
Order File	
Other Linker Flags	
Path to Link Map File	<multiple values>
Perform Single-Object Prelink	☐
Prebinding	☐
Prelink libraries	
Preserve Private External Symbols	☐
Runpath Search Paths	
Separately Edit Symbols	☐
Single-Object Prelink Flags	
Symbol Ordering Flags	
Unexported Symbols File	

A build setting condition appears below the build setting title.


Other Linker Flags
Any SDK | Any Architecture

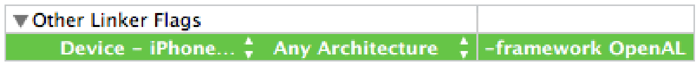
8. Set the SDK condition.

   From the Any SDK pop-up menu, choose the SDK under which you want the build setting specification to apply.

9. Set the build setting value.

   a. Click the Value cell to the right of the Architecture condition (which you should leave as Any Architecture).

b. Enter `-framework <framework_name>` into the Value cell.



Other Linker Flags

Device - iPhone... Any Architecture -framework OpenAL

If you need to, you can add another condition to the Other Linker Flags build setting to specify a different SDK and framework.

For more information about editing build settings, see "Editing Build Settings" in *Xcode Project Management Guide*.

iPhone Development FAQ

Here are some common questions developers ask about iPhone Simulator:

- Can GCC 4.2 be used with the iPhone Simulator SDK?

  No.

- Does the iPhone Simulator application run on network home directories?

  No.

- Do Objective-C properties need to be backed up by instance variables or accessor methods for them to work?

  Yes.

- Why do curl transitions appear as fade transitions iPhone Simulator?

  Because Mac OS X doesn't support curl transitions.

- Do static libraries need to be code-signed before being used in an iPhone application?

  No.

Instructions for Application Testers

This appendix provides instructions to testers about the procedures to follow to test your iPhone applications on their devices.

You may send these instructions, along with any special tasks needed to test your application, to customers interested in testing your application.

Sending Your Device ID to Developers

Before a developer can send you an application for testing, they must register your device under their iPhone Developer Program.

To send your device ID to a developer for test-program registration:

1. Launch iTunes.

2. Connect your device to your computer.

3. Select the device in the Devices list.

4. In the Summary pane, click the Serial Number label. It changes to Identifier.

5. Choose Edit > Copy.

6. Email your device identifier to the developer.

   Be sure to include your name and device name in the email.

Installing an Application for Testing

After being successfully registered in a developer's testing program, the developer will send you an archive containing two files: the application and a provisioning profile. You need to install both files into iTunes to be able run the application on your device.

After opening the archive:

1. Drag the provisioning profile (the file with the `.provision` extension) to the Library group.

2. Drag the application (the file with the `.app` or `.exe` extension) to the Library group.

   The application appears in the Applications list.

3. Sync your device.

If the version of iPhone OS on your device is earlier than the test application can run on, you need to update your device with the current release of iPhone OS.

Sending Crash Reports to Developers

When the application you're testing crashes, iPhone OS creates a record of that event. The next time you connect your device to iTunes, iTunes downloads those records (known as crash logs) to your computer. To help get the problem fixed, you should send crash logs of the application you're testing to its developer.

Sending Crash Reports from Macs

To send crash logs to developers:

1. In the Finder, open a new window.
2. Choose Go > Go to Folder.
3. Enter `~/Library/Logs/CrashReporter/MobileDevice`.
4. Open the folder named after your device's name.
5. Select the crash logs named after the application you're testing.
6. Choose Finder > Services > Mail > Send File.
7. In the New Message window, enter the developer's email address in the To field and `<application_name> crash logs from <your_name>` (for example, `MyTestApp crash logs from Anna Haro`) in the Subject field.
8. Choose Message > Send.
9. In the Finder, you may delete the crash logs you sent to avoid sending duplicate reports later.

Sending Crash Reports from Windows

To send crash logs to developers, enter the crash log directory (Listing A-1 and Listing A-2) in the Windows search field, replacing `<user_name>` with your Windows user name.

Listing A-1 Crash log storage on Windows Vista

```
C:\Users\<user_name>\AppData\Roaming\Apple
computer\Logs\CrashReporter/MobileDevice
```

Listing A-2 Crash log storage on Windows XP

```
C:\Documents and Settings\<user_name>\Application Data\Apple
computer\Logs\CrashReporter
```

Open the folder named after your device's name and send the crash logs for the application you're testing in an email message using the subject-text format `<application_name> crash logs from <your_name>` (for example, `MyTestApp crash logs from Anna Haro`) to the application's developer.

Glossary

active build configuration The build configuration used to build a product. See also **build configuration**.

active SDK The SDK used to build a product and the runtime environment on which the product is to run. See also **SDK family**.

Apple Developer Connection (ADC) A program that lets you access the iPhone Dev Center, which provides access to documentation, tools, and resources needed to develop iPhone applications. See also **iPhone Developer Program**.

application ID A string that identifies an iPhone application or a set of iPhone applications from one vendor. They are similar to bundle identifiers. This is an example application ID: `GFWOTNXFIY.com.mycompany.MyApp`.

base SDK Project setting that specifies the default SDK to use when building the project's targets. Targets can override this setting with the iPhone OS Deployment Target build setting.

build configuration A named collection of build settings that build one or more products in a project in different for specific purposes—for example, for debugging or for release.

certificate signing request (CSR) File that contains personal information used to generate a development certificate. Certificate signing requests are created by the Keychain Access application.

code completion A shortcut that automatically suggests likely completions as you type an identifier or a keyword. The suggestions are based on the text you type and the surrounding context within the file.

development certificate File that identifies an iPhone application developer. Xcode uses development certificates to sign application binaries.

instrument A data-gathering agent developed using the Instruments application. Instruments collect performance information about an application or an entire system.

Instruments application A performance analysis tool used to gather and mine application-performance data.

iPhone Dev Center An Apple developer center that provides all the resources needed to develop iPhone applications. Access to this developer center requires an ADC membership. See also **Apple Developer Connection**.

iPhone Developer Program A program that allows you to develop iPhone applications, test them on devices, and distribute them to your customers through the App Store.

iPhone Simulator application An application that simulates the iPhone OS runtime environment and user experience in Mac OS X for testing iPhone applications in early stages of development.

Program Portal A restricted-access area of the iPhone Dev Center that allows you to configure devices to test your iPhone applications

project window A window that displays and organizes the files that make up an Xcode project.

provisioning profile A file that allows applications in development to be installed on an iPhone OS–based device. It contains one or more development certificates, an application ID, and one or more device IDs

SDK family Group of SDK releases used to build software products for a particular Apple platform. The available SDK families are iPhone Device SDK, iPhone Simulator SDK, and Mac OS X SDK.

test provisioning profile A provisioning profile issued to users not on an iPhone application developer team. It allows them to install and test applications that have not been published to the App Store.

Xcode A set of tools and resources used to develop iPhone and Mac applications.

Xcode application The main application of the Xcode integrated development environment (IDE). It manages the other applications that are part of Xcode and provides the main user interface used to develop software products.

Document Revision History

This table describes the changes to *iPhone Development Guide*.

Date	Notes
2009-01-06	Made minor content additions.
	Explained that iPhone Simulator binaries can be used on only one release of the simulator.
2008-11-14	Added information about new iPhone Simulator features.
	Added "Adding the iTunes Artwork to Your Application" (page 52).
	Added information about the Simulate Memory Warning and Toggle In-Call Status Bar commands to "Manipulating the Hardware" (page 31).
	Added "Core Location Functionality" (page 33).
	Added information about using static libraries in iPhone applications to "Creating an iPhone Project" (page 12).
2008-10-15	New document that describes how to develop iPhone applications using Xcode.
	Incorporates content previously published in *iPhone OS Programming Guide* and *iPhone Simulator Programming Guide*.