Xcode Overview

Developer

Contents

About Xcode

Xcode is Apple's integrated development environment (IDE) that you use to build apps for Apple products such as the iPad, iPhone, and Mac. Xcode provides tools to manage your entire development workflow—from creating your app, to testing, optimizing, and submitting it to the App Store.

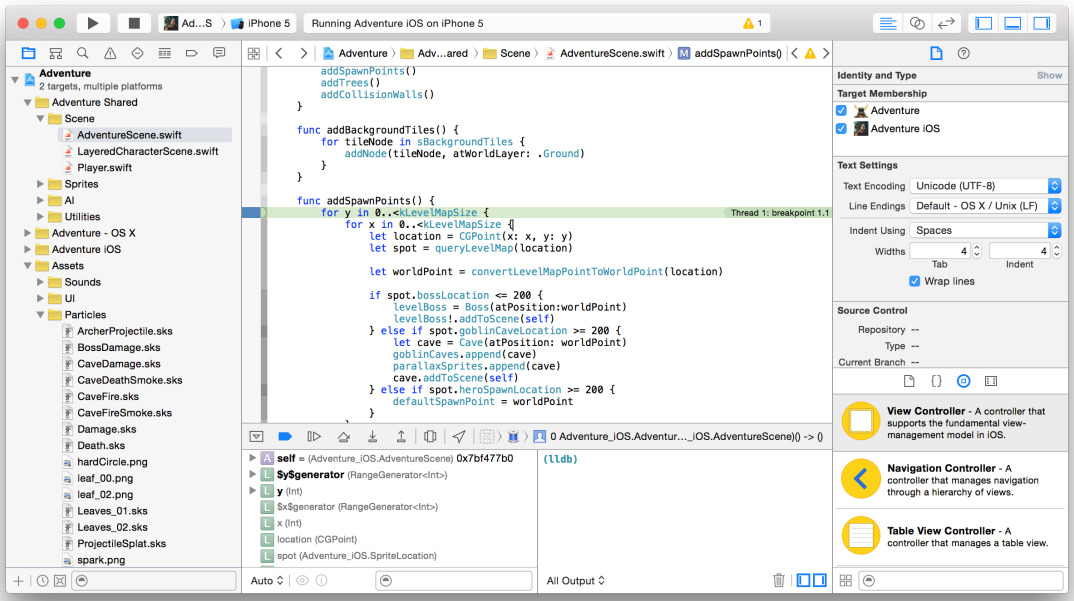
At a Glance

Use the App Store app on your Mac to download Xcode. It's free. After you download Xcode, it automatically appears in Launchpad, where you can click the icon for Xcode to launch it.

Single-Window Interface

The Xcode interface integrates code editing, user interface design, asset management, testing, and debugging within a single workspace window. The window reconfigures its content as you work. For example, select a file in one area, and an appropriate editor opens in another area. Select a symbol or user interface object, and its documentation appears in a nearby pane.

You can focus on a task by displaying only what you need, such as only your source code or only your user interface layout. Or you can work with your code and UI layout side by side. You can further customize your environment by opening multiple windows and multiple tabs per window.

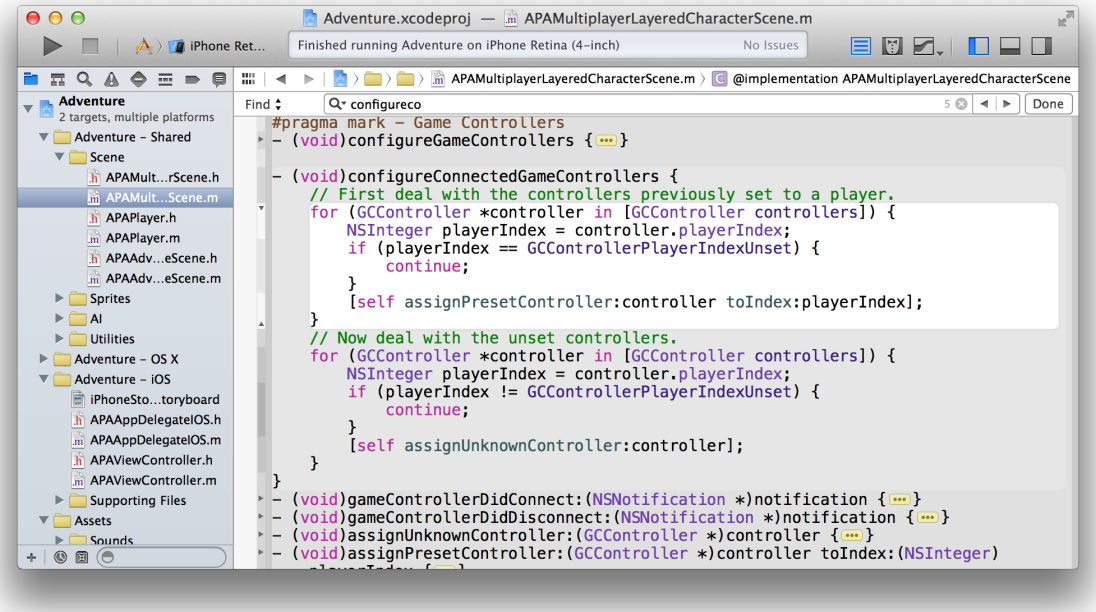


Relevant Chapters: Develop Your App in the Workspace Window (page 19), Maintain Your Code and Other Resources in Projects or Workspaces (page 30).

Assisted Source Code Editing

Whether you are using Objective-C, Swift, C, C++, or a mix, Xcode checks your source code as you type it. When Xcode notices a mistake, the source code editor highlights the error and when possible, offers to fix it. Xcode speeds up your typing with intelligent code completion. Reduce your typing further with ready-to-use code snippets and source file templates, either the ones provided or ones you add. With Swift, Playgrounds let you experiment with code without building and running your app. For more information on playgrounds, see Playground Help.

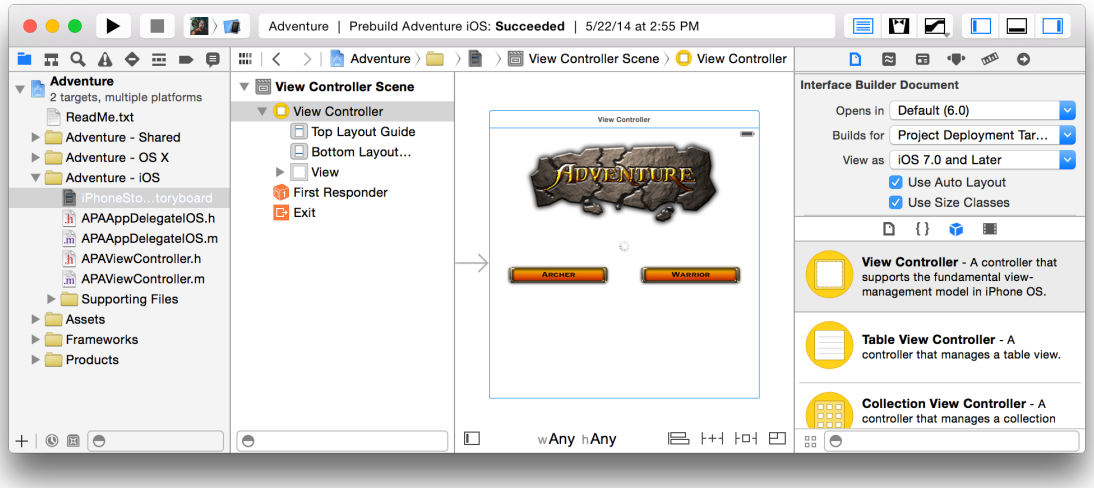
You can easily configure the source editor to display multiple views of the same file or to view multiple related files at once. Search-and-replace and refactoring operations help you make extensive changes to your code quickly and safely. With these and other capabilities, Xcode makes it easier for you to write better code faster than you thought possible.



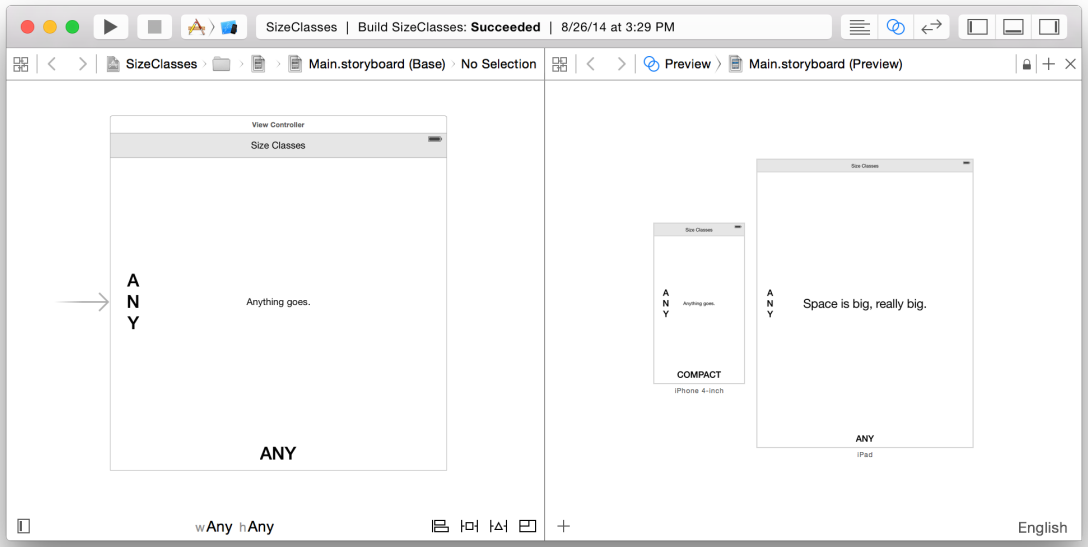
Relevant Chapter: Write Code in the Source Editor (page 38).

Graphical UI Design

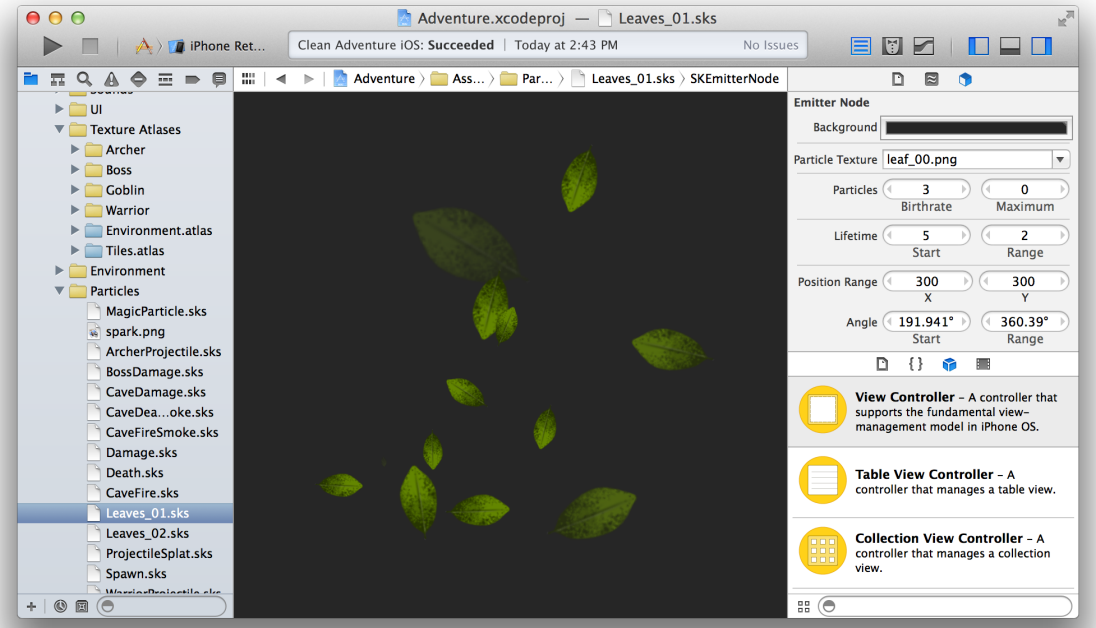
Interface Builder is a visual design editor that's integrated into Xcode. Use Interface Builder to create the user interfaces of your iOS or Mac apps by assembling windows, views, controls, menus, and other elements from a library of configurable objects, or from ones you create. Use Storyboards to specify the flow of your applications and the transitions between scenes. Then graphically connect the objects and transitions to your implementation code.



With the Auto Layout feature, define constraints for your objects so that they automatically adjust to screen size, window size, and localization. With Size Classes, tune your mobile UI for any combination of screen size and orientation: customizing Auto Layout constraints, adding or removing views, and even changing the font.



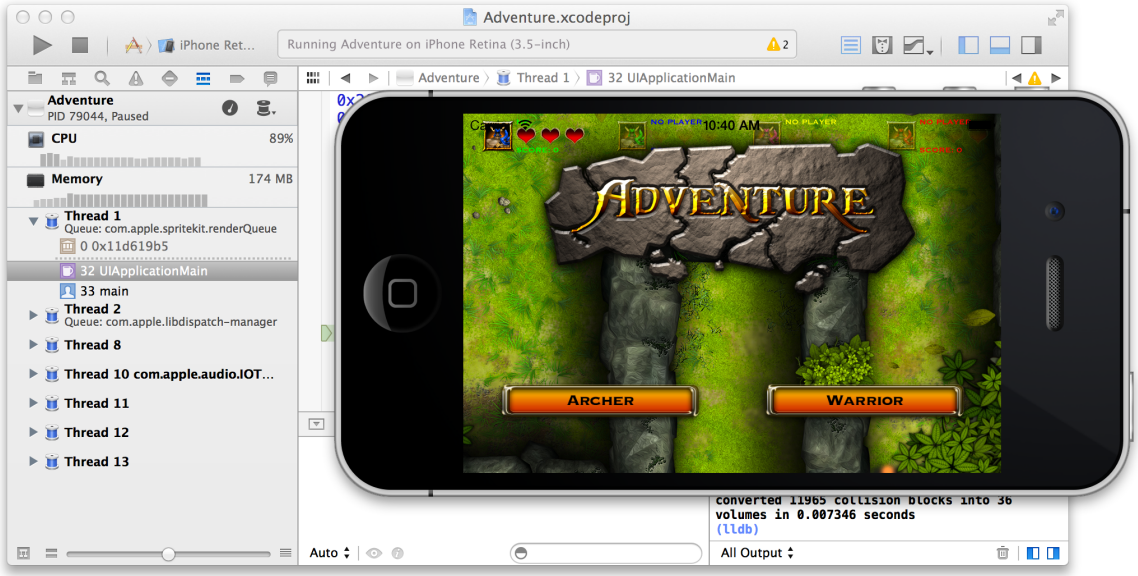
The asset catalog in Xcode helps you manage the many images you'll use for your app's user interface—items such as icons, custom artwork, and launch images for iOS devices. With the particle emitter editor in Xcode, you can enhance your iOS or Mac game by adding animation effects involving moving particles such as snow, sparks, and smoke. For Mac apps, the SceneKit editor helps you work with scenes created in 3D authoring tools and exported as digital asset exchange (DAE) files.



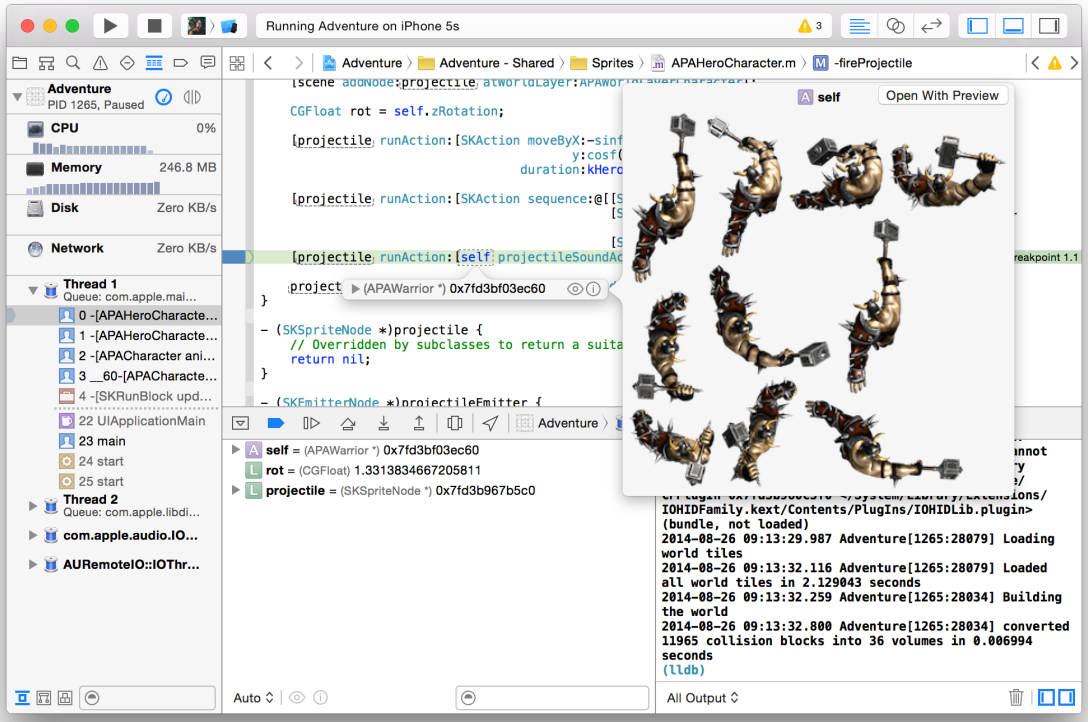
Relevant Chapters: Build a User Interface (page 61) and Add Icons, Images, and Effects (page 84).

Integrated Debugging

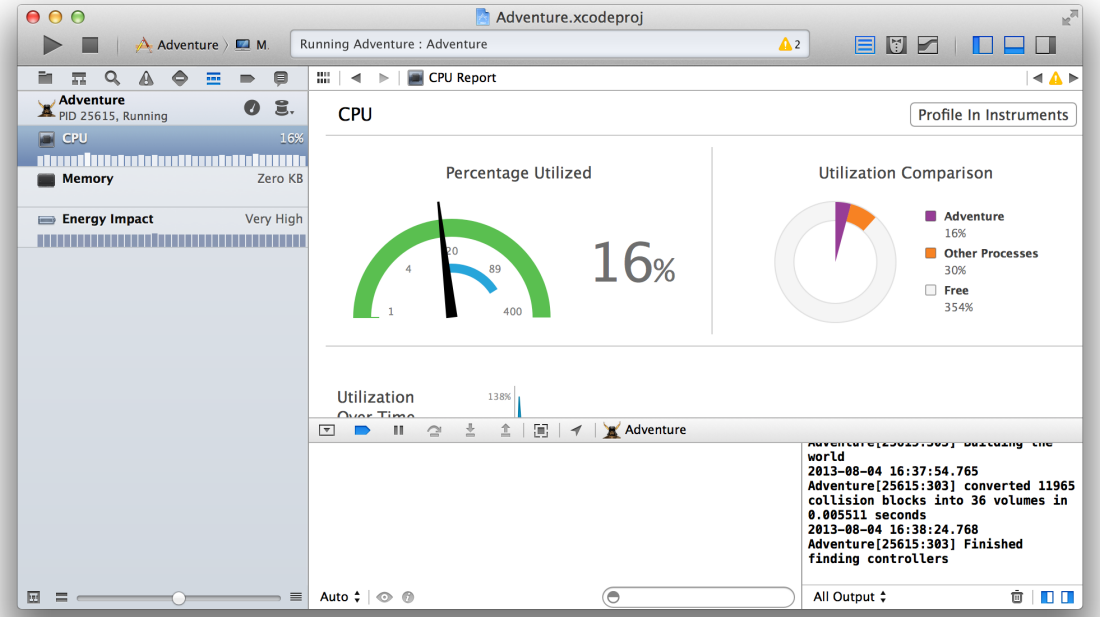
When Xcode launches your app in debug mode, it immediately starts a debugging session. If you are running an iOS app, Xcode launches it either in iOS Simulator or on an iOS device connected to your Mac. If you are running a Mac app, Xcode launches it directly on your Mac.



You can debug your app directly within the source editor. View the contents of an object by moving your mouse over a variable name, and then use Quick Look to inspect a particular value. The debug area and the debug navigator let you carefully control the execution of your app while you examine the code. For finer control, the console gives command line access to the debugger.



Debug gauges display your app's resource consumption to help you identify problems before your users do.

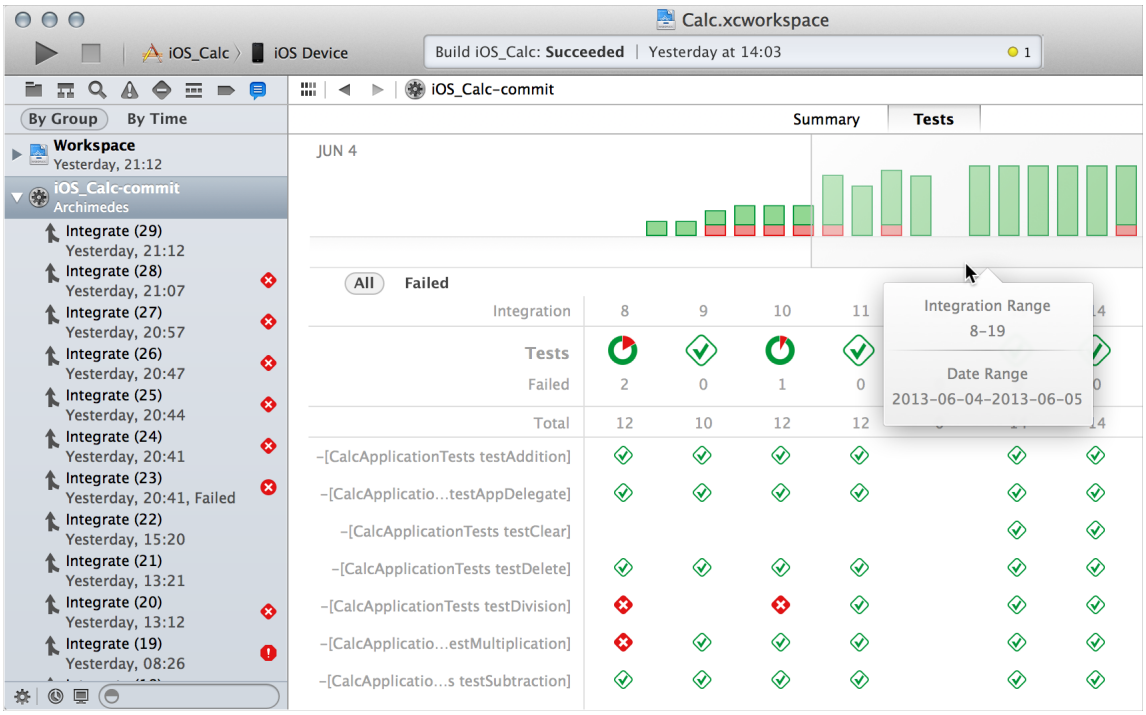


Relevant Chapters: Run Your App (page 91) and Debug Your App (page 100).

Testing and Continuous Integrations

To help you build a better app, Xcode includes a testing framework for functional and performance testing. You write the tests and use the test navigator to run those tests and see the results. You test code functionality unit tests. Performance tests make sure important parts of your app don't leave the user waiting. Set triggers for running tests on a regular basis so you catch regression bugs in code and in performance.

Run your tests in the test navigator, look at the results, and make any changes needed to pass the tests. You can use the Xcode service, available in OS X Server, to automate the execution of tests. From Xcode on your development Mac, you create *bots* that run on a separate server to execute your unit tests either periodically or on every source code commit.



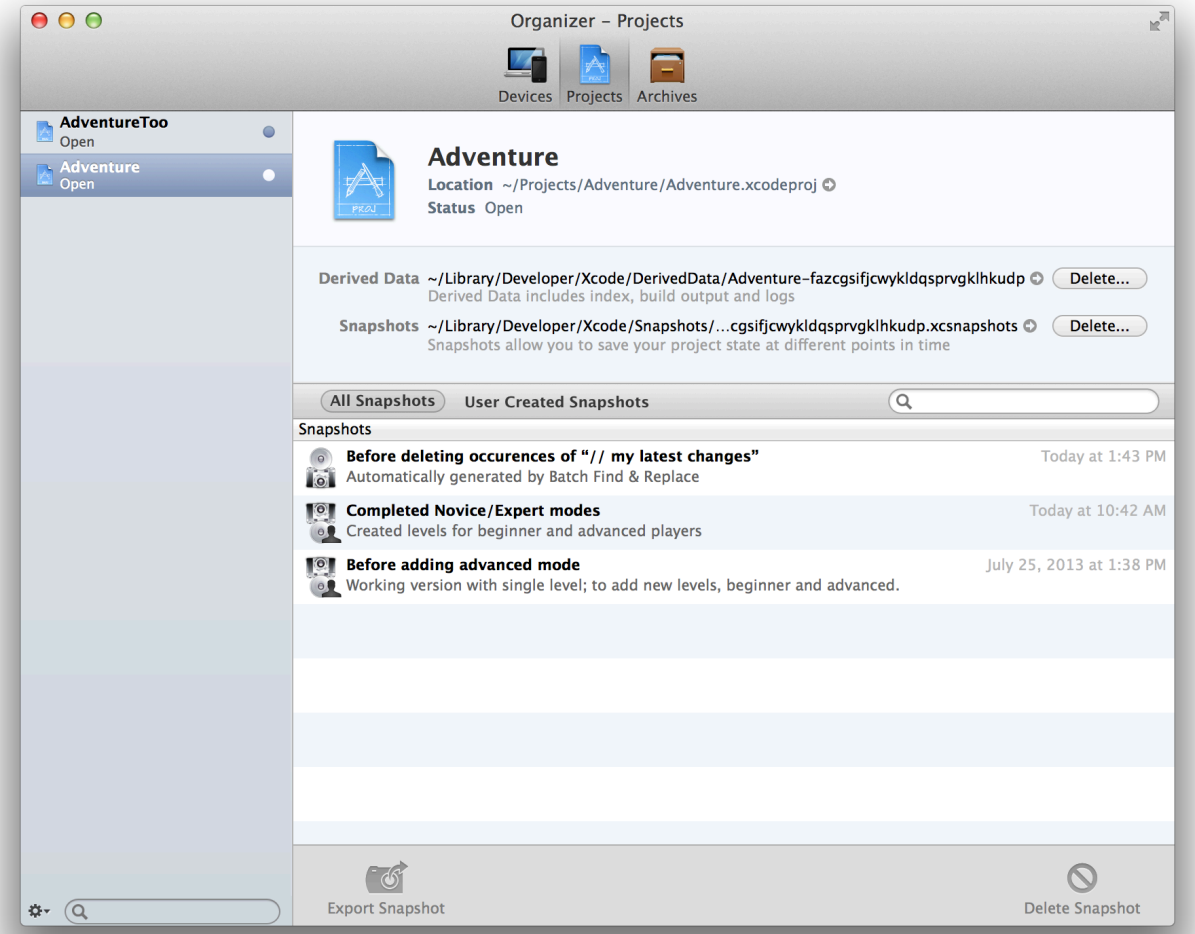
In addition to running unit tests, bots automatically perform static analysis on your code, build your app, and archive it for distribution to testers or the App Store. While performing these continuous integrations of your app, bots report build errors and warnings, static analyzer problems, and unit test failures.

Relevant Chapter: Test Your App (page 112).

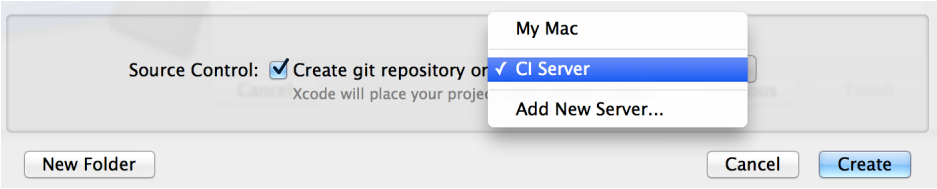
Automatic Saves, Project Snapshots, and Source Control Management

While you work, Xcode automatically saves changes to source and project files. This feature requires no configuration, because Xcode continuously tracks your changes and saves them. You can revert a file to a previous state with Undo and Revert Document commands.

You can revert an entire project to a previous snapshot of a known working version with the Restore Snapshot command. Snapshots make it easy to back up the current version of your project. You create a snapshot by choosing File > Create Snapshot. You can also set Xcode to automatically create snapshots before you perform any mass editing operations and as part of a workflow.



To keep track of changes at a fine-grained level, use the Xcode source control management features. Xcode supports two popular source control systems: Git and Subversion. You can access remote Git and Subversion source code repositories, and you can create local Git repositories. Using the Xcode service, available with OS X Server, you can host Git repositories on your own server.

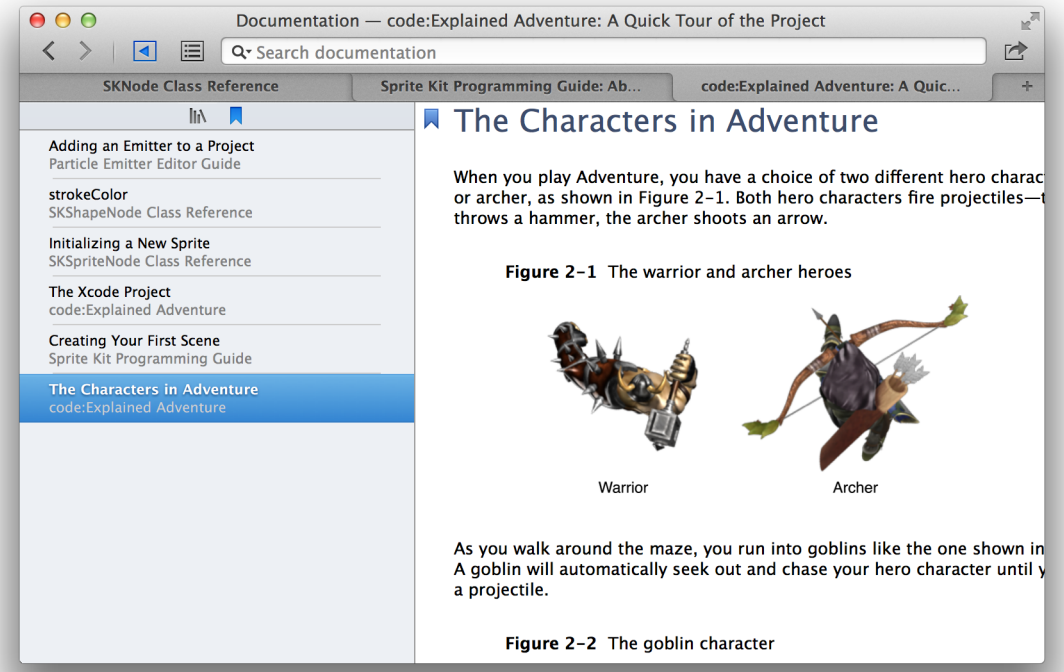


Relevant Chapter: Save and Revert Changes (page 115).

Integrated Documentation

While you're coding, Xcode makes detailed technical information available at your fingertips. When you want it, Quick Help keeps concise API information always in view, and Xcode application help is always close at hand with step-by-step instructions for performing common Xcode tasks. Xcode includes extensive documentation for using Xcode, and it provides comprehensive SDK documentation, including programming guides, tutorials,

sample code, detailed framework API references, and video presentations by Apple engineers. All of these resources are viewable from the Xcode documentation viewer. As updated documentation becomes available, it downloads automatically in the background.



Relevant Chapter: Learn More About Xcode (page 124).

App Distribution to Testers and the App Store

Most of your development time is spent on coding tasks, but to develop for the App Store, you need to perform a number of administrative tasks throughout the lifetime of your app. In addition to Xcode, you'll use the Member Center web tool to manage developer program accounts and entitlements, and you'll use the iTunes Connect web tool to check the status of your contracts, set up tax and banking information, obtain sales and finance reports, and manage metadata about the app.

Xcode project configurations help prepare your app for distribution to beta testers and for submission to the App Store. Submitting your app is a multistep process that begins when you sign into iTunes Connect and supply necessary product information. In Xcode, you create an archive of your project and submit it to the store. When your app is approved, you use iTunes Connect to release it by setting the date. (If you are distributing your Mac app outside the store, you follow a slightly different process.)

Relevant Guide: *App Distribution Guide*.

See Also

Many of the screenshots used to illustrate this document are taken from the *Adventure* Xcode project described in *code:Explained Adventure*. To explore the Xcode features described in this guide on your Mac, obtain Xcode from the App Store, then download the Adventure project by clicking either link in this paragraph.

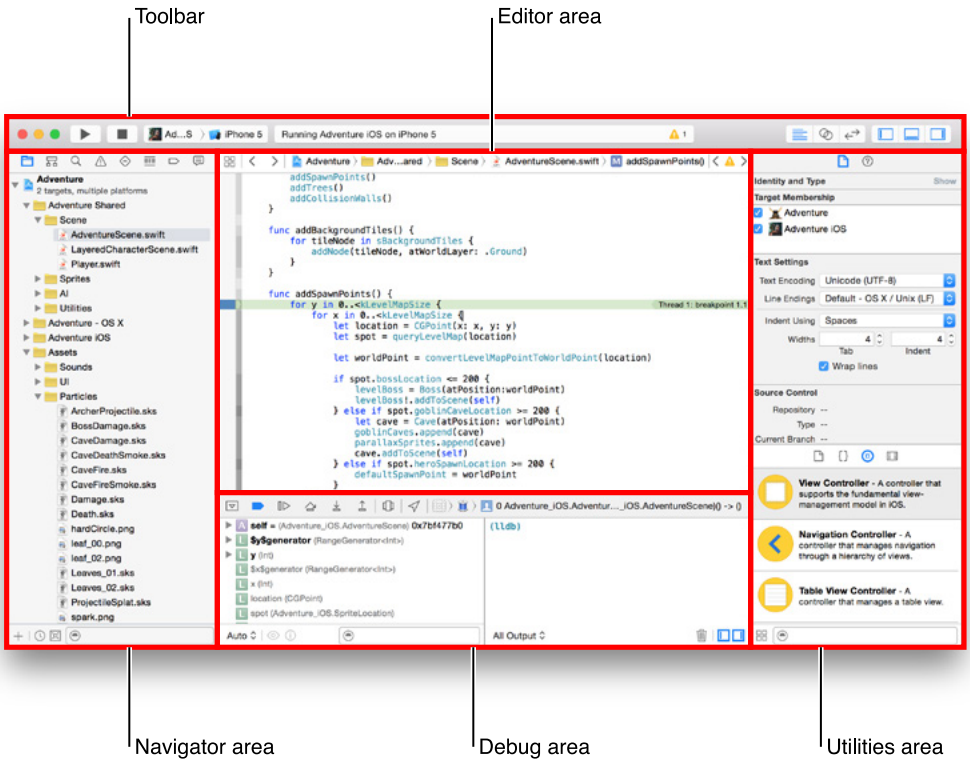
This guide introduces you to the major features and capabilities of Xcode. For a hands-on introduction to using Xcode, read either *Start Developing iOS Apps Today* or *Start Developing Mac Apps Today*. In each document, you use Xcode to create a simple app, and you learn the basics of programming with Objective-C.

Develop Your App in the Workspace Window

Perform your core development tasks in the Xcode workspace window, your primary interface for creating and managing projects. A project is the main unit of development in Xcode. It includes all the elements needed to build your app, framework, plug-in, or other software product. It also maintains the relationships between those elements. For more detail on projects, see A Project Is a Repository of Files and Resources for Building Apps (page 31).

The workspace window automatically adapts itself to the task at hand, and you can further configure the window to fit your work style. You can open as many workspace windows as you need.

The components of the workspace window are shown in the following figure.



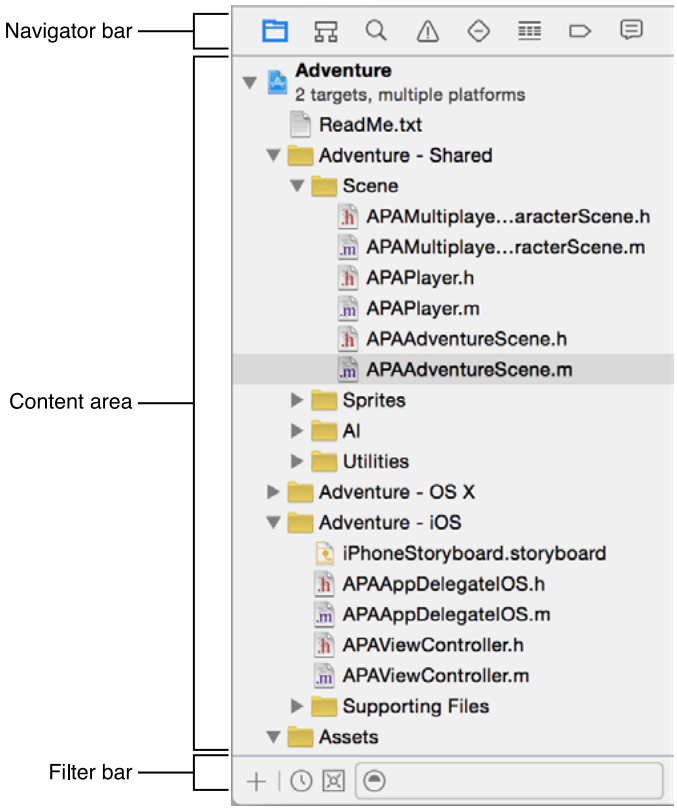
The workspace window always includes the *editor area*. When you select a file in your project, its contents appear in the editor area, where Xcode opens the file in an appropriate editor. For example, in the figure above, the editor area contains `AdventureScene.swift`, a swift code file that is selected in the Navigator area on the left of the workspace window.

The workspace window displays up to three optional areas used in performing different tasks in the development lifecycle. Hiding areas not in use can help you focus on your current task. You can hide or show these optional areas by using the workspace configuration buttons on the far right side of the toolbar:

- Show and hide the **navigator area**. Use this area for navigating all facets of your project, including files, symbols, breakpoints, build issues, tests, breakpoints, and build reports. You can also search for any string in your project.
- Show and hide the **debug area**. Use this area for viewing variables, interacting with the debugger console, and controlling the execution of your application.
- Show and hide the **utilities area**. Use this area to inspect or modify attributes of files, graphical user interface elements, sprites, and other elements in your project. Also use it to access a library of ready-made resources. See Access Resources and Inspect Elements in the Utilities Area (page 25).

Navigate Your Workspace

Access files, symbols, unit tests, diagnostics, and other facets of your project from the navigator area. In the **navigator bar,** you choose the navigator suited to your task. The **content area** of each navigator gives you access to relevant portions of your project, and each navigator's **filter bar** allows you to restrict the content that is displayed.



Choose from these options in the navigator bar:

- **Project navigator.** Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.
- **Symbol navigator.** Browse the symbols in your project as a list or hierarchy. Buttons on the left of the filter bar let you limit the shown symbols to a combination of only classes and protocols, only symbols in your project, or only containers.
- **Find navigator.** Use search options and filters to quickly find any string within your project.
- **Issue navigator.** View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.
- **Test navigator.** Create, manage, run, and review unit tests.

- **Debug navigator.** Examine the running threads and associated stack information at a specified point or time during program execution.
- **Breakpoint navigator.** Fine-tune breakpoints by specifying characteristics such as triggering conditions.
- **Report navigator.** View the history of your build, run, debug, continuous integration, and source control tasks.

Typing text in the filter bar text input field shows only the items in the content area containing the search term. Most navigators show buttons on the left side of the filter bar used to further restrict what content is shown. Some filter bars have an Add button (+) on the left that you use to add an element to the content area. The button on the left of the filter bar in the Report navigator ( ) is used for interacting with bots. Using Bots from the Report navigator is covered in more detail in Manage and Monitor Bots from the Report Navigator.

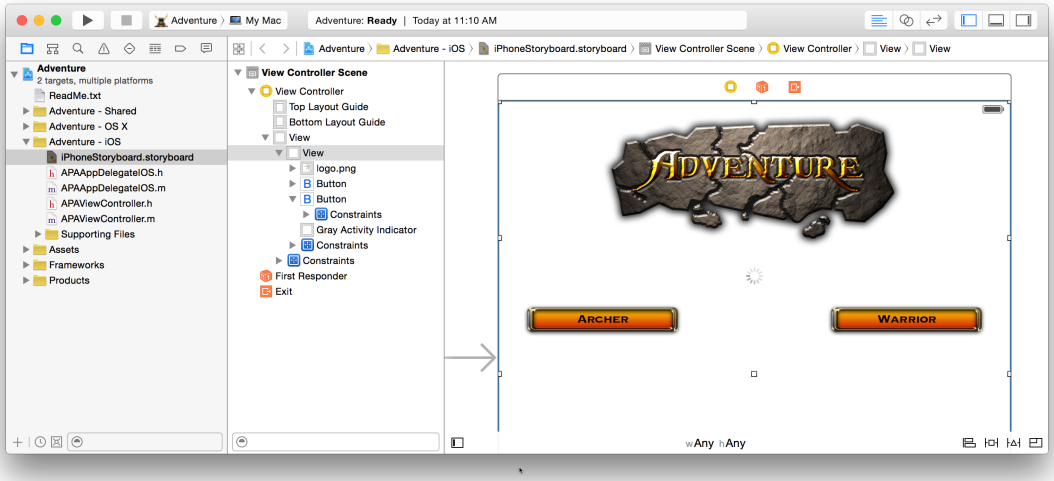
Select files in the content area to view or edit them.

Edit Your Project Files

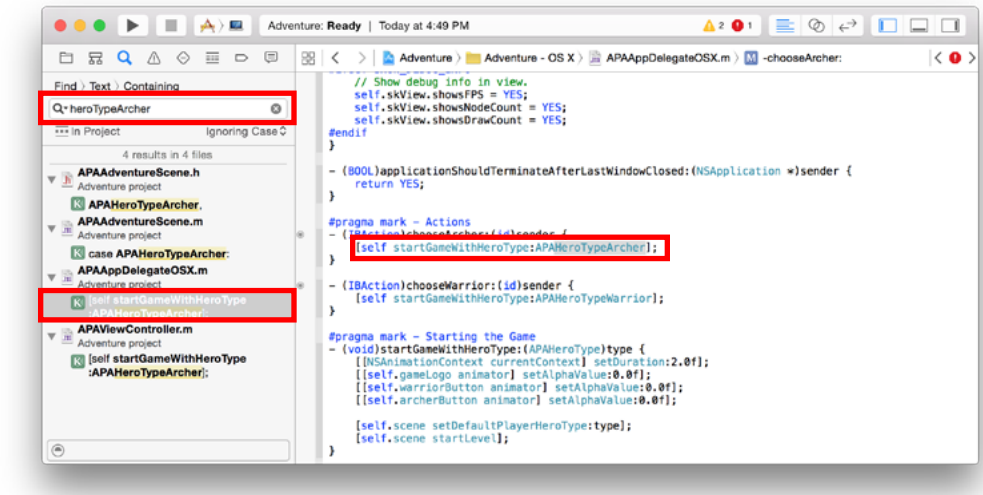
Most development work in Xcode occurs in the editor area, the main area that is always visible within the workspace window. The editors you use most often are:

- **Source editor.** Write and edit source code.
- **Interface Builder.** Graphically create and edit user interface files.
- **Project editor.** View and edit how your apps should be built, such as by specifying build options, target architectures, and app entitlements.

When you select a file from the content area of a navigator, Xcode opens the file in an appropriate editor. In the screenshot, the file `iPhoneStoryboard.storyboard` is selected in the project navigator, and the file is open in Interface Builder. Interface Builder is showing both the outline view on the left and the canvas on the right. For more information, see Build a User Interface (page 61). (The optional utilities and debug areas are hidden to maximize space for the navigator and editor.)



The following screenshot shows a number of search results appearing in the find navigator's content area. One of the results is selected, and its text string appears in the source editor.

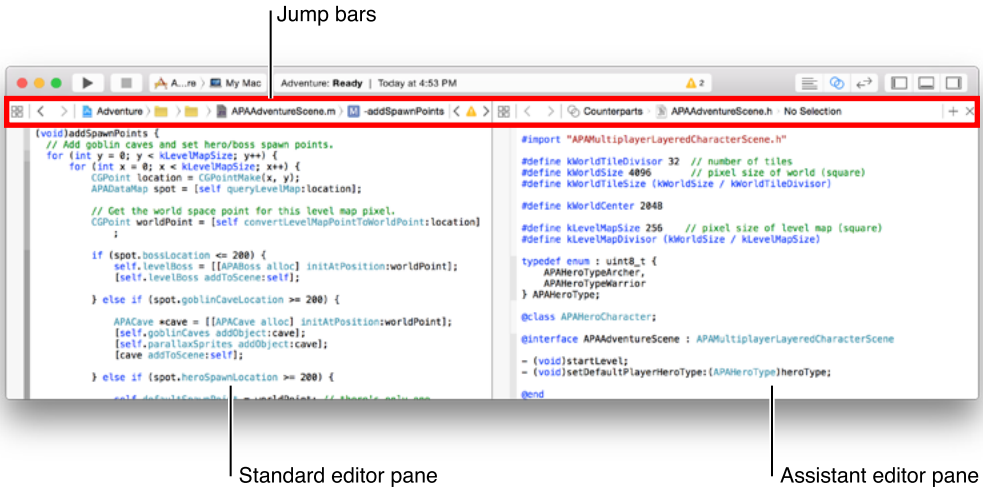


Configure the editor area for a given task with the editor configuration buttons on the right side of the toolbar:

- **Standard editor.** Fills the editor area with the contents of the selected file.

- **Assistant editor.** Presents a separate editor pane with content logically related to content in the standard editor pane. You can also change the content.
- **Version editor.** Shows the differences between the selected file in one pane and another version of that same file in a second pane. This editor works only when your project is under source control.

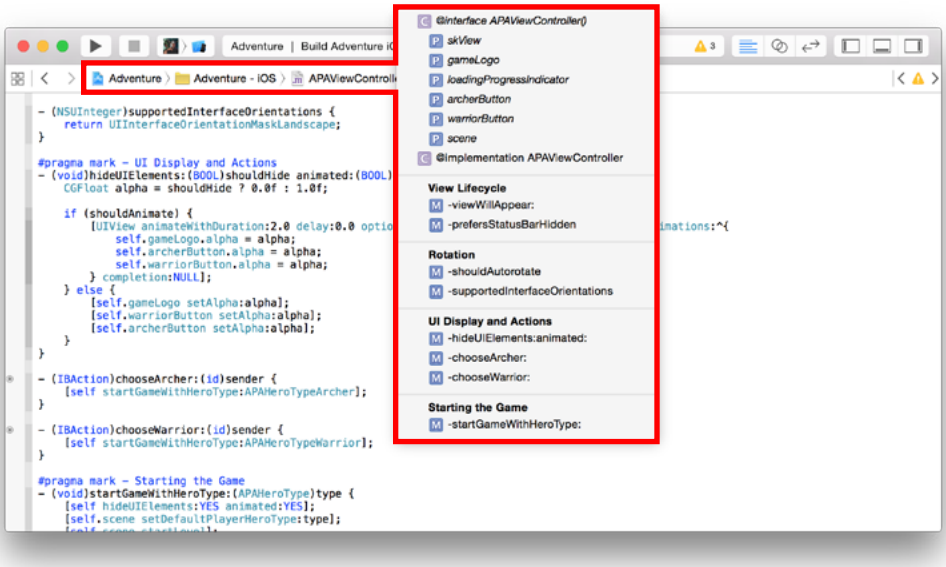
This screenshot shows an implementation file, `APAAdventureScene.m`, open in the standard editor pane. The three optional workspace areas—navigator, debugger, and utilities—are hidden to maximize the editor's content display. Within the source code editor, the assistant pane displays the implementation file's associated header file, `APAAdventureScene.h`.



Every editor or assistant editor pane includes a jump bar—an interactive, hierarchical mechanism for navigating directly to items at any level in your project. The configuration and behavior of the jump bar is customized for its context. The basic jump bar configuration includes three components:

- The related items menu ( ) offers additional selections relevant in the current context, such as recently opened files or the interface (`.h`) file for an implementation (`.m`) file you are editing.
- Previous and Next buttons ( ) allow you to step back and forth through your navigation history.
- The hierarchical path menu allows you to change what is shown in the editor or assistant editor pane by navigating to a new item. It is made up of one or more segments depending on what part of the path you click.

Click a segment in the hierarchical path menu to see a pop-up menu of related items. For example, if the segment identifies the name of the project, you use the jump bar to navigate to and open any file within the project. If the segment identifies the name of a folder, you can use the jump bar to open a file within the folder. If the segment identifies the name of a source file, you use the jump bar to show and select a symbol within the currently open file.

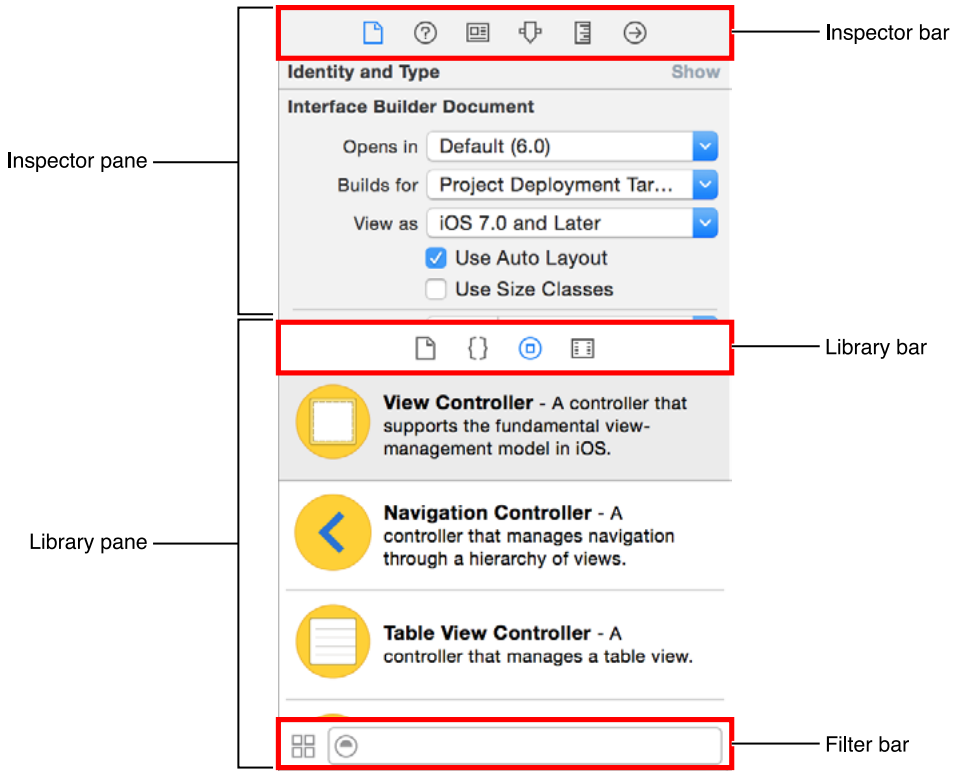


Access Resources and Inspect Elements in the Utilities Area

The utilities area on the far right of the workspace window gives you quick access to these resources:

- Inspectors, for viewing and modifying characteristics of the file open in an editor
- Libraries of ready-made resources for use in your project

The top pane of the utilities area displays inspectors. The bottom pane gives you access to libraries.



Use the **inspector bar** to choose the inspector best suited to your current task. Two inspectors are always visible in the inspector bar (additional inspectors are available in some editors):

- **File inspector.** View and manage metadata for the selected file. Typically you will localize storyboards and other media files and change settings for user interface files.
- **Quick Help.** View details about a symbol, an interface element, or a build setting in the file. For example, Quick Help displays a concise description of a method, where and how the method is declared, its scope, the parameters it takes, and its platform and architecture availability.

Use the **library bar** to access ready-to-use libraries of resources for your project:

- **File templates.** Templates for common types of files and code constructs.
- **Code snippets.** Short pieces of source code for use in your software, such as class declarations, control flows, block declarations, and templates for commonly used Apple technologies.
- **Objects.** Items for your app's user interface.
- **Media.** Files containing graphics, icons, sound files, and the like.

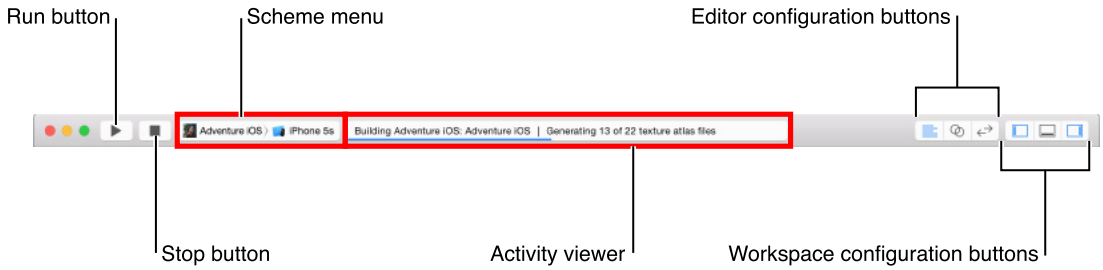
To use a library, drag it directly to the appropriate area. For example, to use a code snippet, drag it from the library to the source editor; to create a source file from a file template, drag its template to the project navigator.

To restrict the items displayed in a selected library, type relevant text into the text field in the **filter bar**. For example, type “button” in the text field to show all the buttons in the Objects library.

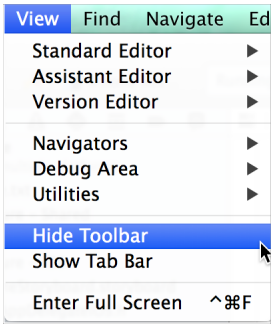
Manage Common Tasks with the Workspace Toolbar

The toolbar at the top of the workspace window provides quick access to frequently used commands. The **Run button** builds and runs your products. The **Stop button** terminates your running code. The **Scheme menu** lets you configure the products you want to build and run. The **activity viewer** shows the progress of tasks currently executing by displaying status messages, build progress, and other information about your project.

You’ve seen how the **editor configuration buttons** let you configure the editor area, and you’ve seen how the **workspace configuration buttons** hide or show the optional navigator, debug, and utilities areas.

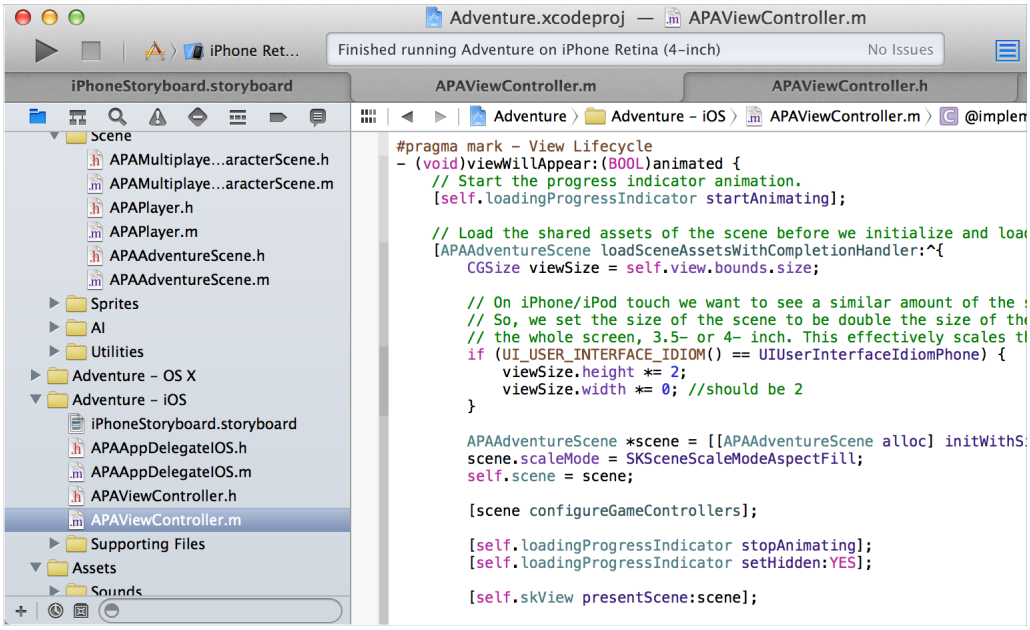


The **View menu** includes commands to hide or show the toolbar.



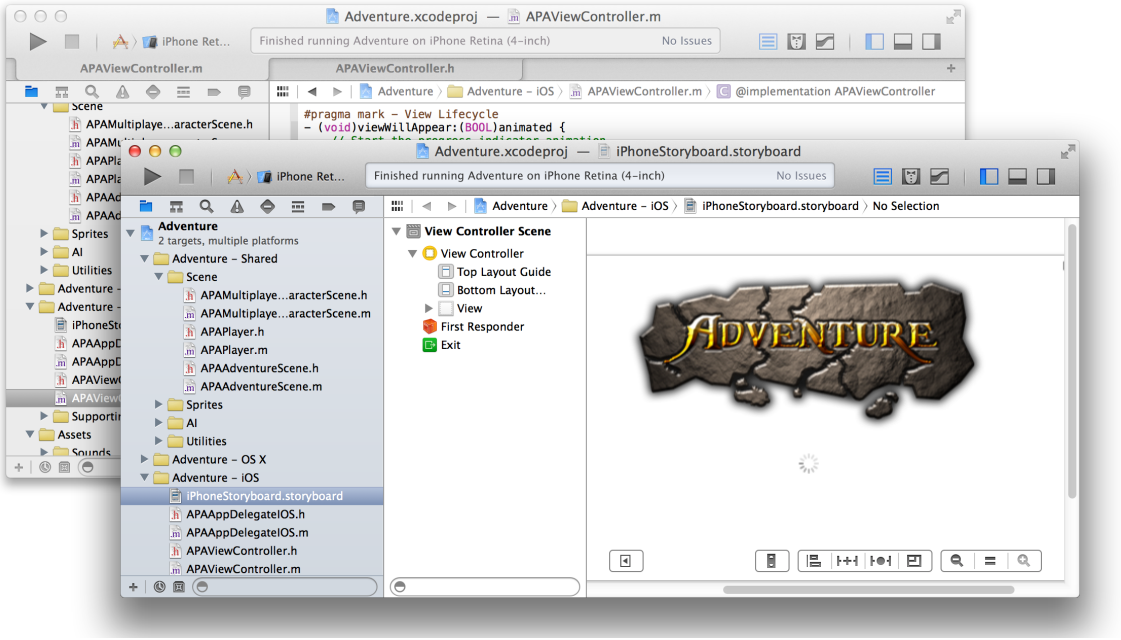
Work in Multiple Tabs or Multiple Windows

Use Safari-style tabs to implement multiple, workflow-specific layouts of the workspace window. For example, in the screenshot below the active tab is showing the contents of an implementation file (`APAViewController.m`) in the source editor. The tab on the right is for the related header file (`APAViewController.h`), and the one on the left is for the app storyboard (`iPhoneStoryboard.storyboard`.) Clicking a tab makes it the active editor.



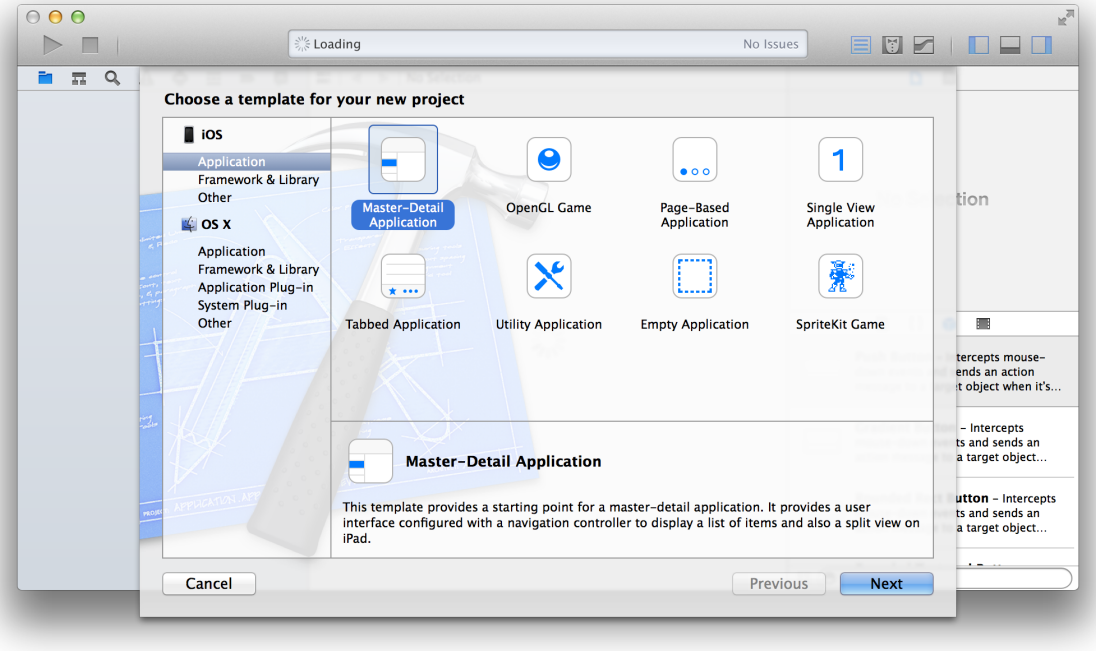
The View menu contains commands to show and hide the tab bar. To create a tab, choose File > New > Tab. To remove a tab, move the pointer to the tab and click its close button.

Create multiple workspace windows by choosing File > New Window. Each tab or window can be customized independently of the others, for example, by showing and hiding the utilities area with the Hide/Show Utilities button ( ) in the toolbar.

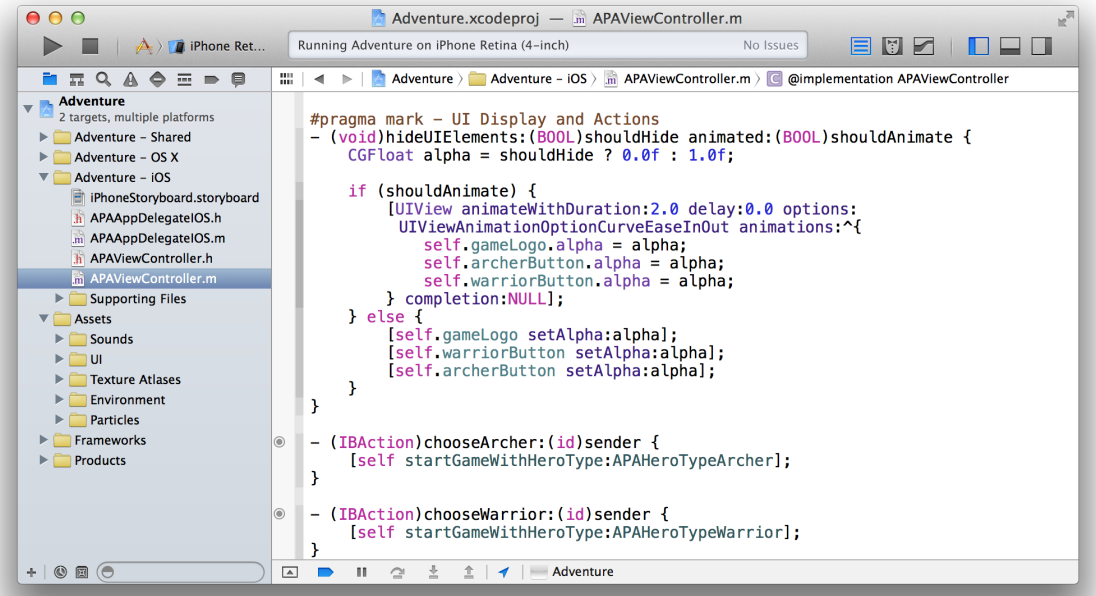


Maintain Your Code and Other Resources in Projects or Workspaces

Apps you create in Xcode require a **project**, which keeps the necessary files and resources organized. You start a project by choosing File > New > New Project. Xcode opens a new workspace window and displays a dialog in which you choose a project template. Xcode provides built-in templates for developing common styles of iOS and Mac apps. These templates include essential project configuration and files that help you start your development effort quickly.



View the names of project files in the project navigator. When you select a file in the project navigator, the file's contents appear in the appropriate editor or viewer. The screenshot below shows the Adventure project. An implementation file (`APAViewController.m`) is selected in the project navigator, and the file's contents appear in the source editor.

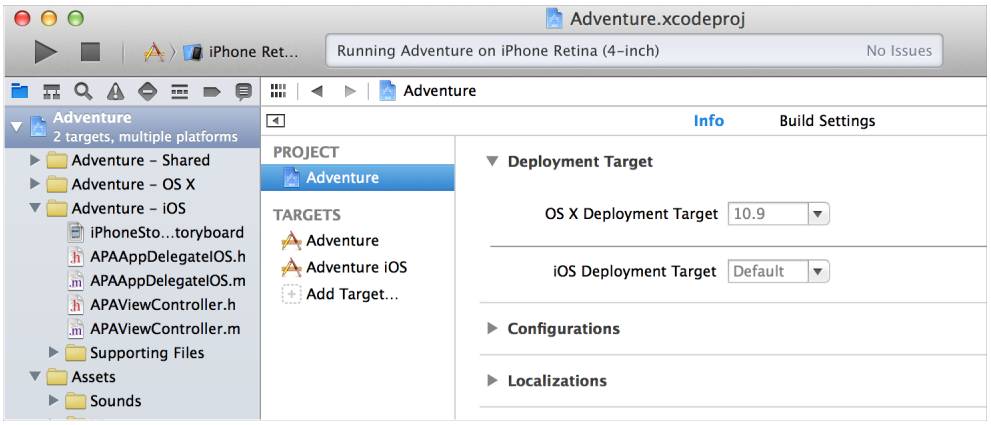


A Project Is a Repository of Files and Resources for Building Apps

A project contains the elements needed to build one or more apps (or other software products, such as command-line tools and plug-ins). The project also maintains the relationships among these elements. These elements include:

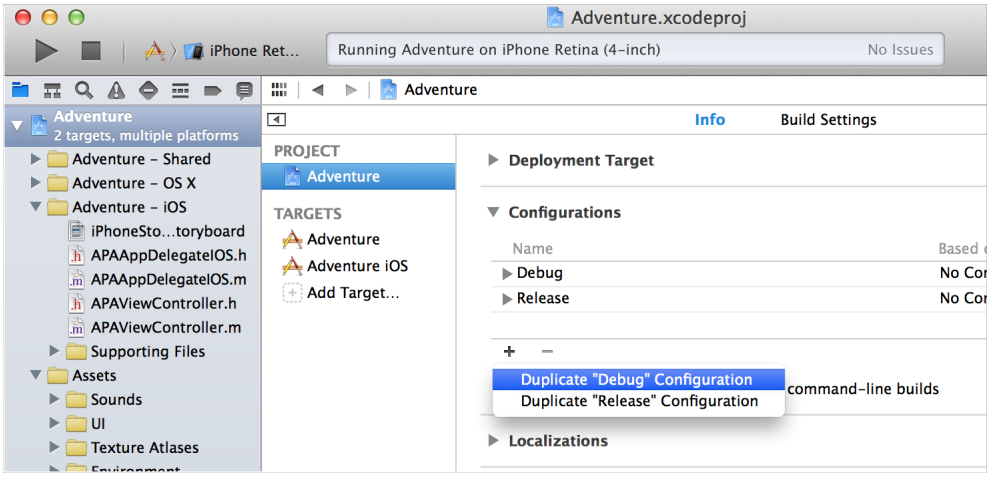
- References to source code files (including implementation files and header files where appropriate), libraries and frameworks, image files, and user interface files
- Groups, for organizing files in the project navigator
- Project-level build configurations
- Targets, each of which produces a single app

By selecting the project name in the project navigator, you open the project editor. You can use the project editor to specify every aspect of how your apps should be built, from the version of the software development kit (SDK) to specific compiler options. In this screenshot, the Adventure project is selected in the project navigator *and* in the project editor. The project editor displays the Info pane for the Adventure project.



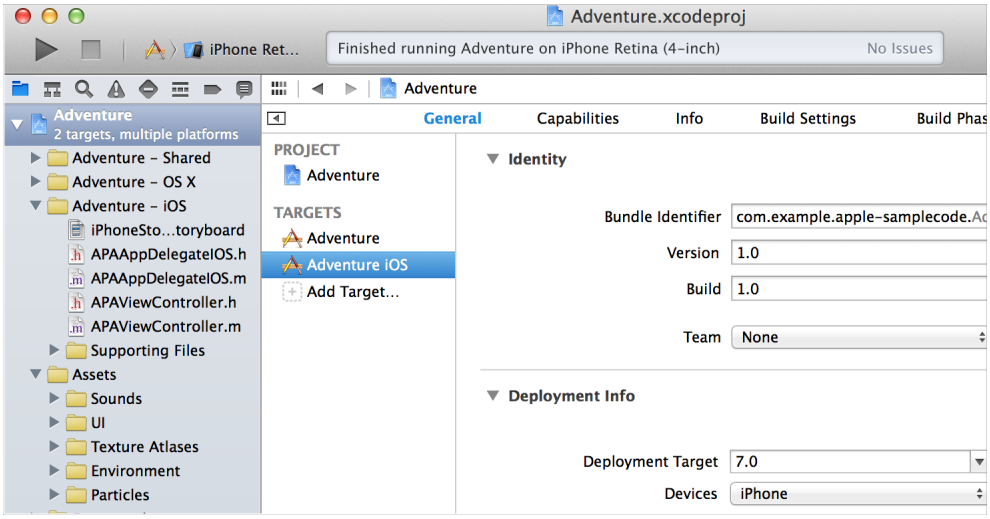
When you create a project, Xcode provides two standard project-level build configurations: debug and release. These configurations differ mostly in whether they include debug information and in the degree to which each build is optimized. These two build configurations are probably sufficient for your product development needs. Most developers never need to change the values of the vast majority of build settings.

To add more build configurations, open the project editor, duplicate one of the project's existing configurations, and then modify its settings. For example, you might configure a build that's fully optimized but that also includes debug information in order to debug your optimized code.



Apply App-Specific Settings to a Target

Every project contains at least one target. A target specifies a product to build, such as an iOS or Mac app. Select a target in the project editor to view and modify the target's settings. In the screenshot below, the *Adventure iOS target* of the *Adventure project* is selected in the *project navigator,* and the *Adventure iOS target* is selected in the *project editor*. The project editor displays the General pane for the target.



The General pane for a target shows basic settings that you occasionally check and possibly edit. You typically assign values for these settings elsewhere during the app development process, for example, in dialogs that appear when you create a new project.

For an iOS app, the General pane contains target settings for:

- The bundle identifier, a string that identifies the app to the operating system and to the App Store
- The version number under which to publish the app
- The build number, which identifies a particular build of the app
- The name of your Apple Developer Program development team
- The deployment target, which is the earliest iOS version on which the app runs
- The devices for which to build the app
- The main user interface file to load when the app launches
- The user interface orientations (portrait, upside down, landscape left, landscape right) that the app supports

For a Mac app, the General pane contains target settings for:

- The application category, for classifying the app on the Mac App Store
- The bundle identifier

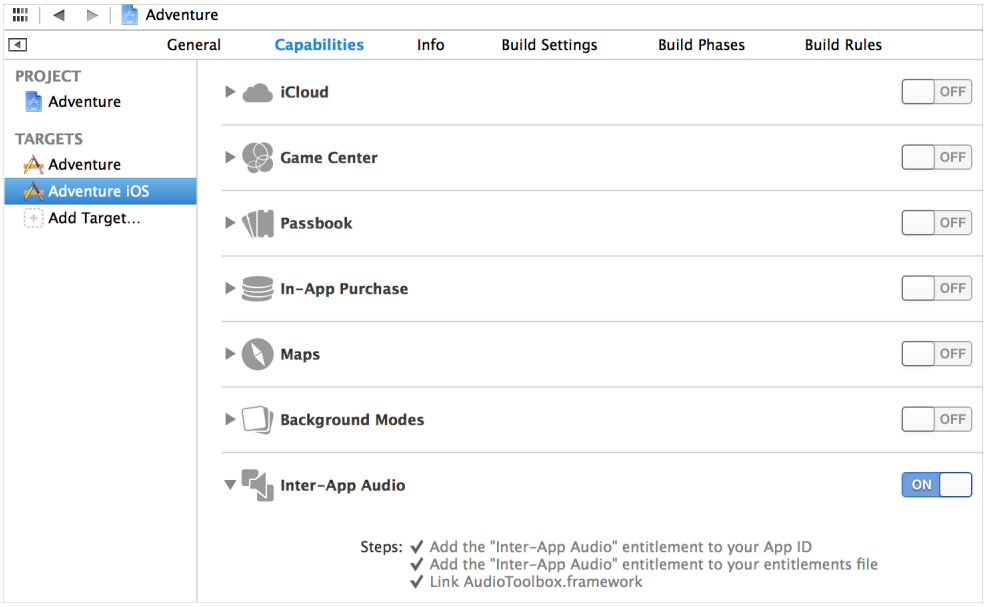
- The version number
- The build number
- An option to code sign the app for the Mac App Store, to code sign the app with a developer ID for distribution outside the Mac App Store, or to leave the code unsigned
- The deployment target, which is the earliest OS X version on which the app will run
- The icon that OS X uses to identify the app to the user

Specifying debug or release builds is done elsewhere. See Run Your App (page 91).

Add Technology Features to a Target

To add various Apple technologies—such as iCloud, Game Center, In-App Purchase, and Maps—to your app, select its target in the project editor and click Capabilities. Add a capability by setting a switch to On. Xcode adds the necessary entitlements file to your project and links the target to the necessary frameworks. In some cases, Xcode might encounter issues enabling a capability. If so, that information will be displayed in the information area for that capability.

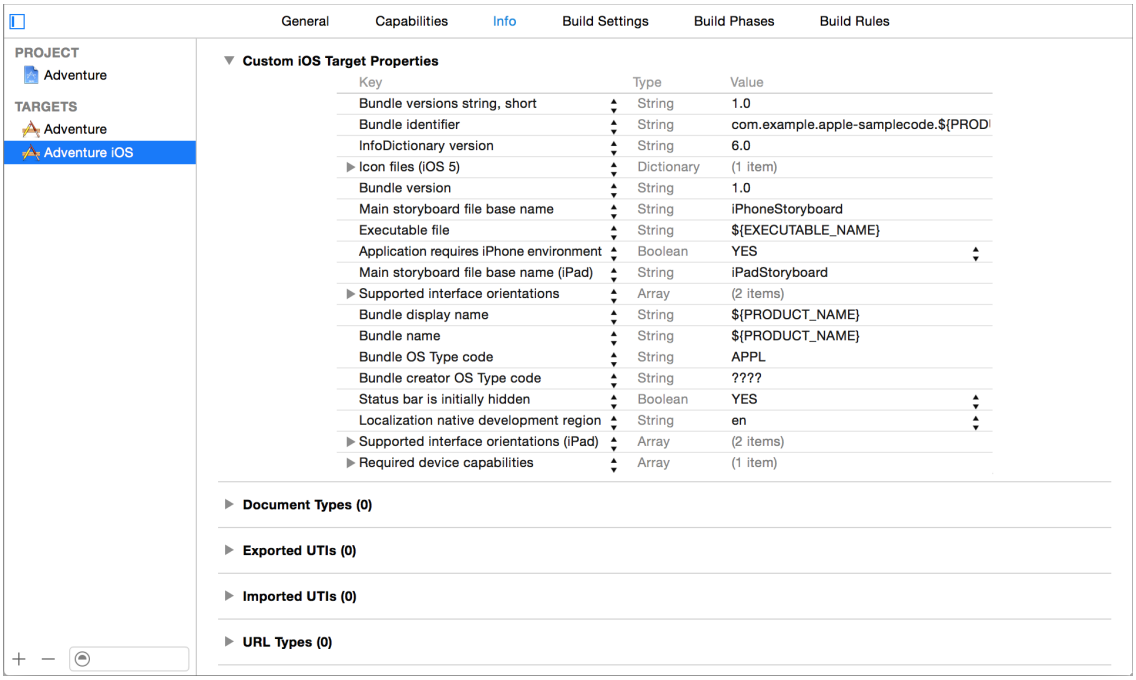
You can show or hide detail for a capability by clicking the disclosure triangle to the left of the capability name. For capabilities that are off, this area describes the capability and actions that occur when the capability is turned on. For capabilities that are on, use this area to view or update any associated configuration and to identify issues that need fixing.



For more information on adding capabilities, see Adding Capabilities.

Add File Type and Service Information to a Target

The Info pane for a target shows you properties associated with your app, file types that your app can create or open, and for OS X, services provided by your app. Most of the custom target properties are modified in other parts of the Xcode interface (such as the bundle identifier, version, and build number set in the General pane.) The screenshot shows the Info pane for the iOS target of the Adventure app.



The Document Types setting specifies the document types you can create and edit in your app, and provides a custom icon displayed for that document type by iOS or Mac OS.

Add exported and imported UTIs for any file types your app can export or import. Unlike document types, which are usually unique to your app, UTIs specify general formats like plain text or `.png`. For example, UTIs support copying and pasting to and from the clipboard between apps. See *Uniform Type Identifiers Reference* for more information and a list supported of types.

The URL Types setting lets you specify custom schemas for exchanging data with other apps by using custom protocols. For example, some existing schemas include `http`, `mailto`, and `sms`. For more information, see Using URL Schemes to Communicate with Apps (iOS) or *Launch Services Programming Guide* (Mac OS) for more information.

Mac OS apps use the Services item to add items that appear in the Services menu. For more information, see *Services Implementation Guide*.

Override Build Settings for a Target

A target contains instructions—in the form of build settings and build phases—for building a product. A target inherits the project's build settings. Although most developers seldom need to change these settings, you can override any of the project's build settings by specifying different settings at the target level. Select a target in the project editor to modify the target settings in the Info, Build Settings, or Build Phases pane.

Use Workspaces to Work on Related Projects

Workspaces are a collection of projects that help you reduce the complexity of larger applications. Workspaces have several benefits:

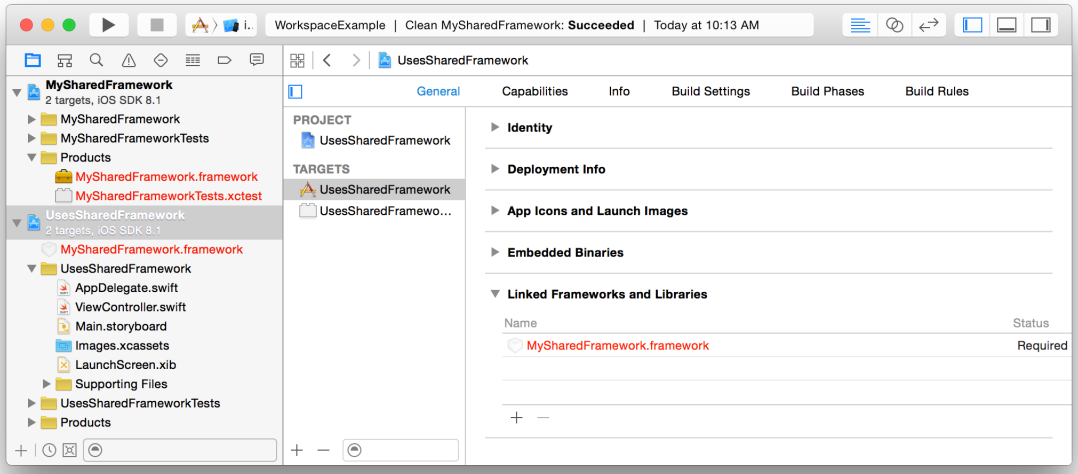
- Any project in the workspace has access to all the content from any other project in that same workspace, including compiled content.
- You can set up dependencies between projects so that a single build command builds all required pieces for the chosen target.
- You can include static libraries or modules, either your own or those of a third party.
- You can break up large projects into smaller pieces, allowing easier maintenance and sharing of functionality.

Create a workspace by choosing File > New > Workspace. After you create a workspace, you can create new projects within it and add existing projects to it. After you create the workspace, open the workspace file instead of the project file.

Convert an existing project into a workspace by choosing File > Save As Workspace. The existing window for the project is converted to a workspace window for the new workspace.

The screenshot shows an example of a workspace with two Xcode project files. The top project in the navigator area is a framework called `MySharedFramework`. The other project file is an app, `UsesSharedFramework`, that includes the shared framework. Using a workspace gives the app project access to everything in the shared

framework project and makes tasks like debugging much easier. Adding the framework to the list of linked frameworks for the app creates a dependency between the app and the framework. Xcode checks whether the framework needs to be built before the app.



For more information on workspaces, see Xcode Workspace, Creating a Workspace, and Adding an Existing Project to a Workspace.

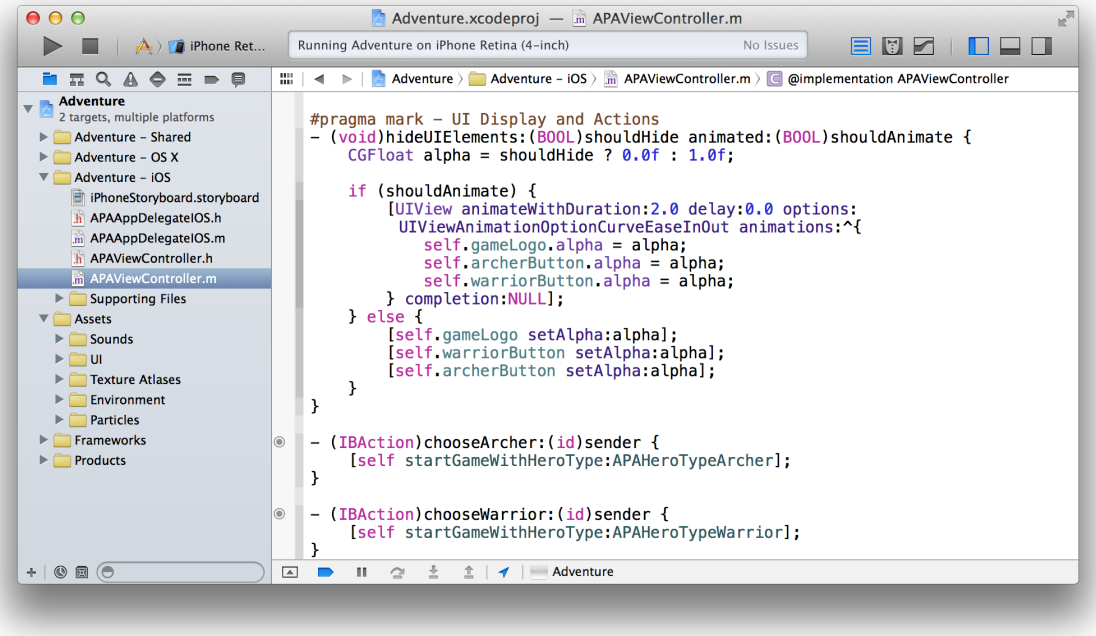
Close and Reopen a Project or a Workspace

To close a project or workspace, choose File > Close Project or File > Close Workspace. Xcode remembers which windows you had open and how they were configured, and it restores them when you reopen the project or workspace.

Write Code in the Source Editor

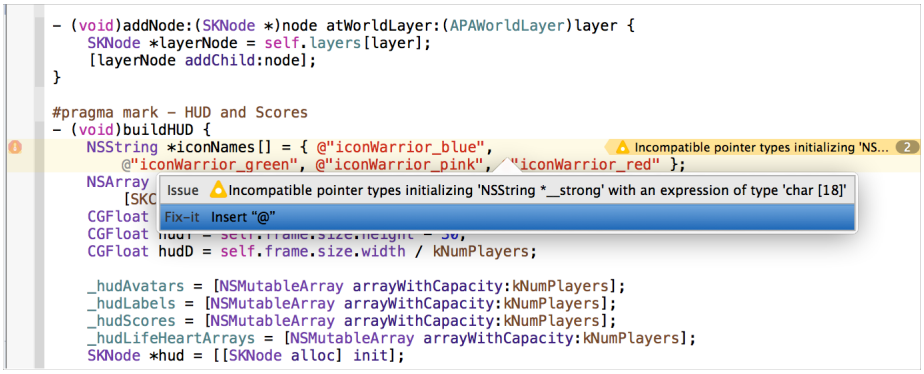
You spend most of your development time writing, editing, and debugging code. With features like syntax correction, code completion, and static code analysis, the Xcode source editor helps you enter code quickly and accurately. Customizable features like split windows, keyboard shortcuts, and syntax-aware fonts and text colors allow you to configure the source editor to suit your work style.

To view and edit a source file, select it in the project navigator. The file's contents appear in the editor area of the workspace window.



Fix Errors as You Type

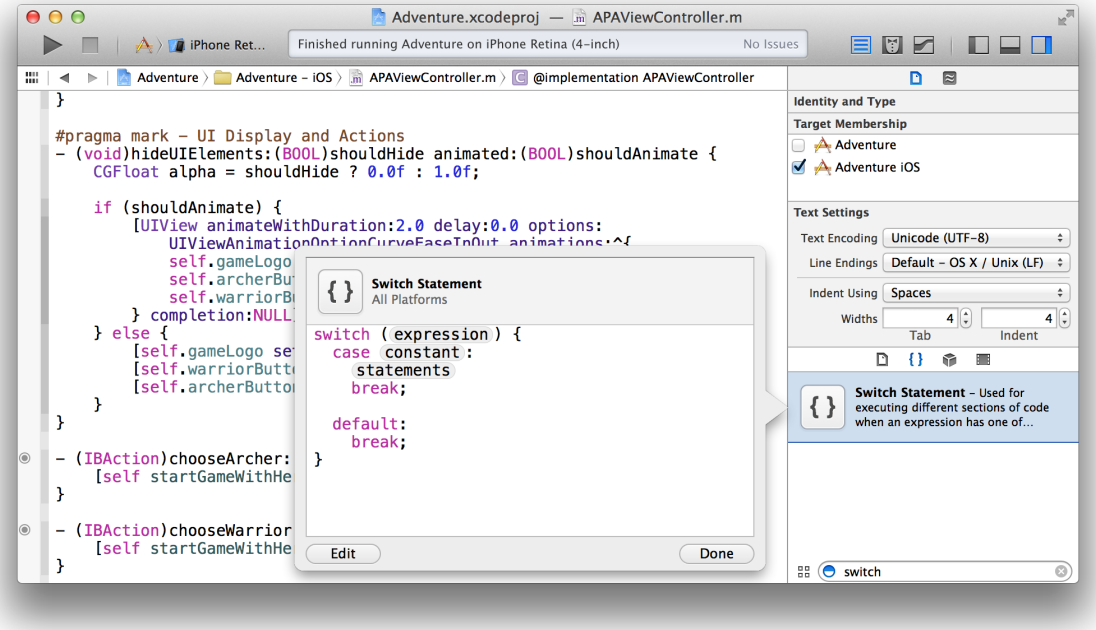
As you type into the source editor, Xcode scans your text. When you make a syntax error, Xcode marks it with a red underline or a caret. Click the error, and Xcode displays a message describing the issue.



Often, Fix-it offers to repair your error automatically. Select a suggested correction, and press Return to accept it. In the screenshot, Fix-it suggests inserting the “@” character before the text string. For more information, see Catching Mistakes with Fix-it.

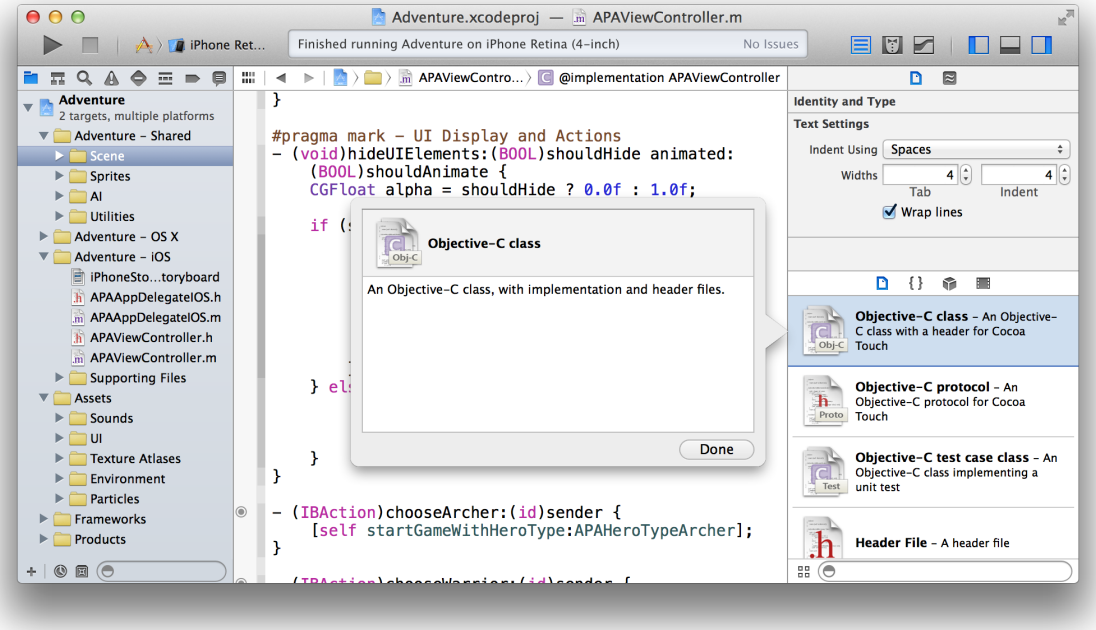
Drop Code Snippets into Your Files

Use code snippets to enter source text with minimum effort. You can drag a code snippet directly from the Code Snippet Library into a source file. To access the Code Snippet Library, click the Code Snippet button ({}) in the utilities area of the workspace window. The Code Snippet Library provides useful standard snippets, such as the switch statement snippet shown in the screenshot. To add your own code snippets to the library, create your own snippets, and add shortcuts, see *Source Editor Help* .

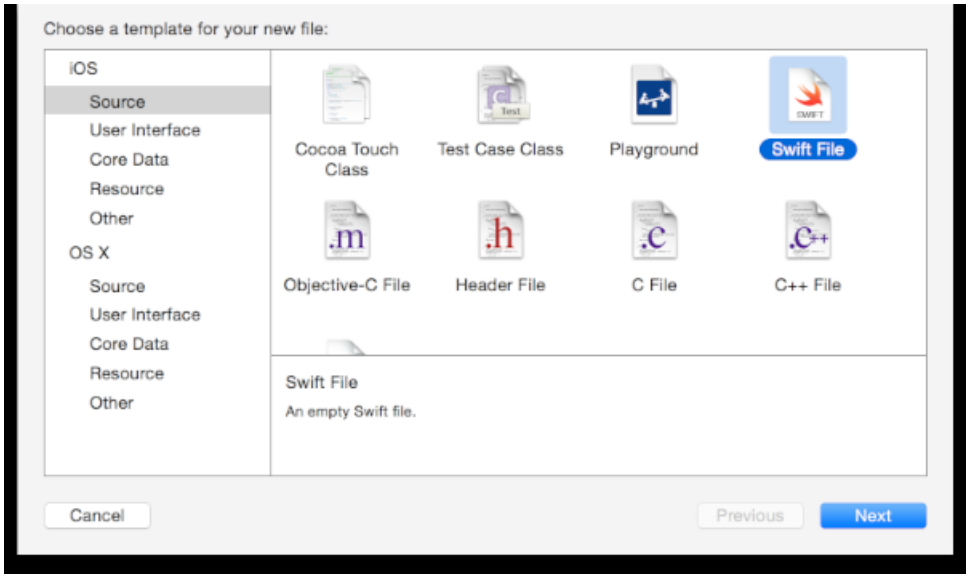


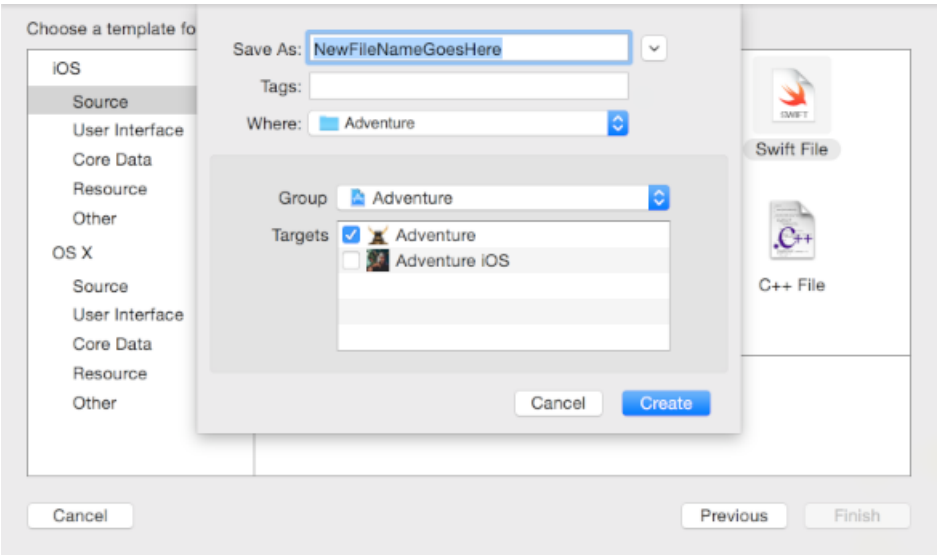
Create Source Files from Templates

Use file templates to add files to your project with minimum effort. To access the File Template Library, click the File Template button ( ) in the utilities area of the workspace window. Create a source file by dragging its template to the project navigator.



Alternatively, choose File > New File or press Command-N. Xcode brings up the New File dialog, where you can choose a template for your file. After choosing a template and pressing Next, you name the file and add it to your project.





Perform Static Code Analysis

Use the static analyzer to find bugs in your code before you even run your app. The static analyzer tries out thousands of possible code paths in a few seconds, reporting potential bugs that might have remained hidden or bugs that might be nearly impossible to replicate. This process also identifies areas in your code that don't follow recommended API usage, such as Foundation, UIKit, and AppKit idioms.

To perform static code analysis, choose Product > Analyze. The Xcode static analyzer parses the project source code and identifies these types of problems:

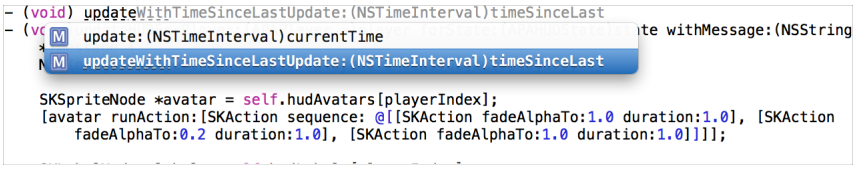
- Logic flaws, such as accessing uninitialized variables and dereferencing null pointers
- Memory management flaws, such as leaking allocated memory
- Dead store (unused variable) flaws
- API usage flaws that result from not following the policies required by the frameworks and libraries the project is using

The static analyzer reports problems in the issue navigator, available by clicking the Issue Navigator button ⚠ in the project navigator bar. Select an analyzer message in the issue navigator to display the associated code in the source editor. Click the corresponding message in the source editor. Use the pop-up menu in the analysis results bar above the source code editor to study the flow path of the flaw. Then edit the code to fix the flaw.

For more detail, see Performing Static Code Analysis in *Xcode Help*.

Speed Up Typing with Code Completion

When you begin typing the name of a symbol, Xcode offers inline suggestions for completing the name. Click an item in the suggestion list to select it, or use the Up Arrow and Down Arrow keys to change the selected suggestion. Press Return to accept the suggestion.

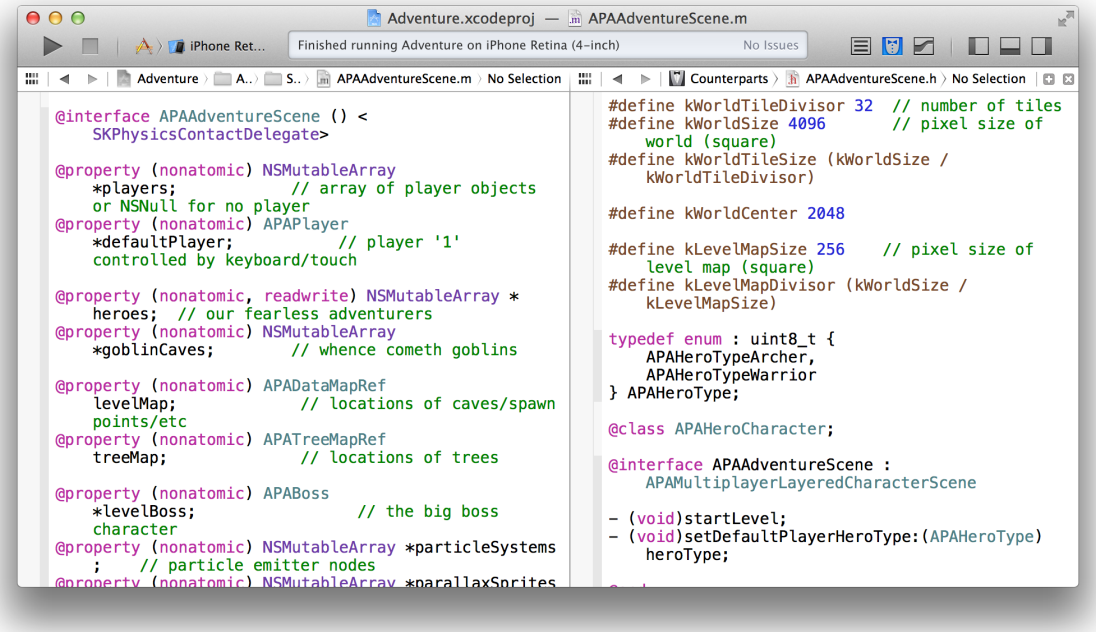


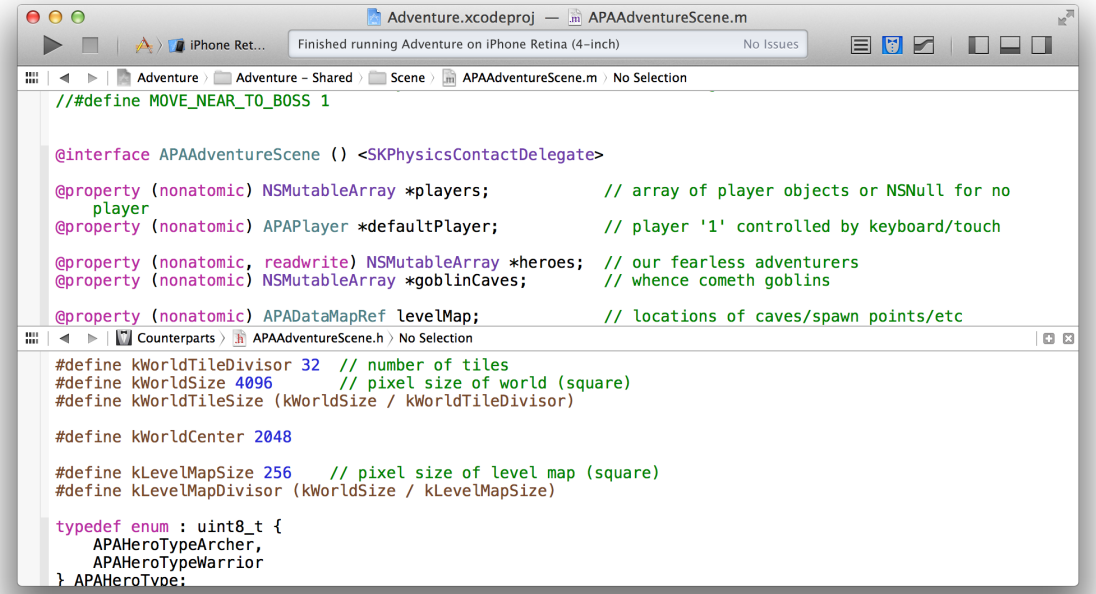
When a method or function contains parameters or arguments, code completion includes a placeholder for each. To move from one placeholder to another, choose Navigate > Jump to Next Placeholder (or Navigate > Jump to Previous Placeholder). Alternatively, Tab navigates to the next placeholder and Shift-Tab navigates to the previous one.

For more detail, see Entering Text with Code Completion.

Split the Editor to Display Related Content

Split the editor pane to see multiple views of the same file or to view multiple related files at once. For example, you can simultaneously view an implementation file and its header file counterpart. To split the source editor, open an assistant editor pane by clicking the Assistant Editor button ( ) in the workspace toolbar. The split can be vertical or horizontal.



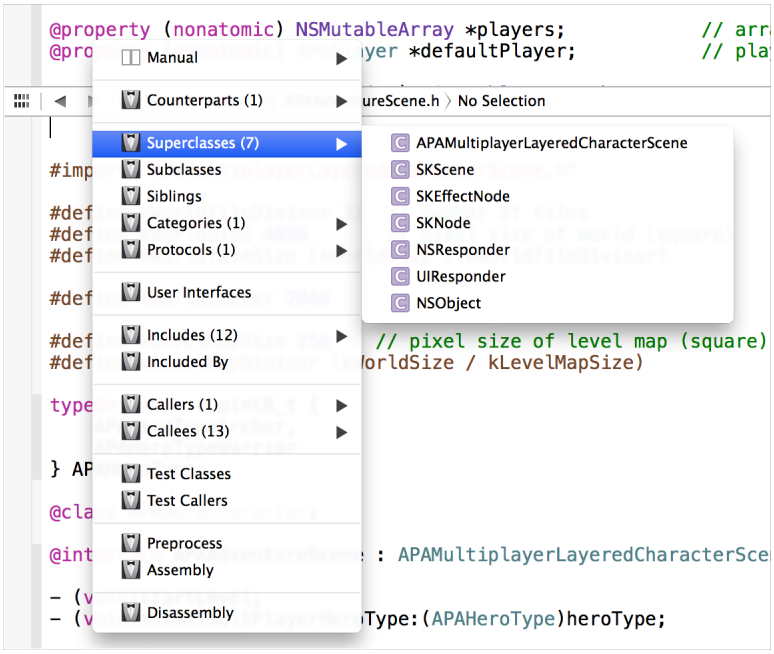


To change the orientation of the split, choose View > Assistant Editor, and then choose one of the menu options. In both of the screenshots above, the navigator and utilities areas are closed to maximize the viewing area of the source editor.

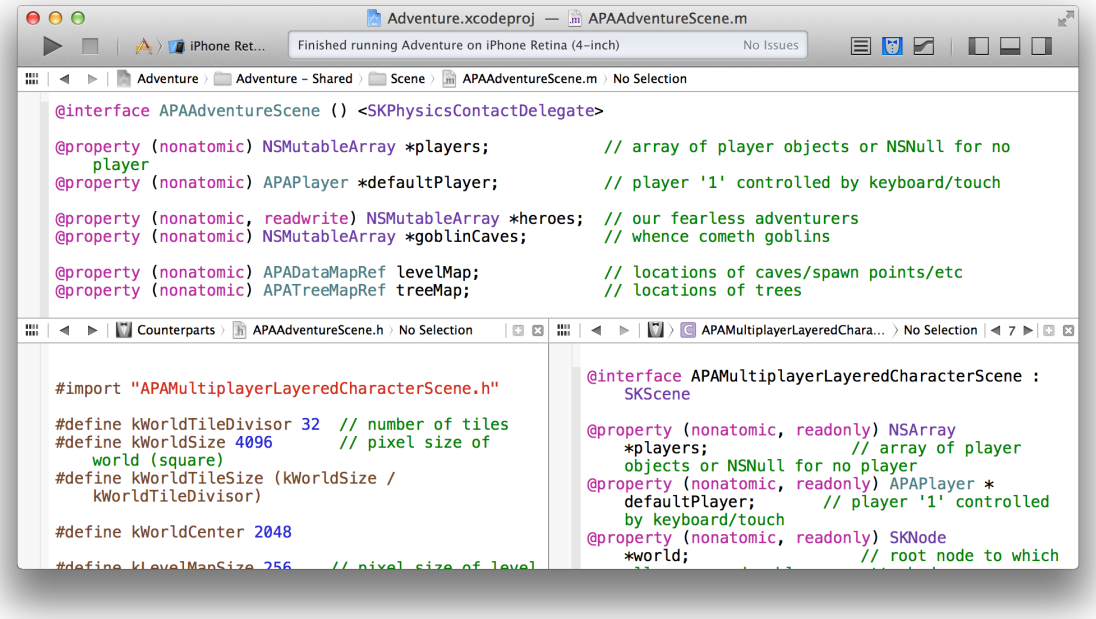
When you open an assistant editor pane, you can set it to either of two modes: manual or tracking. In manual mode, you select the file to display by navigating to it in the jump bar. The contents of the assistant editor do not change as you change the contents of the main editor.

In tracking mode, you select a criterion from a pop-up menu. Criteria include include groupings such as counterparts, superclasses, subclasses, and siblings. Once you choose a criterion, Xcode lists the appropriate files in a sub-menu. As you change the file in the main editor, Xcode updates the assistant editor based on the selected criterion.

To change the mode, select one from the Assistant pop-up menu. (The Assistant pop-up menu is the first item to the right of the back and forward arrows in the assistant editor jump bar.)

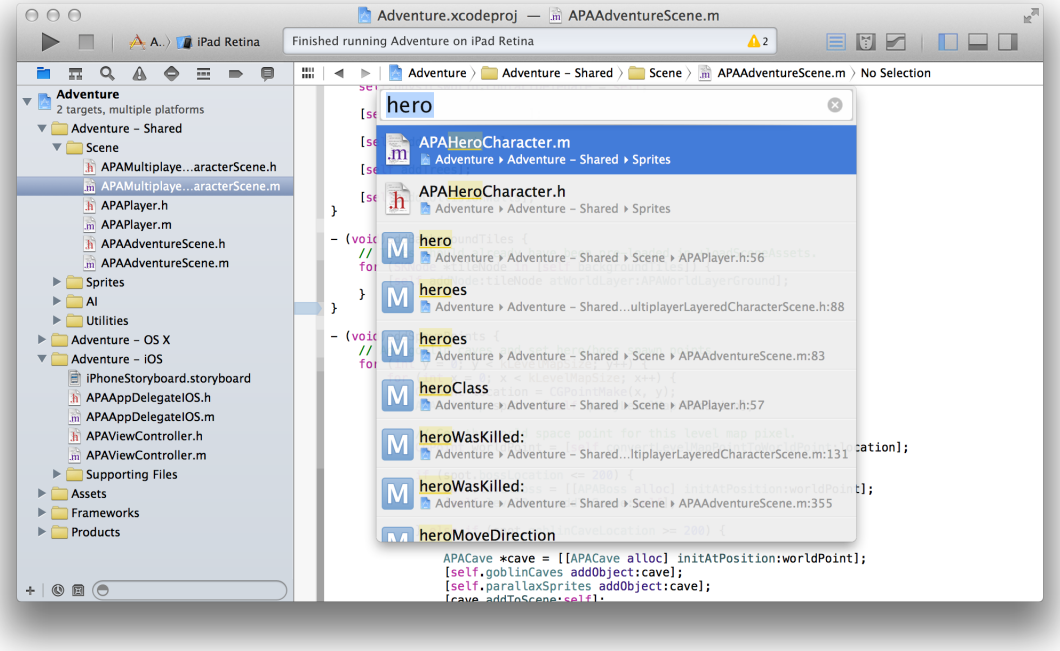


You can further split the assistant editor pane by clicking the Add button ( ) in the top-right corner of the assistant editor pane. The nearby close button ( ) closes it again.



Open a File Quickly

Choose File > Open Quickly to locate files that define a specified symbol or whose filenames contain a specified string. Open Quickly searches are case insensitive and are limited to the current project and to the active software development kit (SDK). From the search results list, double-click the file you want to open.



To open the file in the assistant editor pane, hold down the Option key when you double-click. To open the file in a separate window, press Option-Shift. To see a dialog letting you specify where the file should open, press Option-Shift-click.

Use Gestures and Keyboard Shortcuts

Gestures and keyboard shortcuts can simplify and enhance your use of the source editor. Besides the common Multi-Touch gestures in OS X, these gestures are particularly applicable within the source editor:

- A two-finger click opens a contextual menu for the editor (as does Control-click or Left-click with the mouse).
- A two-finger swipe up or down scrolls vertically, and left or right scrolls horizontally.
- A two-finger swipe left or right navigates through any files opened in an editor. Swiping left shows the previous file, and swiping right shows the next file.

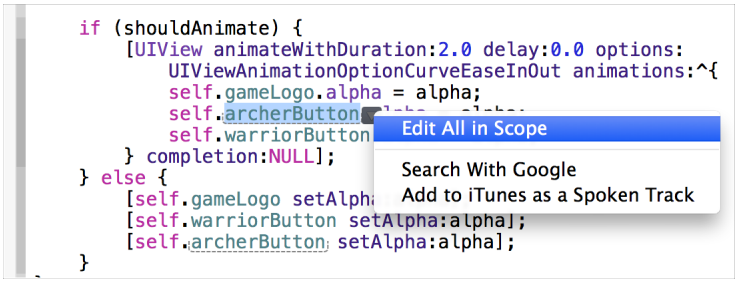
Keyboard sequences serve as shortcuts for many common menu commands in Xcode. For example, Shift-Command-O invokes the Open Quickly command from the File menu, and Shift-Command-J invokes the Jump to Definition command from the Navigate menu. Other keyboard shortcuts assist with editing operations. For example, Control-K deletes every character from the insertion point to the end of the line.

Keyboard shortcuts are established through key bindings, which you can view and modify by choosing Xcode > Preferences and selecting Key Bindings.

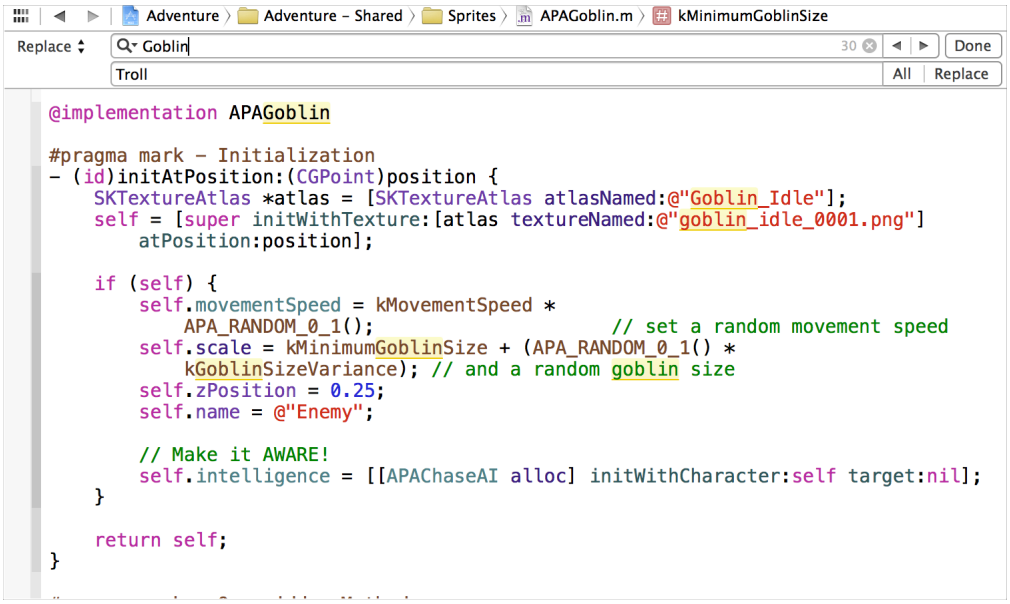
Automate Extensive Changes in Your Code

Xcode offers several approaches to making changes that apply to multiple lines of text.

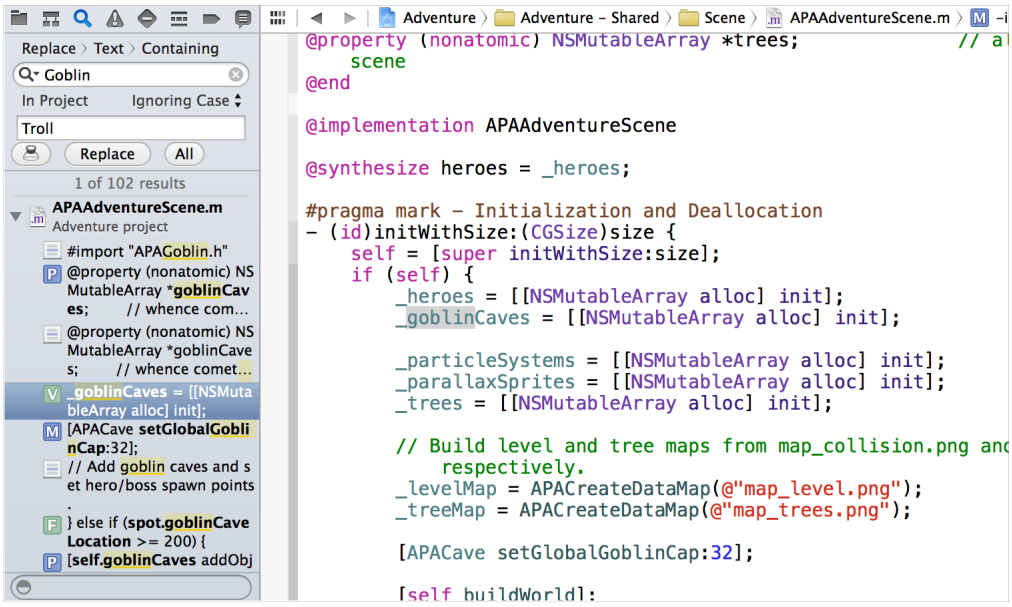
You can simultaneously modify all the occurrences of a symbol, such as the name of a local variable or parameter, within a scope. Place the insertion point in the symbol you want to edit. When the disclosure triangle appears, click it to display the menu, and choose Edit All in Scope. Edit the symbol. As you type new text, all instances of the symbol change simultaneously.



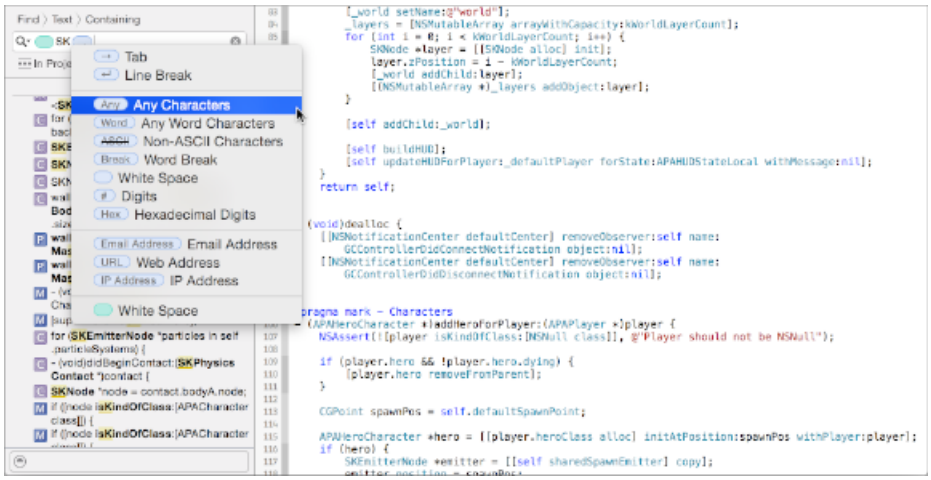
Change instances of a text string in a single file by choosing Find > Find and Replace.



Change instances of a text string in your project or workspace by choosing Find > Find and Replace in Project. This command displays the find navigator. You can customize the operation—for example, to limit the scope of the search or to match the case of letters in the string. The find navigator provides a preview that allows you replace all instances of the string or to accept or reject individual replacements.

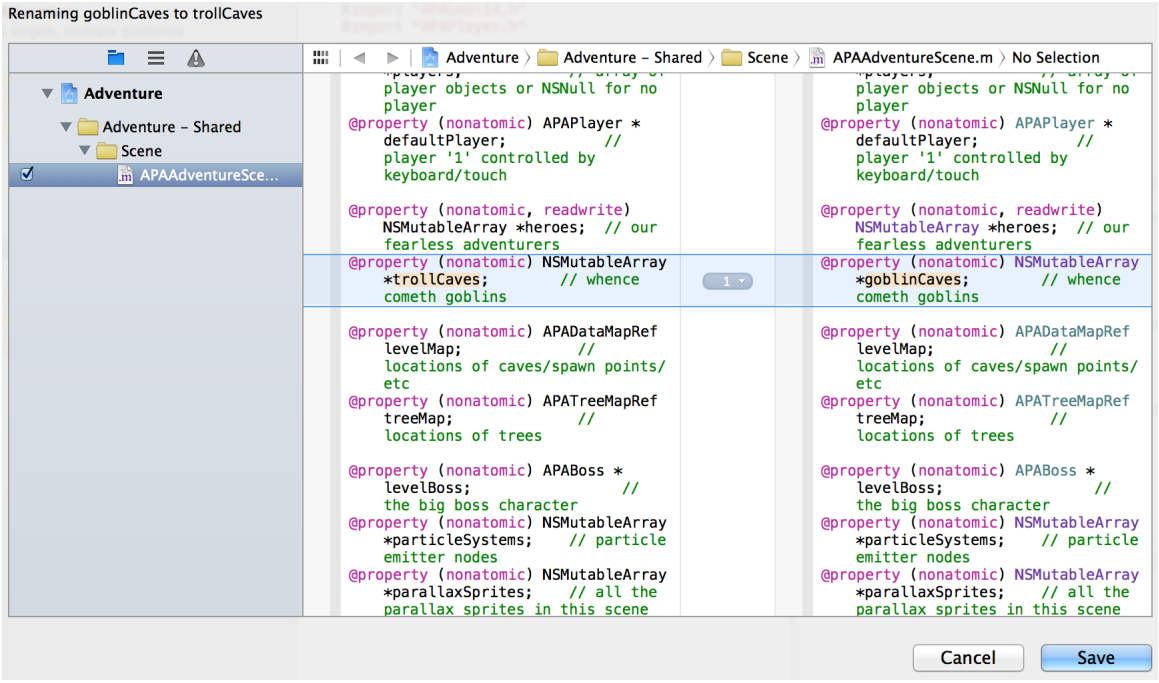


You can use wildcard string patterns in the search field. To enter a component of a pattern, click the disclosure triangle on the left of the find string field and choose Insert Pattern. Choose a component from the pop-up menu of patterns. Xcode inserts the wildcard at the current location of the cursor in the find string.



You can refactor your code to improve its structure, readability, and maintainability without changing its behavior. A refactoring operation (also called a *transformation*) is applied to a code fragment or a symbol that you select in the source editor. You can rename symbols, extract code into methods, create superclasses, move items up to the superclasses or down to their subclasses, and encapsulate variables throughout your project files.

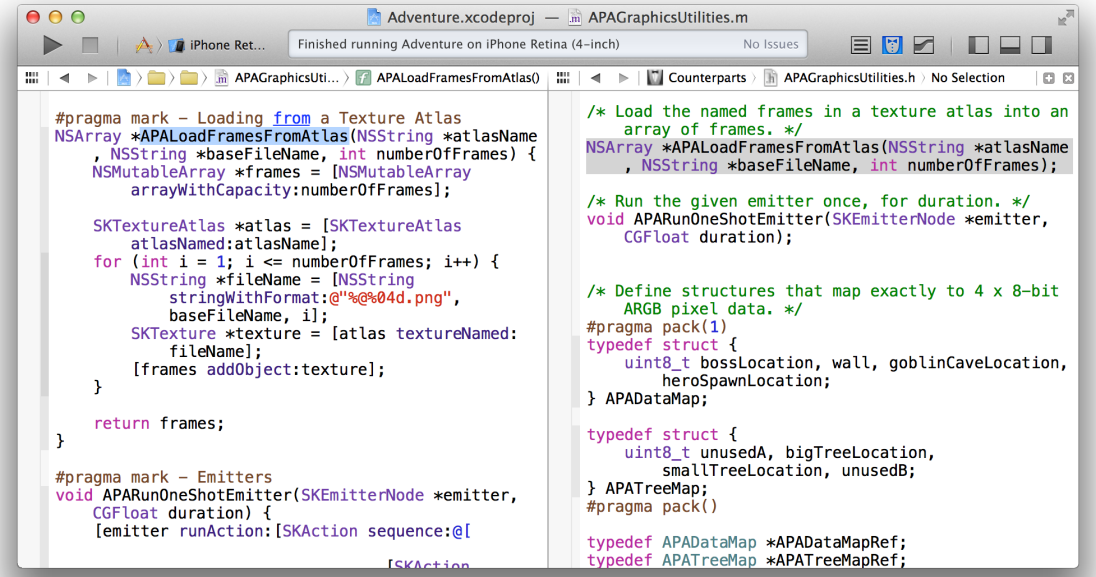
After selecting the code fragment or symbol you want to refactor, choose Edit > Refactor and then choose the appropriate refactoring command. A preview pane shows you how each change will appear when applied. Deselect a file in the leftmost pane of the preview dialog to leave it out of the refactoring operation. You can edit your source code directly in the preview. Any such edits are shown in the preview and are included in the refactoring operation.



Display the Definition of a Symbol

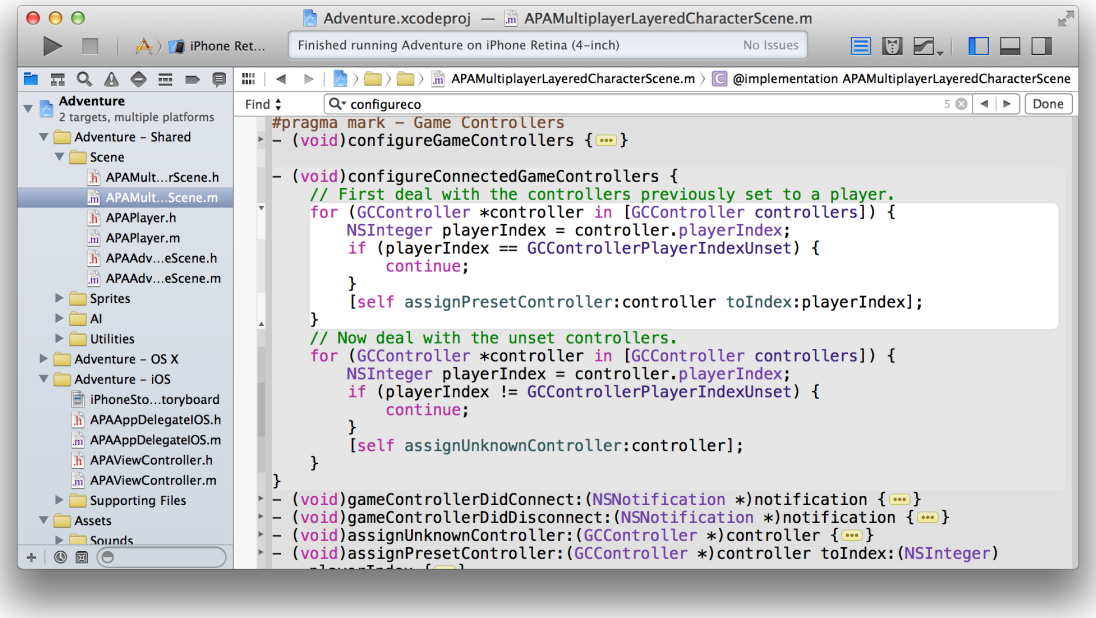
Place the pointer over a symbol and Command-click to display the symbol definition. The source editor navigates to the symbol definition and highlights it. If the definition is in a separate file, the source editor displays that file. (Alternatively, place the pointer over a symbol and choose Navigate > Jump to Definition.)

Place the pointer over a symbol, and Option-Command-click to display its definition in the assistant editor pane, as illustrated for the `APALoadFramesFromAtlas` function in the screenshot. This approach lets you keep the symbol in view as you inspect its definition.



Examine the Structure of Your Code with Code Folding

You can more easily focus your attention on a particular method or function in source code by hiding the other parts of the source code. Choose Editor > Code Folding > Fold Methods & Functions. Navigate to the method you want to unfold and double-click the Ellipsis button to unfold the method. The screenshot shows the `configureConnectedGameControllers` method unfolded.



Move the pointer into the focus ribbon on the left edge of the editor to display a scope—such as the `for` statement in the screenshot—in a focus box. Additional scopes are indicated by degrees of shading in the code.

For more detail, see Folding and Unfolding Source Code.

Match Pairs of Braces, Parentheses, and Brackets Automatically

Xcode helps you balance delimiters automatically. For example:

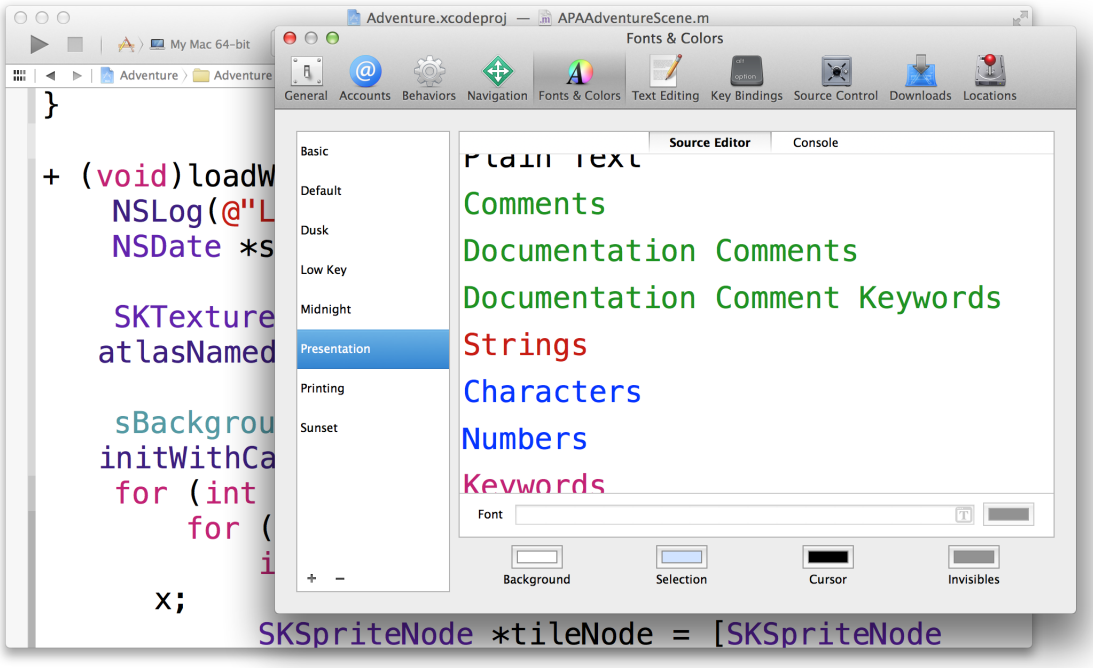
- Position the pointer over the focus ribbon on the left edge of the source editor. Xcode highlights the scope at that location, as shown in the previous screenshot.
- Type an opening brace. Xcode automatically inserts a closing brace after you enter a line break.
- Type a closing brace or other delimiter. Xcode briefly highlights its counterpart.
- Use the Right Arrow key to move the insertion point past a closing delimiter. Xcode briefly highlights its counterpart.

- Choose Editor > Structure > Balance Delimiter. Xcode selects the text surrounding the insertion point, including the nearest set of enclosing delimiters.
- Double-click any delimiter. Xcode selects the text enclosed by the delimiter and its counterpart.

For more detail, see Matching Pairs of Braces, Parentheses, and Brackets.

Choose Syntax-Aware Fonts and Text Colors

Xcode parses code based on the language, and it assigns a syntactic label to each token or string—for example, each comment, keyword, and class name defined in the project. Xcode assigns a color and font to each syntactic type to make it easier for you to read the code. You can select from several font and color themes by choosing Xcode > Preferences and then selecting Fonts & Colors. For example, the Presentation theme increases the font sizes so that the text is easier to read when projected on a screen. You can also create your own custom font and color themes.



Customize Editing and Indenting Options

You can change source editing and indenting settings to suit your preferences. Choose Xcode > Preferences, and select Text Editing to modify options such as these:

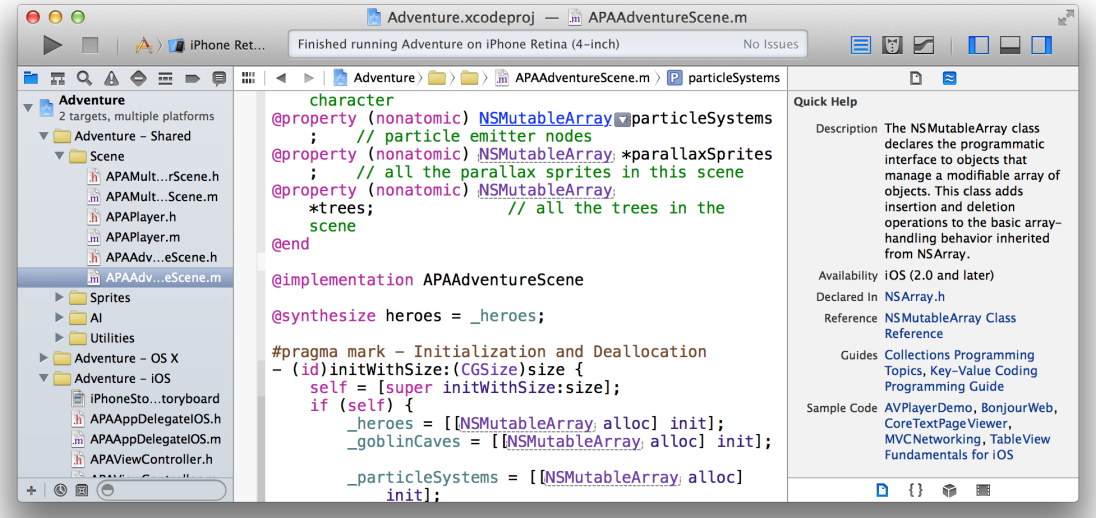
- Display line numbers in the source editor gutter.

- Automatically insert closing braces as you type.
- Suggest code completions while you enter code.
- Use spaces or tabs for an indent.
- Soft-wrap lines.
- Perform syntax-aware indenting.

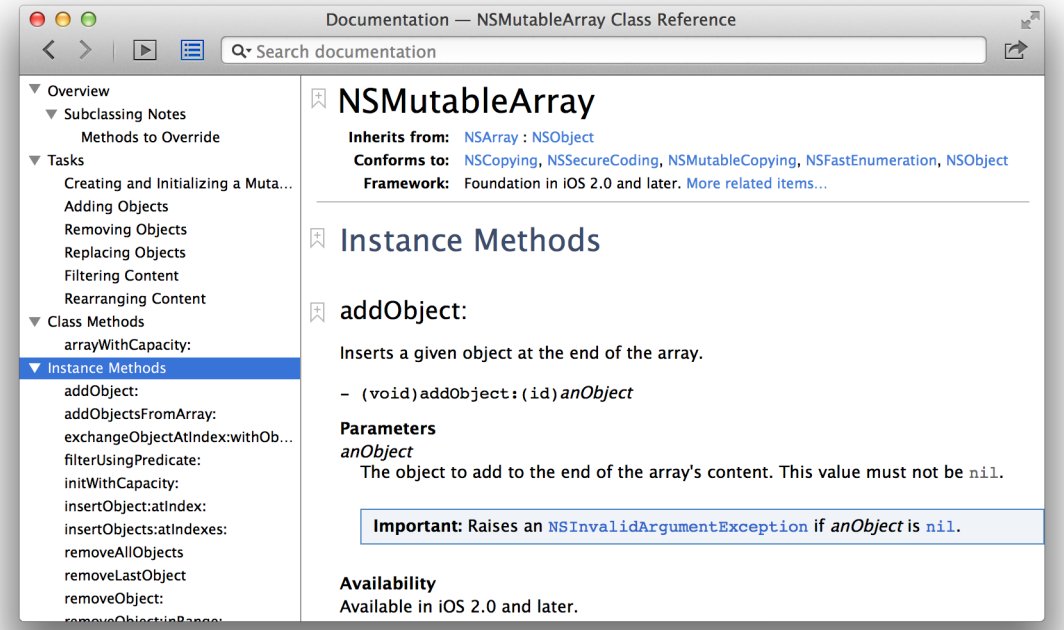
Look Up Documentation for a Symbol

Find concise reference documentation for a symbol, such as a method or property, by placing the insertion point in the symbol. Click the Quick Help button ( ) in the inspector pane toolbar. If the inspector pane is not open, in the main toolbar click the button to show the navigator in the workspace configuration button set. Quick Help for that symbol appears in the utilities area.

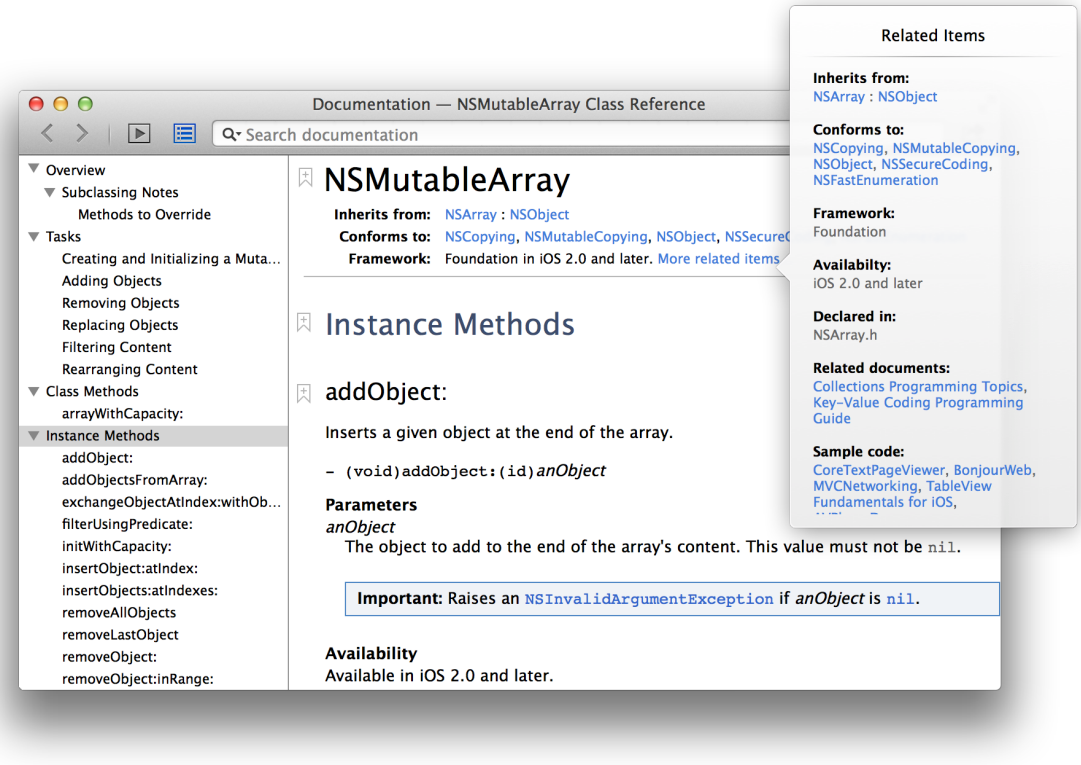
The information includes links to complete reference documentation for the symbol, the header file where the symbol is declared, related programming guides, and related sample code. (To view summary information in a pop-up window—the declaration, a description of the symbol, any return value, its release availability, header file, and a link to its related reference document—Option-click the symbol.)



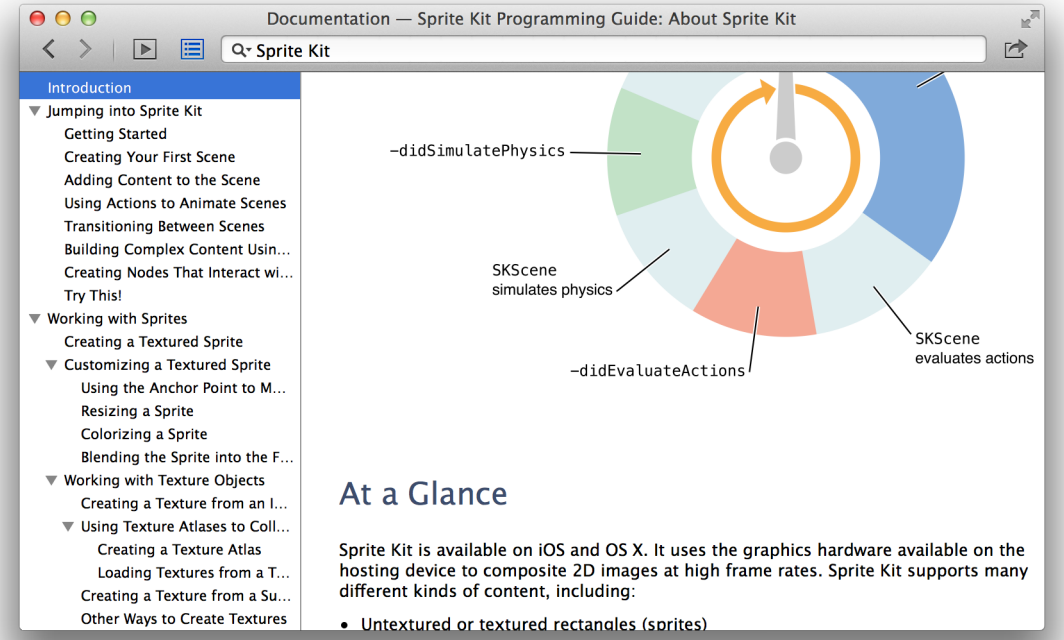
Click a link in Quick Help, and Xcode opens a separate Xcode document viewer window. The Xcode document viewer provides access to information without taking your focus away from the file you're editing.



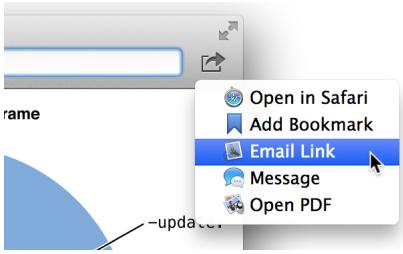
The document viewer delivers in-depth programming guides, tutorials, sample code, and video presentations by Apple engineers, in addition to detailed framework API references. From a class reference, click "More related items" near the top of the viewer for links to additional documents relevant to your programming task.



Use the search field in the toolbar to locate additional information about the API or programming concept.

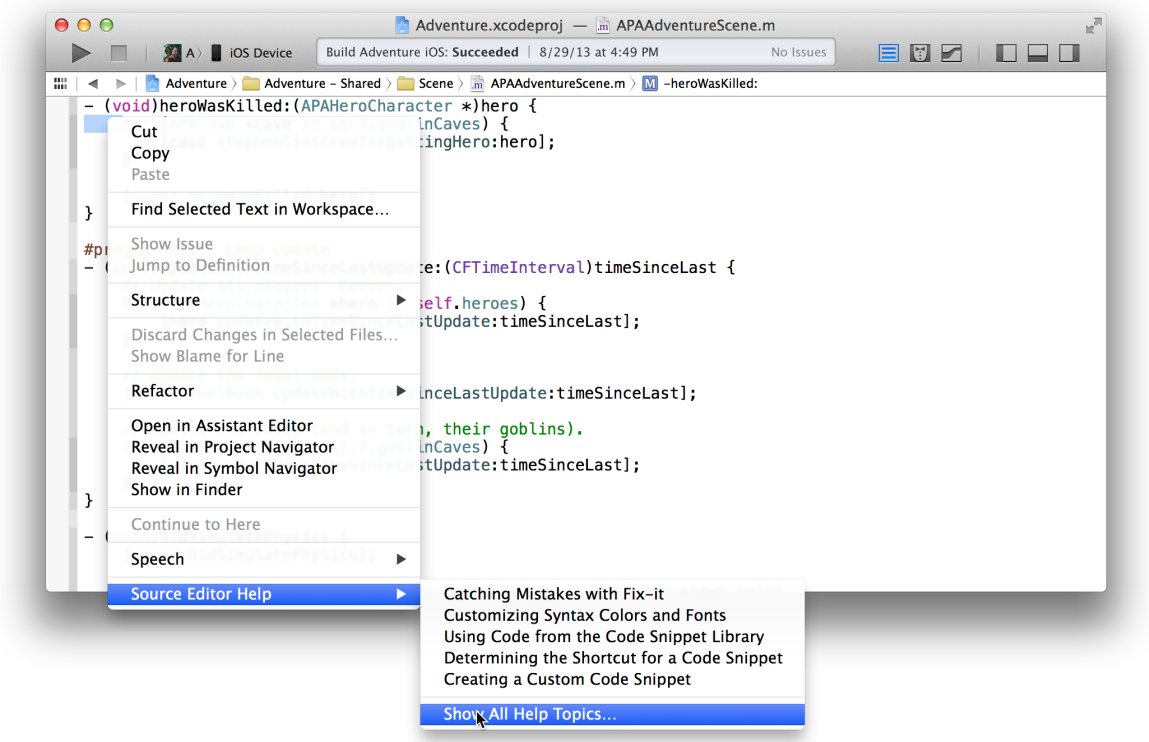


To include a link to the document in a message, click the Share button ( ) and choose Email Link or Message. You can open the document in Safari in HTML or PDF format from this menu. For a sample code project, click Open Project at the top of the window to download the project and open it in Xcode.

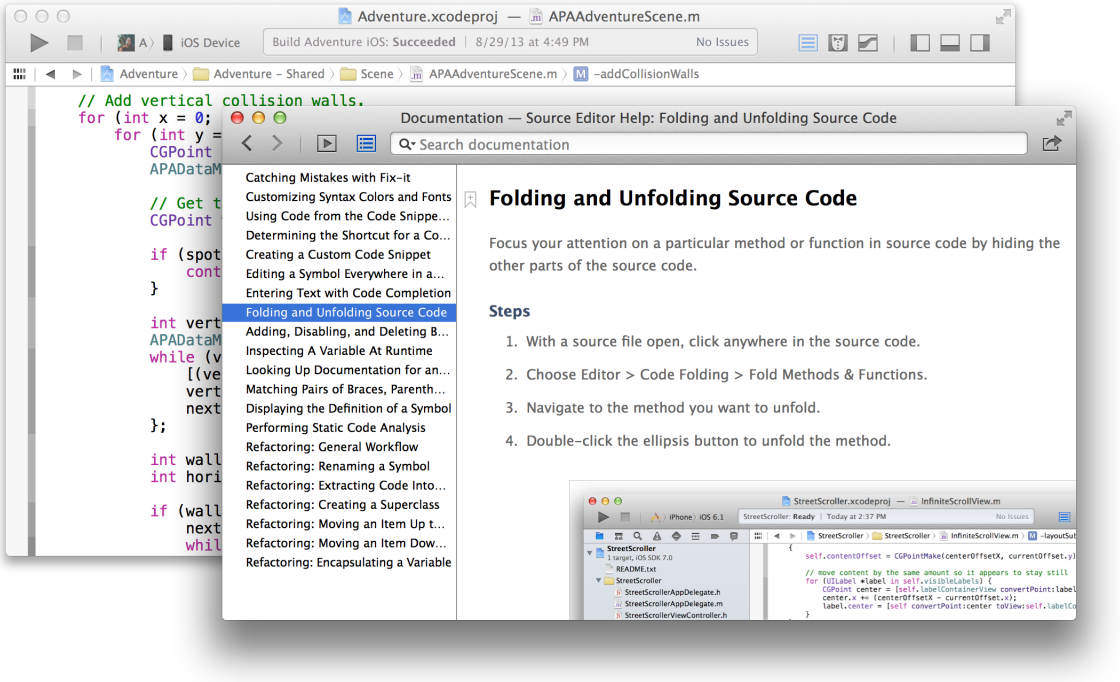


Find Help for Using the Source Editor

Step-by-step instructions for performing common source editor tasks are available directly in Xcode. Control-click anywhere in the source editor to see a short list of the most common operations. Choose Show All Help Topics to see all help articles for the source editor. Select a task, and a help article appears in the Xcode documentation viewer window.

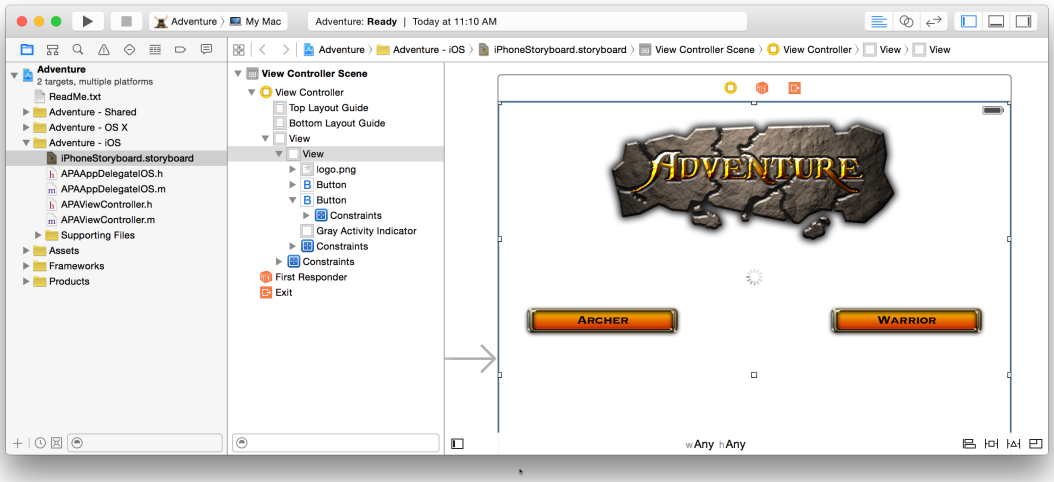


Xcode Help articles are available from shortcut menus throughout Xcode. Control-click in any of the main user interface areas to see a list of help articles available for that area.



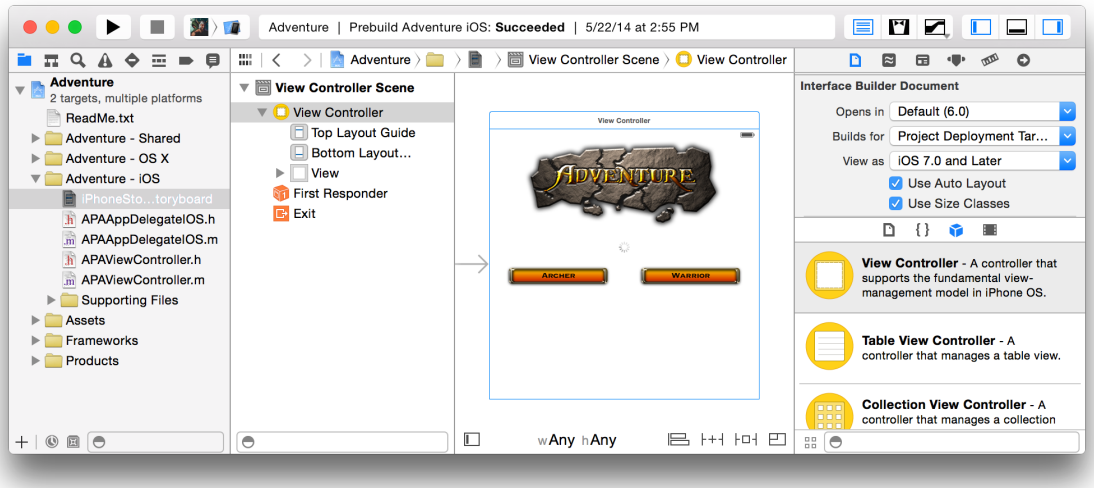
Build a User Interface

You create your app's user interface in Interface Builder. Select a user interface file in the project navigator, and the file's contents open in Interface Builder in the editor area of the workspace window. A user interface file has the filename extension `.storyboard` or `.xib`. An xib file usually specifies one view controller or menu bar. A storyboard specifies a set of view controllers and segues between those controllers. Unlike an xib, a storyboard can contain all of the visual components of your user interface. Default user interface files are supplied by Xcode when you create new projects from its built-in templates.



The contents of `.xib` and `.storyboard` files are stored by Xcode in XML format. At build time, Xcode compiles your `.xib` and `.storyboard` files into binary files known as *nibs*. At runtime, nibs are loaded and instantiated to create new views.

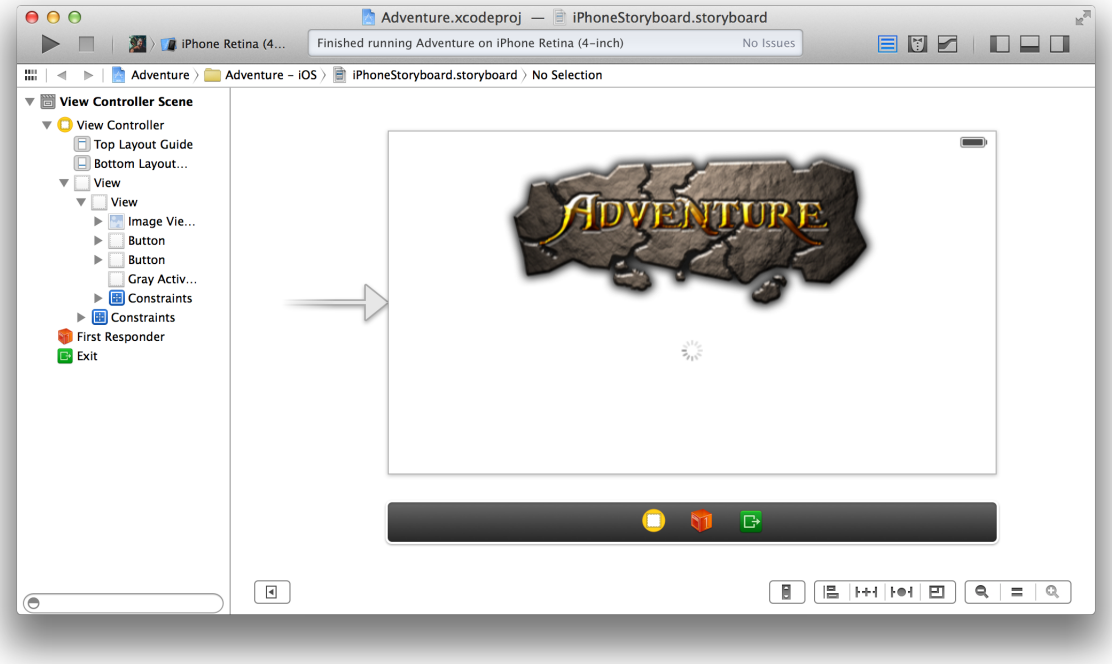
To add user interface elements, drag objects from the utilities area onto the Interface Builder canvas, where you arrange the elements, set their attributes, and establish connections between them and the code in your source files. As you lay out your app's user interface elements in Interface Builder, you can write the code that implements their behavior in the assistant editor.



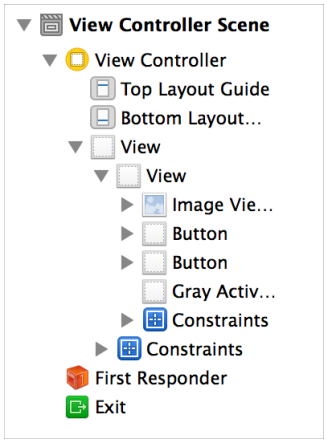
For more detail on the process of building a user interface, and creating and configuring interface builder files, see *Interface Builder Help* .

Add User Interface Elements from the Object Library

Interface Builder has two major areas: the dock (on the left) and the canvas (on the right). The dock lists the objects contained in the user interface file. The canvas is where you lay out these objects in your app's user interface.



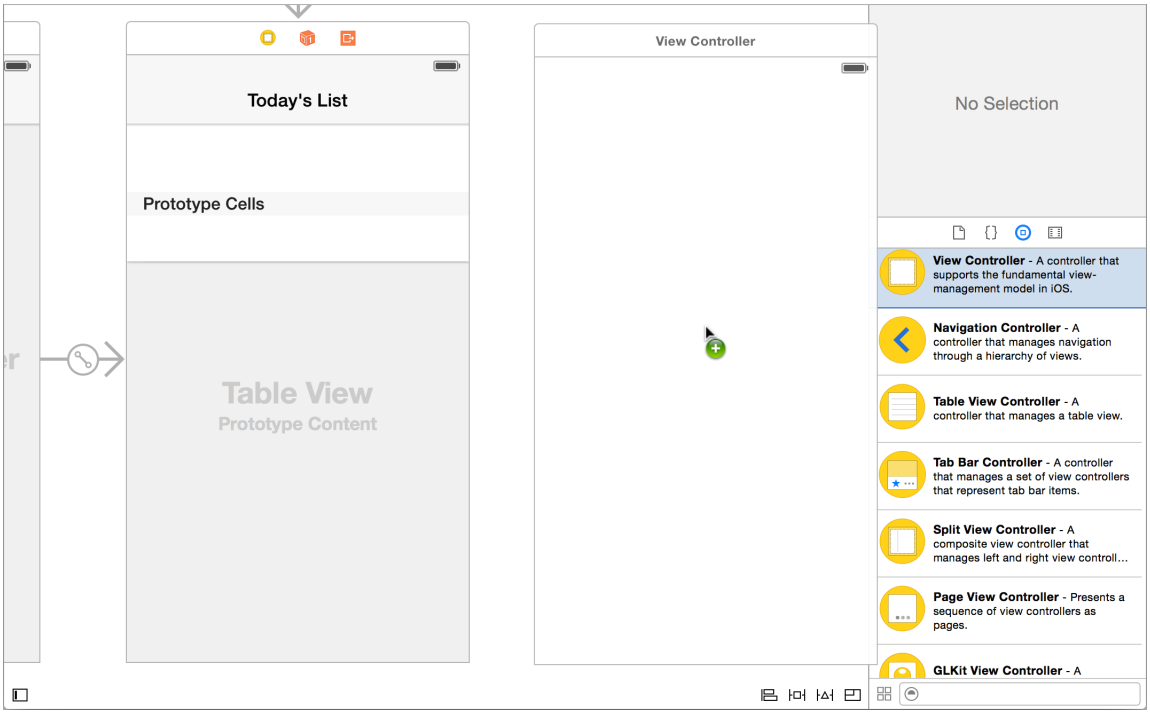
The *outline view* in the dock shows all the objects nested inside higher-level objects.



For xib files, you can display the high-level objects in an *icon view* instead of the outline view by clicking the Hide and Show Document Outline control on the lower left of the interface builder canvas ( ).

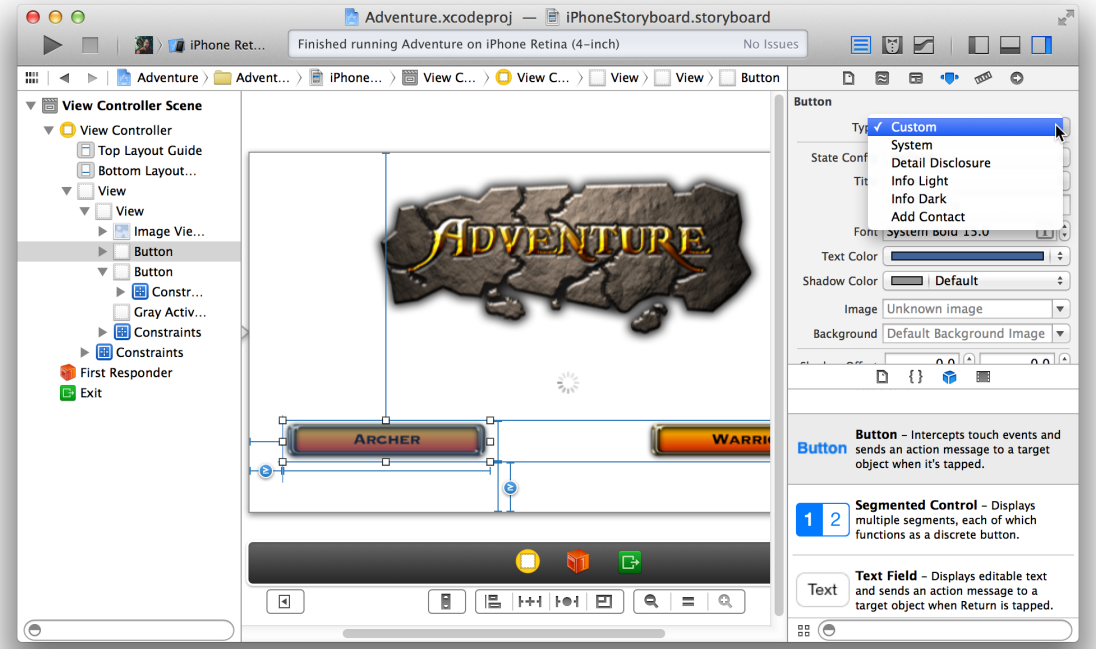
In storyboard files, the top level items in the outline view correspond to top level view controllers, or scenes, on the canvas. Storyboard files do not show an icon view when the outline view is hidden. Each scene on the storyboard has a high-level object view on the canvas as shown below. Starting from the left, the items in the icon view correspond to the scene, the first responder in the scene, and the exit segue for that scene. For more information on scenes, see Design the User Interface of Your App with Storyboards (page 73).

To add an object to your app's user interface, open the utilities area for the workspace window by clicking (one of the workspace configuration buttons in the toolbar.) Select the Object Library from the library pane by clicking the Object button in the library bar. Click the icon representing the object, and then drag it from the library, to either the outline view in the dock or onto the canvas. The screenshot shows dragging a View Controller onto the canvas.



As you add objects to Interface Builder, you resize them by their handles and reposition them by dragging. As you move items, dashed blue lines help you align and position the item within the view.

Above the library bar in the utilities area are the Interface Builder inspectors. You use these inspectors to specify some of the interface objects' appearance and behavior. In the screenshot below, the Attributes Inspector button( ) is used to specify the button type Custom.



For more help with adding objects and other items, see *Interface Builder Object and Media Help* .

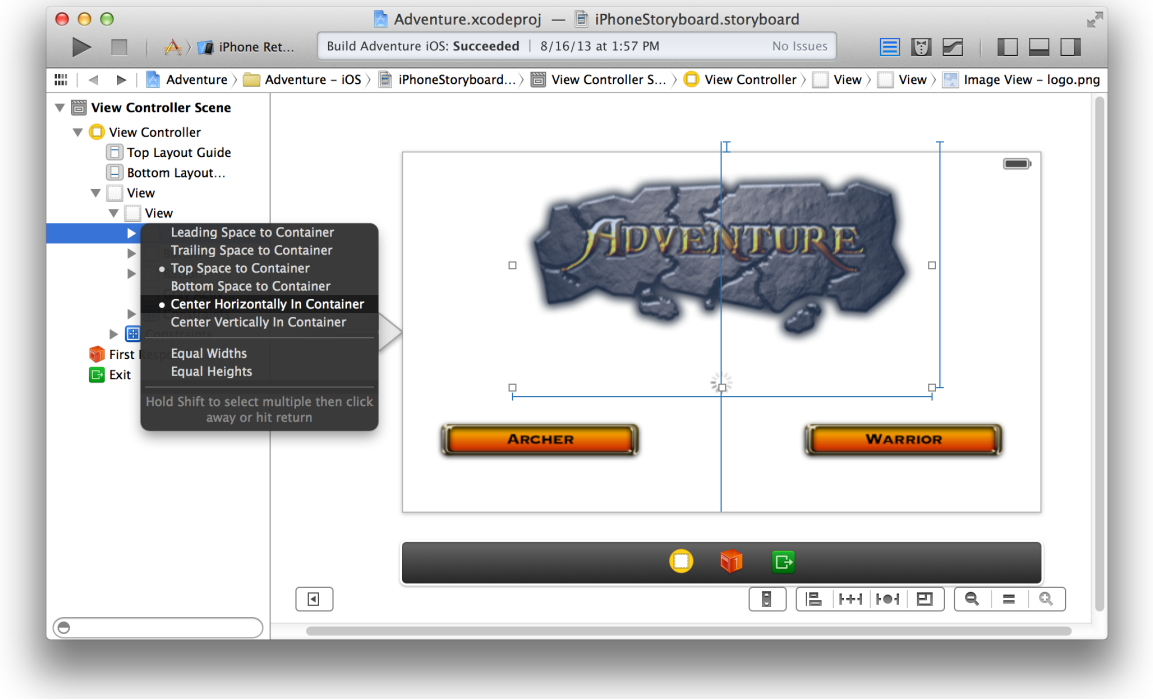
Lay Out User Interface Objects for Automatic Resizing and Positioning

Auto Layout enables your app to adapt to different window sizes, screen sizes, and device orientations. You do this by defining relationship *constraints* between the elements of your user interface. For example, center an image horizontally in a storyboard scene. As the user rotates the iOS device, the image remains horizontally centered in both landscape and portrait orientations of the device.

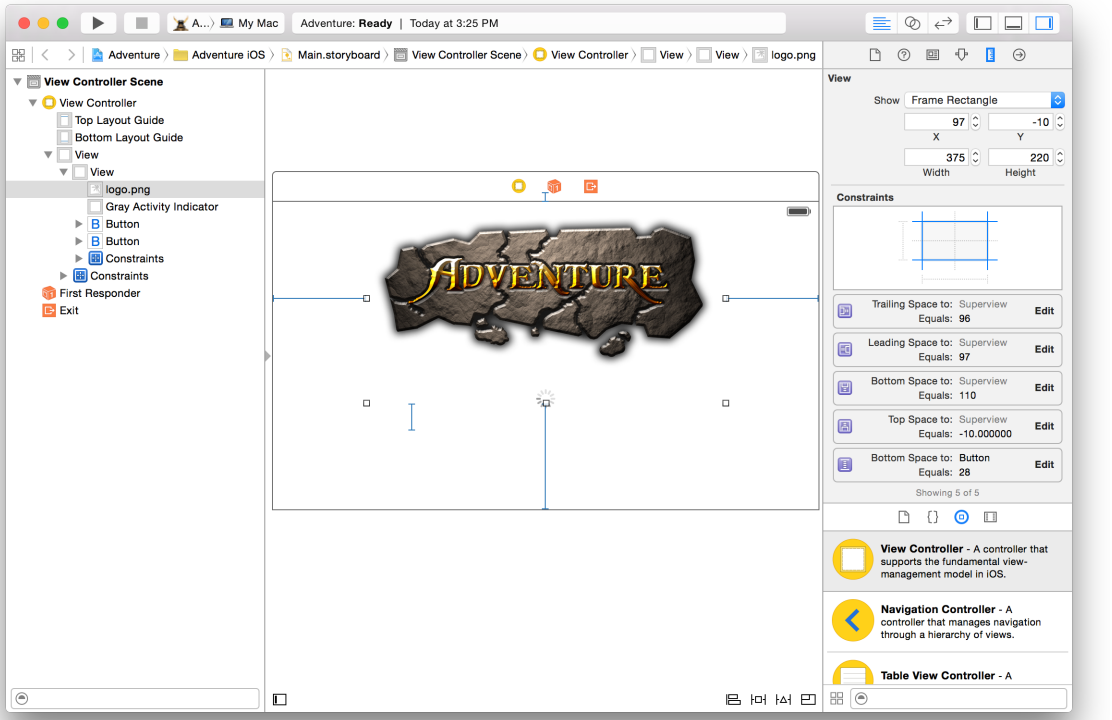
With Auto Layout constraints governing the layout, the objects in your user interface automatically resize and reposition themselves whenever:

- The user changes the screen orientation of an iOS device
- The user resizes a window in a Mac app
- Content dimensions change (for example, when the length of a text string changes in a label or button)

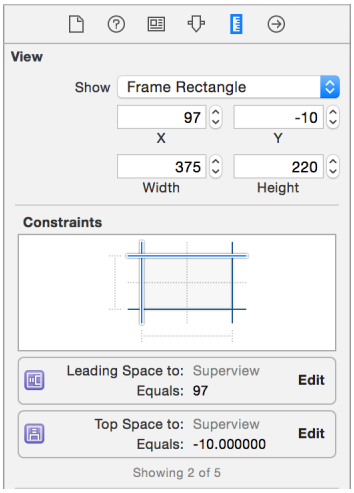
When you Control-drag between two user interface objects, Interface Builder presents you with a list of appropriate constraints. Add constraints to your objects by choosing from this list. The screenshot shows the result of Control-dragging from the main logo image view to the main view of the controller. The pop-up menu shows two constraints between the the image view and its container. "Top Space to Container" is a constraint between the top of the image view and the top of the container. The other horizontally centers the image view in its container.



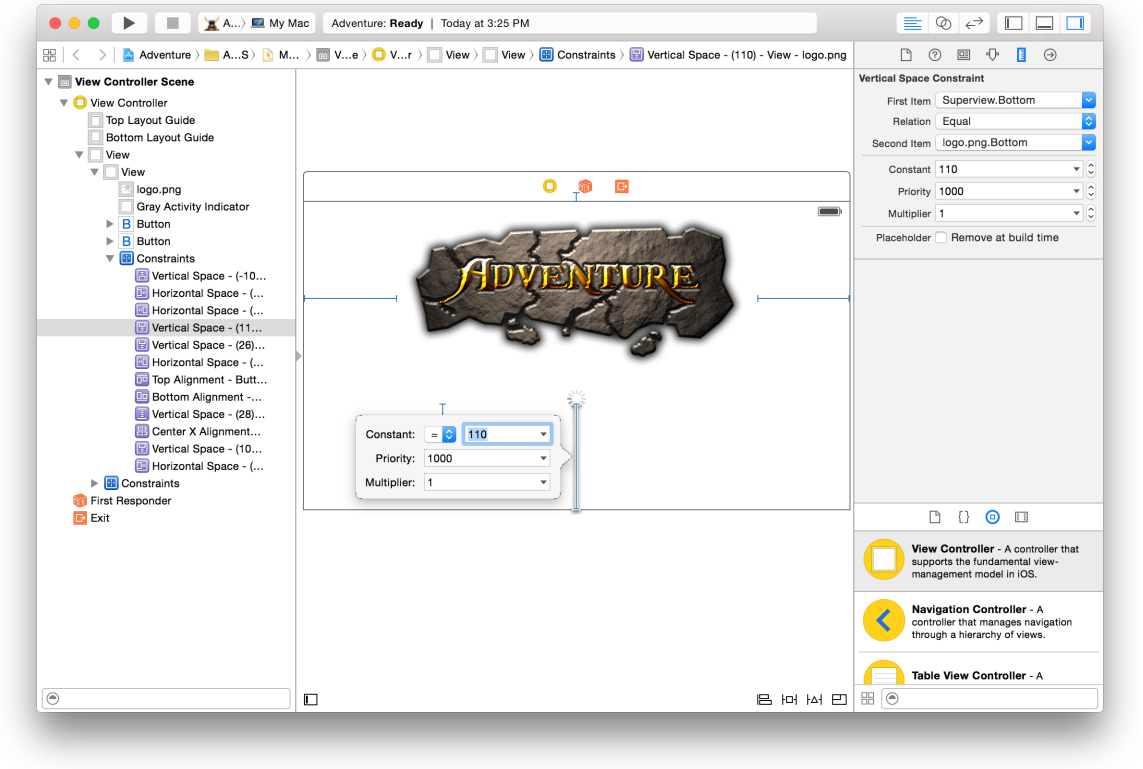
To see the constraints for an object, select the object from the outline view in the Interface Builder dock or select the object on the canvas. Constraints are represented by solid blue lines. You can view a list of constraints by selecting the Size inspector ( ).



The top part of the constraints section shows a graphical representation of the types of constraints for the selected object. Each blue horizontal or vertical line indicates that there are one or more constraints of that type. Selecting one or more lines filters the list to constraints that match the selected types.



View and edit a constraint in the inspector by selecting the constraint on the canvas or double-clicking the constraint in the Size inspector. You can also double-click a constraint to show a pop-up editor with the most frequently edited attributes.



There are more ways to add and edit constraints, including using the buttons along the bottom right of the canvas ( ). For more information on using Auto Layout and constraints see *Auto Layout Help*.

Connect User Interface Objects to Code

You write the code that implements the behavior of your user interface objects. Your code communicates with user interface objects through action and outlet connections.

Create an *action connection* when you need to send a message from a control to your code. A *control* is a user interface object that causes instant actions or visible results when the user manipulates the object. When the user clicks a button, for example, the button sends an action message telling your code to execute an appropriate action. Text fields, sliders, and switches are examples of commonly used controls in iOS; checkboxes, scroll bars, and text fields are commonly used controls in Mac OS.

Create an *outlet connection* when you need to send a message from your code to a user interface object. The object can be a control or any other object defined in your storyboard or xib file, such as a label, progress indicator, or map view. When your code determines that a button should disappear, for example, your code sends a message through an outlet telling the button to hide itself.

The connections are instantiated after the view controller is instantiated but before it has appeared. For more information, see Resource Management in View Controllers, Target-Action, and Target-Action.

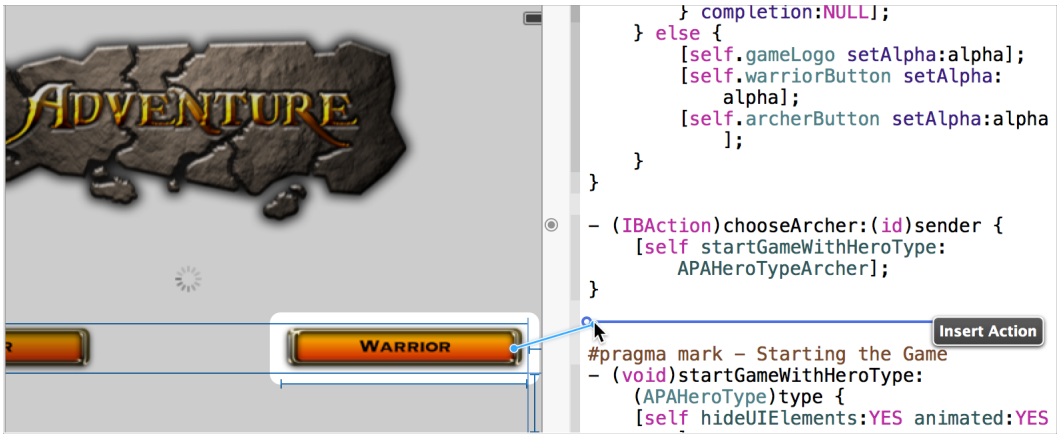
> **Note:** As of iOS 8.0, connections are no longer guaranteed to be instantiated after completing the call to `awakeFromNib`.

Send Action Messages from a Control to Your Code

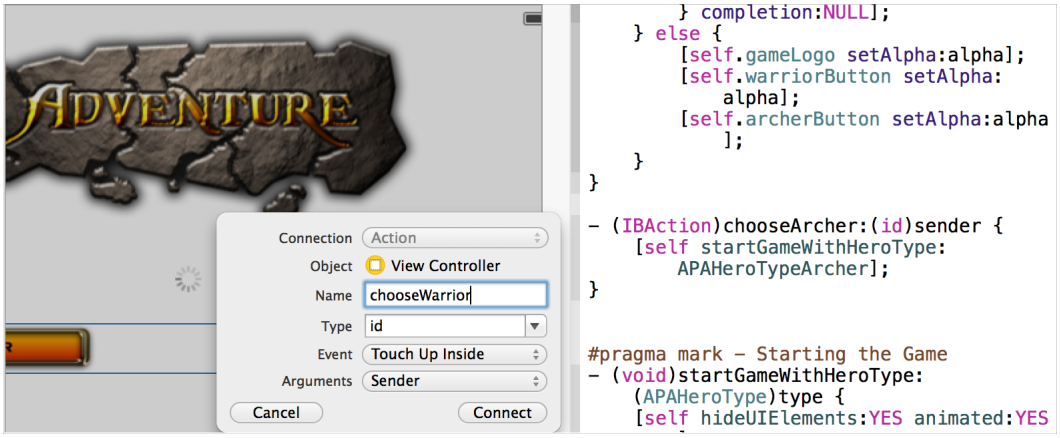
Whenever the user activates a control, such as by tapping it, the control should send a message instructing your code to perform some action. The easiest way to configure a control to send an action message to your code is by Control-dragging from the control in Interface Builder to your object's implementation file.

With Interface Builder open in the standard editor pane, select the control you want to configure, and click the Assistant Editor button ( ) in the workspace toolbar. The assistant editor opens your object's implementation file.

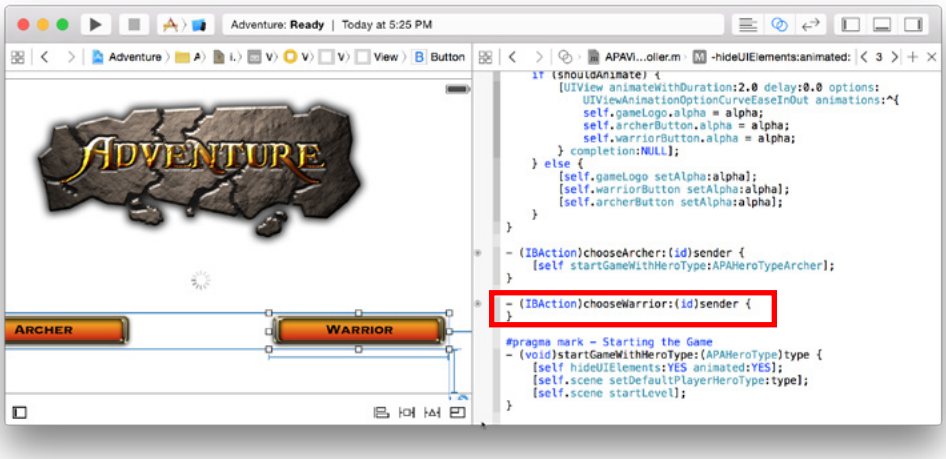
Control-drag from the control in Interface Builder to the implementation file. (In the screenshot, the assistant editor displays the implementation file of the view controller for the Warrior button.) Xcode indicates where you can insert an action method in your code.



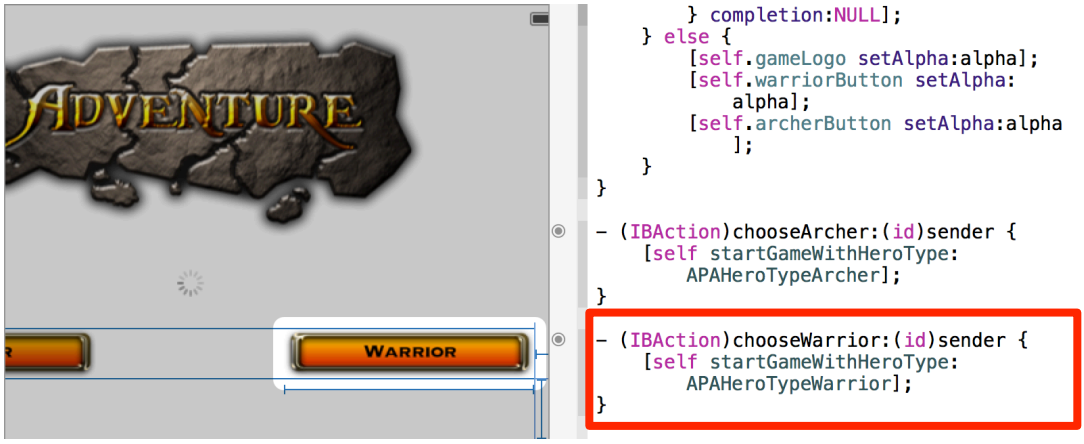
Release the Control-drag. The assistant editor displays a Connection menu. In this menu, type the name of the action method (`chooseWarrior` in the screenshot below), and click Connect.



In the implementation file, Xcode inserts a skeletal definition for the new method, as shown below. The `IBAction` return type is a special keyword indicating that this instance method can be connected to your storyboard or xib file. Xcode also configures the control to call the method when the selected event occurs. As a result, the method gets invoked whenever the control receives an action message.

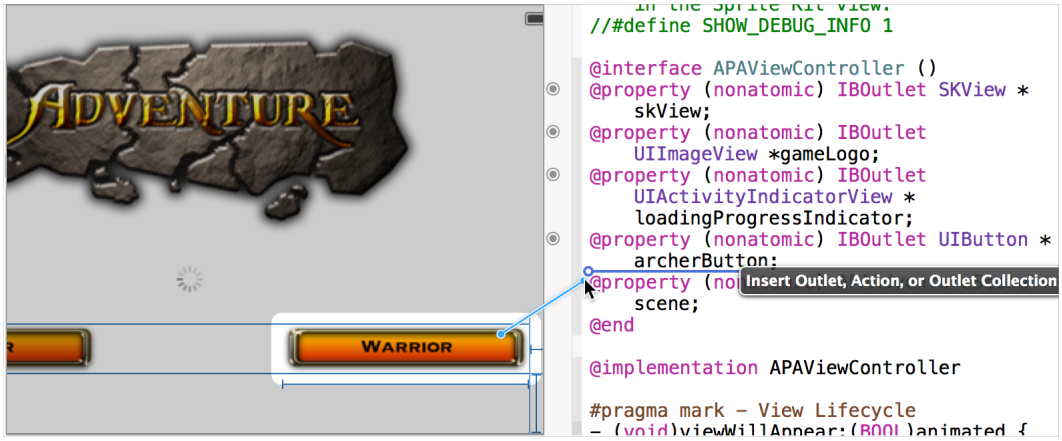


To this skeletal definition, add the source code that implements the action method. Whereas activation of the control is implemented by the system, you implement the control's behavior. In the screenshot below, an action method starts a new game when the user clicks the Warrior button.

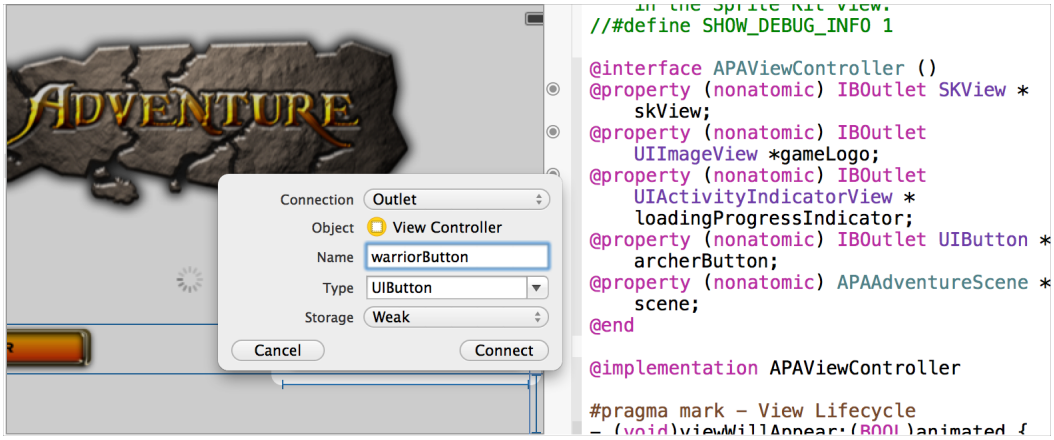


Send Messages to a User Interface Object Through an Outlet

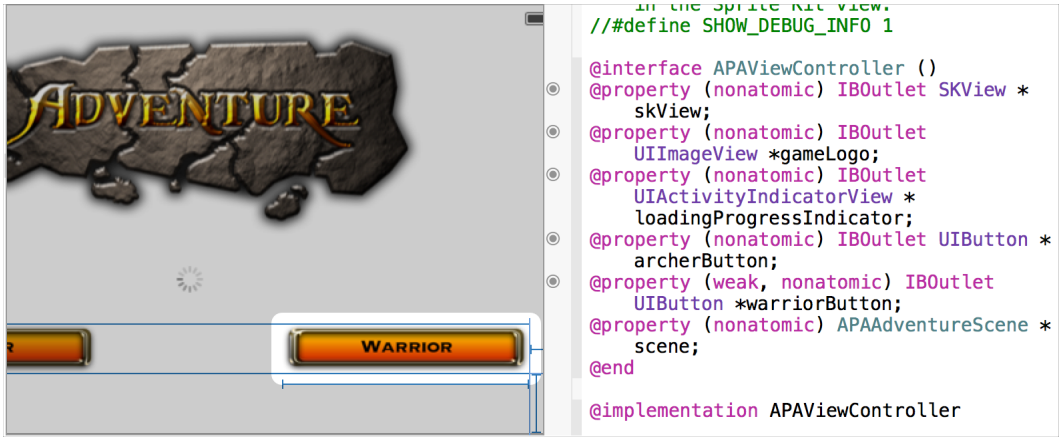
To enable your code to send messages to a user interface object (telling a label to display a text string, for example, or telling a button to appear or disappear), connect the user interface object to an outlet in your code. The easiest way to make an outlet connection is to Control-click-and-drag from a user interface object on the canvas to the implementation or header file. For example, to create an outlet in your view controller and connect a button to that outlet, Control-drag from the button in Interface Builder to the view controller's implementation file in the assistant editor. Xcode indicates where you can insert an outlet declaration in your code.



Release the Control-drag. The assistant editor displays a Connection menu. From this menu, choose Outlet, type the name of the outlet (`warriorButton` in the screenshot below), and click Connect.



Interface Builder adds the declaration for the outlet to the class. (Outlets are defined as `IBOutlet` properties. The `IBOutlet` keyword tells Xcode that this property can be connected to your storyboard or xib file.)



For more help making connections and establishing cocoa bindings, see *Interface Builder Connections Help*.

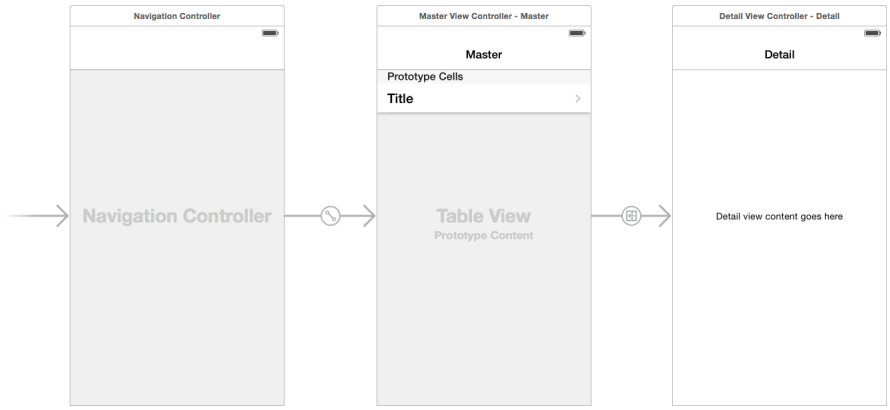
Design the User Interface of Your App with Storyboards

You use a storyboard to graphically lay out the user's path through your iOS or Mac app. Use Interface Builder to specify your user interface in terms of:

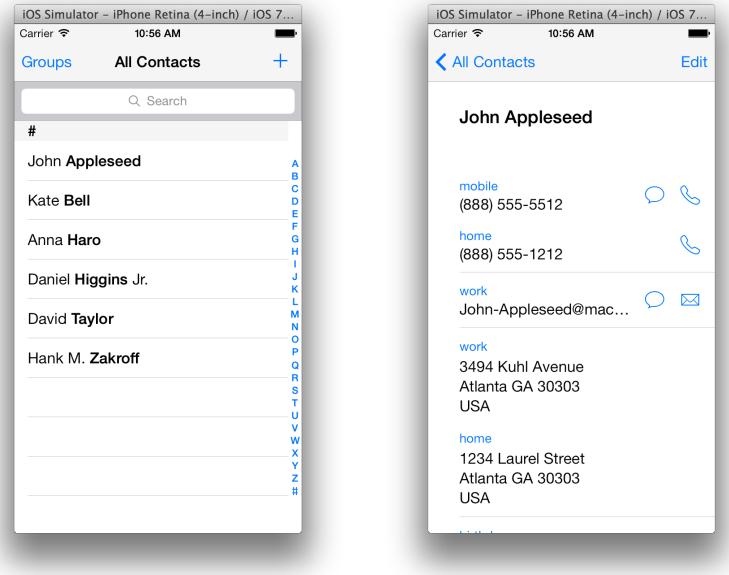
- Scenes
- Segues between scenes
- Controls used to trigger the segues

A **scene** represents an onscreen content area. On iPhone and iPod touch, a screen generally contains a single scene. On iPad and Mac, a screen can be composed of more than one scene. A **segue** represents the transition from one scene to the next scene, such as when one scene slides over another.

The screenshot shows the default storyboard for an iOS project that you create in Xcode with the Master-Detail Application template. This storyboard contains three scenes and two segues. The leftmost scene represents a navigation controller, which manages user navigation between the master and detail scenes. When working from this template, add additional scenes as necessary by dragging view controllers from the Object Library to the canvas and configuring the view controller with the Identity inspector ( ). Drag objects from the object library to lay out each scene. Configure the objects and the segues with the Attributes inspector ( ).



Your master scene might, for example, contain a table listing multiple items. Each item in the master scene has a corresponding detail scene that provides additional information about the item. The navigation controller provides the Back button that returns the user to the master scene from all detail scenes.



For more information about storyboards, see *Storyboard Help*.

Adapt to Multiple iOS Screen Sizes and Orientations with Size Classes

Size Classes enable you to use one storyboard for all different sizes of screens. You build your interface as it will look on most devices, and then update only the objects that need to change when the available screen size changes.

A size class identifies a relative amount of display space for the height and width. The size can be *compact* or *regular* for either height or width. Examples of compact are the width of an iPhone screen in portrait and the height of the screen in landscape. Examples of regular are the height of an iPhone in landscape, and both the width and height of an iPad screen. The specific number of points does not matter, only the relative amount of available space.

Because most of your interface is probably the same for compact or regular, there is an additional size class, *any*. You can lay out most of your interface elements for any width and height, and then lay out any parts of the interface that change for a compact or regular size on either axis.

The size class aware attributes are:

- The constant for a constraint

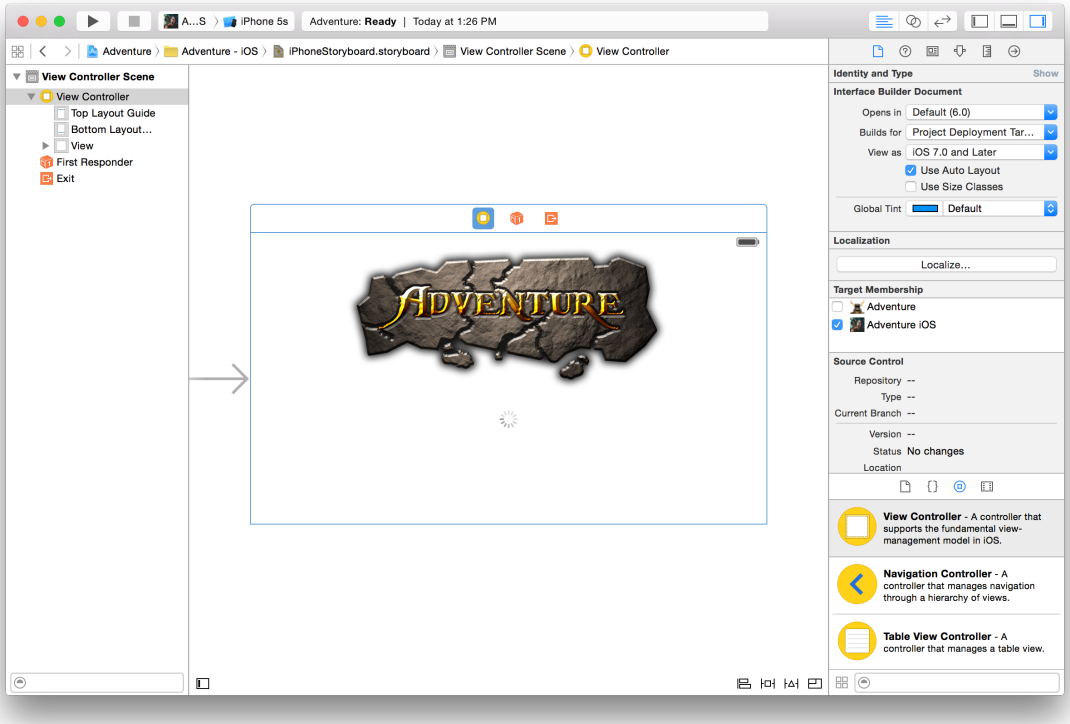
- A Boolean value indicating whether a constraint is installed in the view hierarchy
- A Boolean value indicating whether a view is installed in the view hierarchy
- The font for a text label, field, text view, or button

Some of the user interface changes enabled by modifying these attributes include:

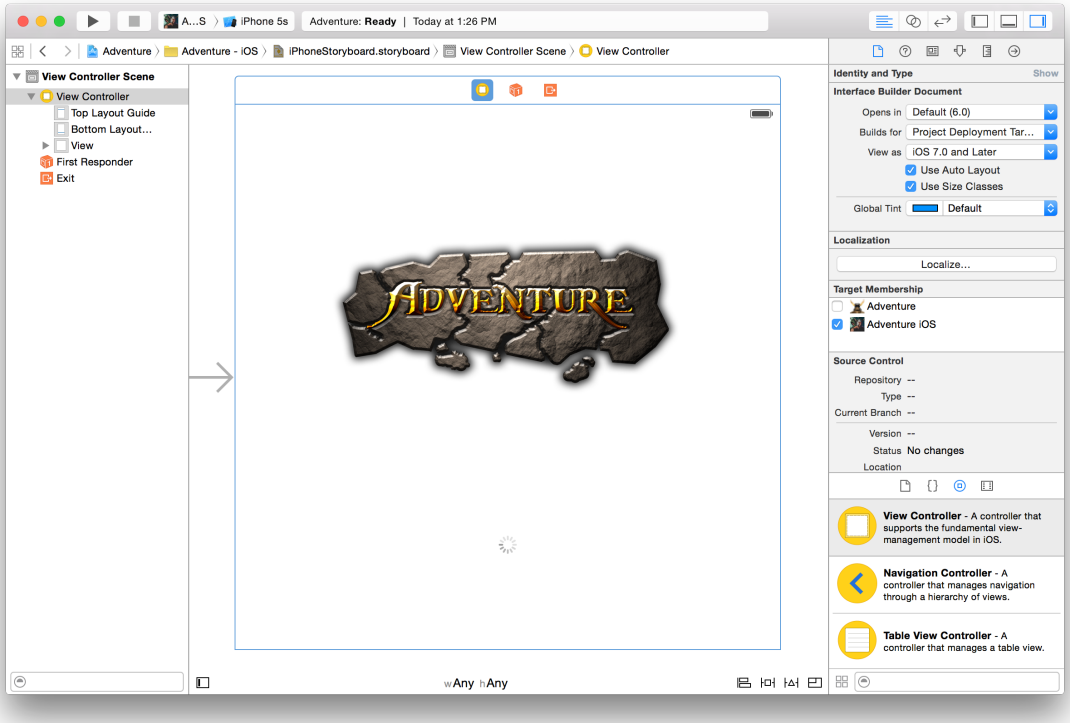
- Changing the size or position of a view
- Adding or removing views
- Adding or removing sets of constraints
- Changing the font in labels, fields, text views, and buttons

Note that if a view or constraint is not installed in the current size class, it is still allocated. Any reference to that view or constraint returns a valid object. Views will not have a superview. If you uninstall a view, Xcode uninstalls all related constraints.

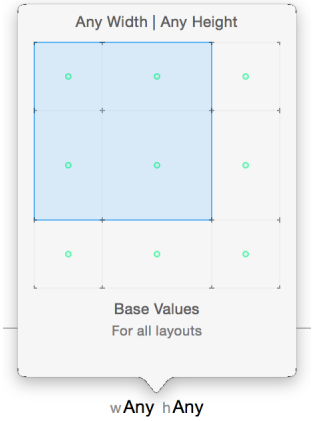
Enable size classes for your storyboard by selecting the storyboard in the project navigator, showing the File inspector, and selecting the Use Size Classes checkbox. Xcode will ask whether you want to convert the storyboard. Turning on size classes also turns on Auto Layout.



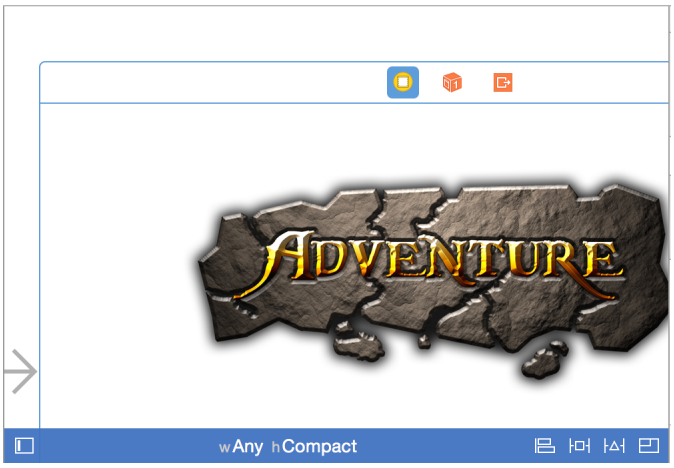
After you enable size classes, Xcode displays all of your top level view controllers on the canvas as squares. There is also a size class control (wAny hAny) at the bottom of the canvas in the center. The control shows the current size class for the width (w) and height (h). In this case, it is any/any.



To change size classes, click the size class control. In the pop-up that appears, move your mouse to the quadrant in the pop-up for the horizontal and vertical size classes you want.



After you choose a new size class, all of the view controllers are resized to reflect the new size class. The bottom bar is shown in blue to remind you that you are no longer editing any/any.



Changes you make to constraints, views, and fonts in a specific size class, are specific for that class. To uninstall a constraint or view, select the constraint or view and press Command-Delete. When you turn off a view, you usually want to turn off all the constraints attached to that view.

For more help on how to see what is active in a particular size class, and for other ways to work with objects, see *Size Classes Design Help*.

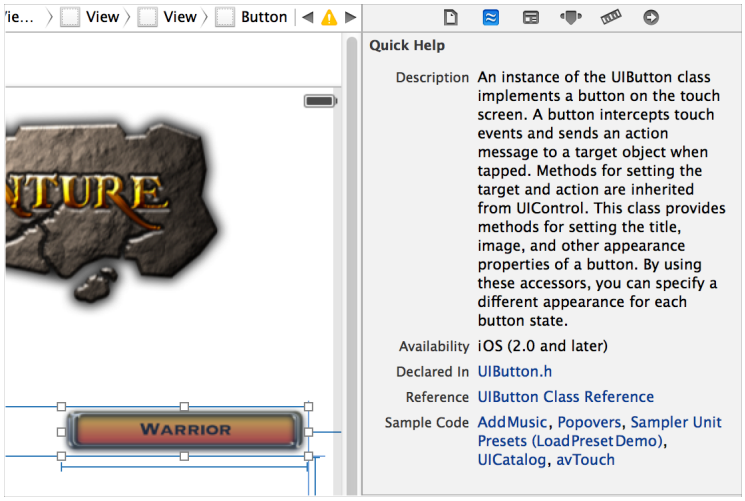
Find and Replace Strings

You can find and replace strings in storyboards and xib files using the built in find commands. This includes finding a string in the storyboard or xib file open in Interface Builder and project wide searches.

For more information on how to search in Interface Builder, see Find and Replace Strings.

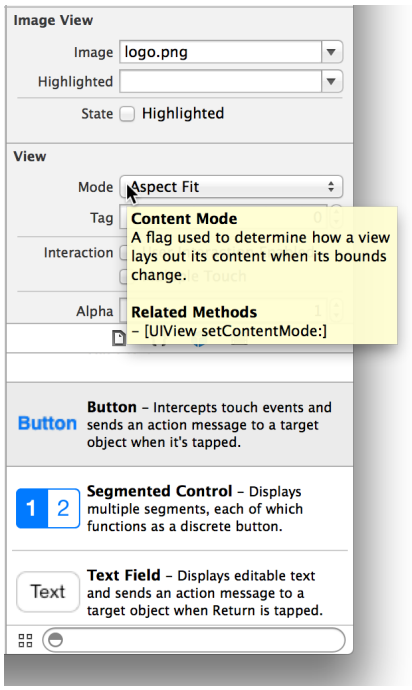
Look Up Documentation for an Object

You can find concise class reference documentation for a user interface object without taking your focus away from Interface Builder. With a file open in Interface Builder, open the utilities area by clicking the right button ( ) in the view selector in the toolbar. Select the Quick Help button ( ) in the inspector bar. In Interface Builder, click the object about which you want information. Documentation appears in the inspector pane of the utilities area.



For complete reference information about the object, click the title of the reference document listed in Quick Help. The reference document opens in the documentation viewer window. You can also open relevant programming guides, sample code, and other related documents by clicking their titles in Quick Help.

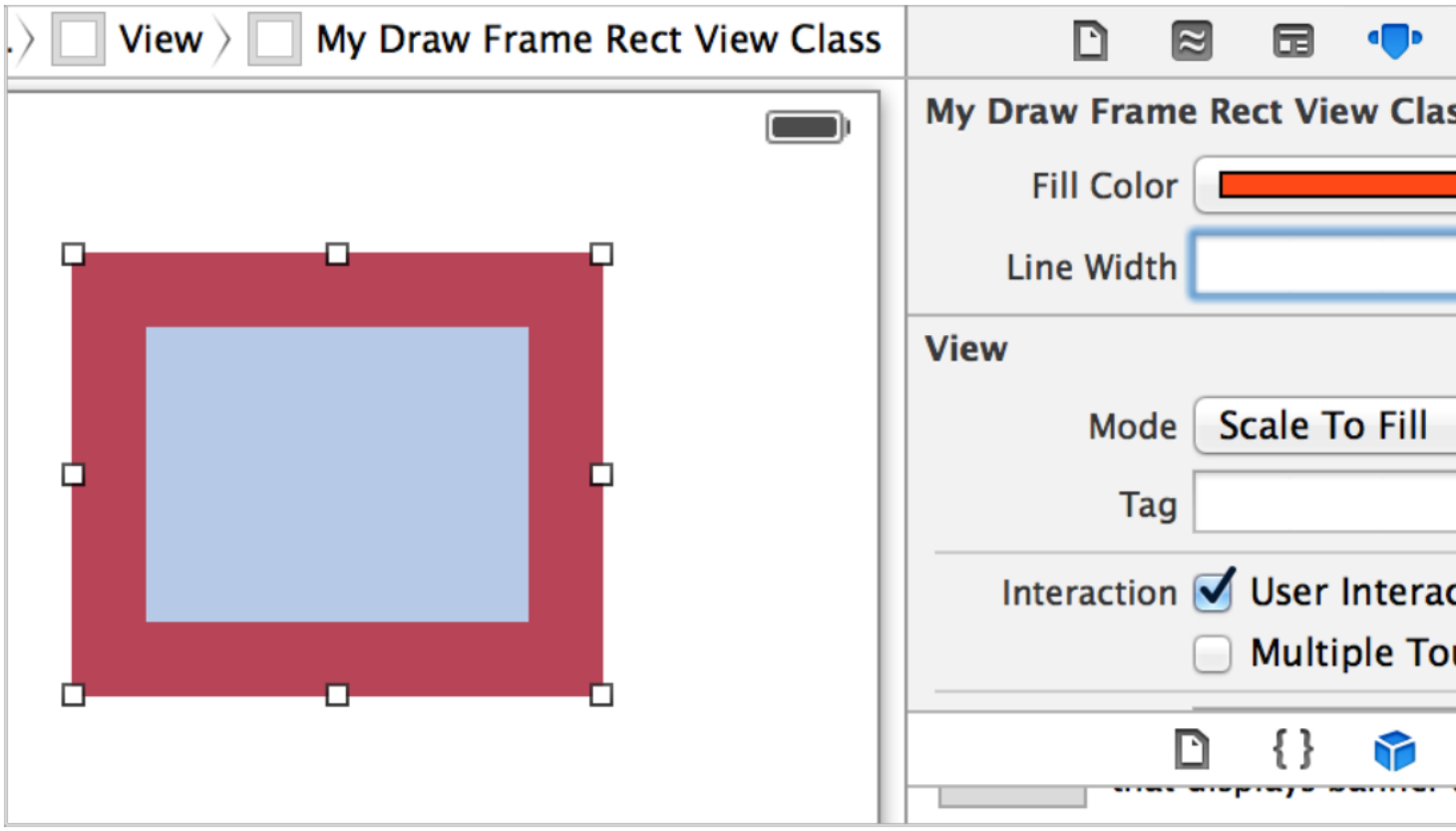
For additional information about settings you configure in the inspectors, move the pointer over a control in an inspector. A help tag appears.



Creating and Rendering Custom View Classes on the Canvas

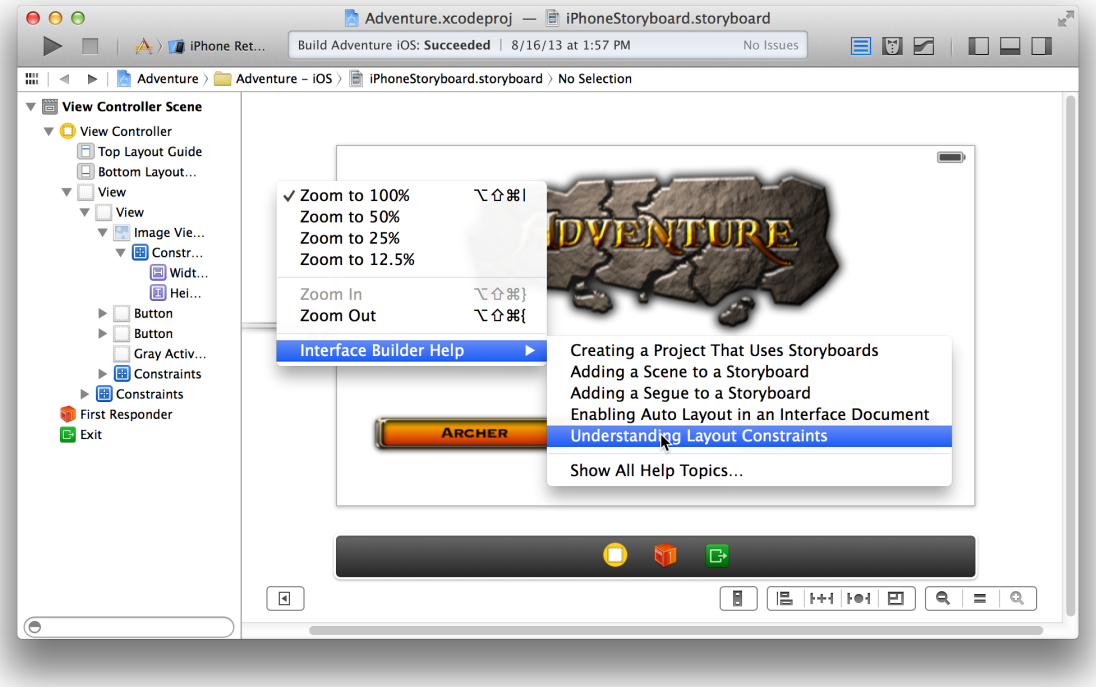
You can render custom view classes in Interface Builder at design time. You can also add properties from your custom view class to the Attributes inspector.

For example, you can add a view class that draws a frame with properties for the color and width of the frame. By following the steps in Creating a Custom View That Renders in Interface Builder the view will draw in Interface Builder and update as the attributes are changes as shown below.



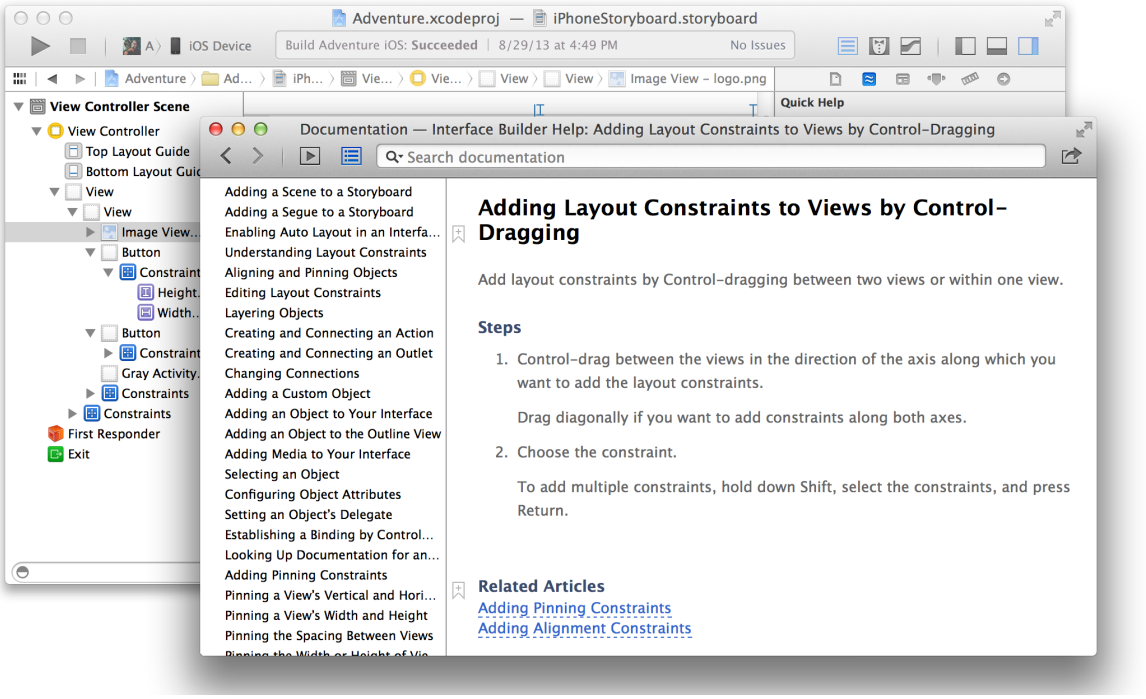
Find Help for Using Interface Builder

Step-by-step instructions for performing common Interface Builder tasks are available directly in Xcode. Control-click anywhere on the Interface Builder canvas to see a short list of the most common operations. Choose Show All Help Topics to see all help articles for the source editor.



Because the Control-click key combination is used by Interface Builder to make connections, you must Control-click on the canvas—not on any object in the user interface—to get the shortcut menu with the list of help articles.

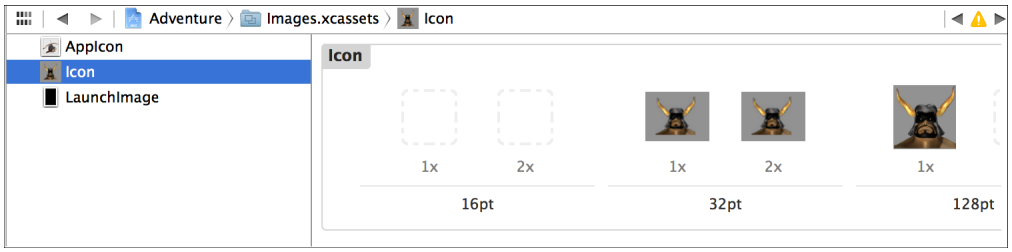
Select a task, and a help article appears in the Xcode documentation viewer window.



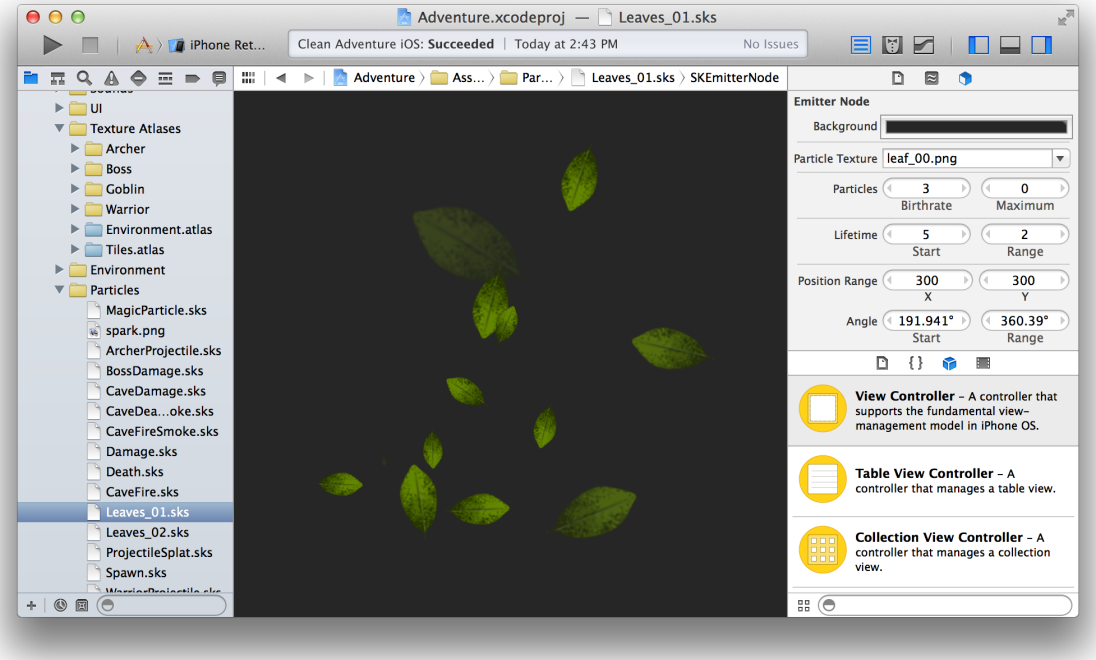
Add Icons, Images, and Effects

To help you create and manage user interface elements for your app, Xcode offers several tools in addition to Interface Builder.

You create many images for your app, including icons, custom artwork, and the launch screens for different iOS devices. Some of these images are required for App Store submission. The asset catalog helps you manage them.



With the particle emitter editor, you can enhance your app by adding animation effects involving moving particles such as snow, sparks, and smoke. These effects are especially useful in games for iOS and Mac.



For Mac apps, the Scene Kit editor helps you work with scenes created in 3D authoring tools and exported as Data Asset Exchange (DAE) files. For more detail, see the *Scene Kit Programming Guide*.

Add App Icons and Launch Images

Create app icons for all of the operating system versions and devices that your app supports. iOS apps and Mac apps require different types of icons. For either platform, add the required versions of your app icons to an asset catalog in Xcode.

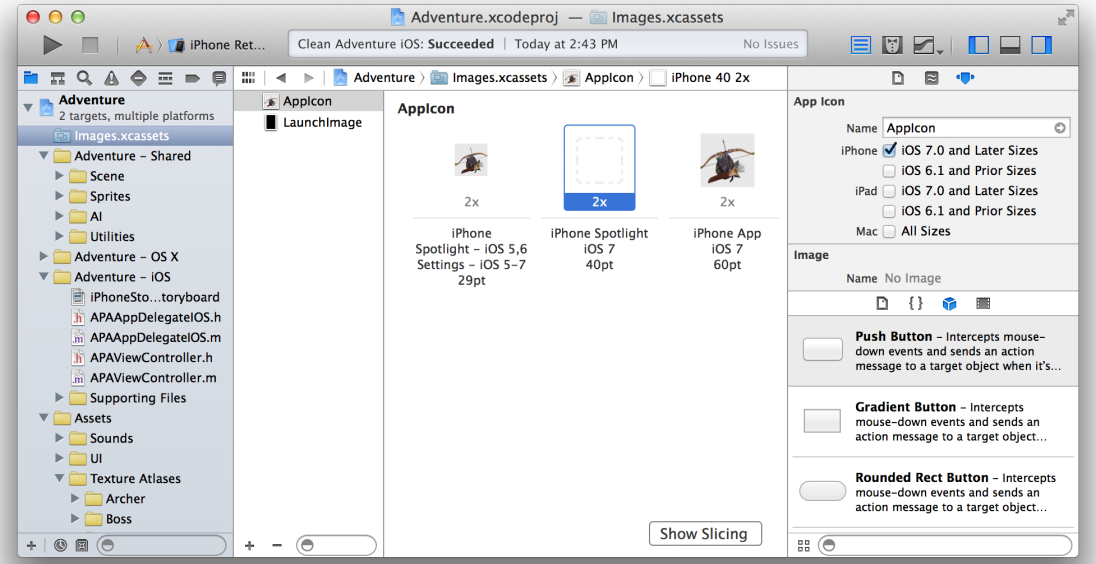
For an iOS app, create an icon to be displayed on a device's Home screen and in the App Store. Xcode doesn't include graphics tools for creating icons; use a graphic design app. Create several different versions of the icon for use in different situations. Your iOS app can include a small icon (to use when displaying search results) and a high-resolution icon (for devices with Retina displays). If your iOS app's target is universal, you also create versions of the icon for iPad and iPhone devices.

For a Mac app, create a set of icons, consisting of pairs of icons (standard and high resolution) for each icon size, in pixels: 16 x 16, 32 x 32, 128 x 128, 256 x 256, and 512 x 512. The Finder uses these icons to represent your app to the user.

Work with Image Assets in the Asset Catalog

When you create a new project, Xcode creates an asset catalog named `Images.xcassets`. Select the asset catalog from the project navigator, and Xcode opens the catalog in the editor area.

The asset catalog contains a list of image sets. Each image set, such as AppIcon in the screenshot, contains all the versions of an image that are necessary to support various devices and scale factors. You can add icon images to your app by dragging them to the appropriate cell in the icon set grid.



You can create additional image sets, such as for buttons and other controls in your app. To create an empty image set or to import images into a new set, click the Add button (+) at the bottom of the image set list.

Create and Set the iOS Launch Images or Launch Screen File

A launch screen is displayed while your app is launching on iOS. The launch screen is displayed as soon as the user taps your app icon, and it stays on the screen until your main interface is displayed. If your app is running on iOS 8 or later, the system uses a launch screen from a xib file and sizes it appropriately for the screen. For deployment targets prior to iOS 8, you add a set of launch images to an asset catalog for each of the possible screen sizes.

New projects are created with a launch screen file called `LaunchScreen.xib`. Alternately, you can create a new launch screen file using File > New, selecting the User Interface category, and choosing a file type of Launch Screen. The launch screen uses size classes to adapt to different screen sizes and orientations, see Adapt to Multiple iOS Screen Sizes and Orientations with Size Classes (page 75) for more information.

Because the launch screen is shown before your app is running, you can only use a single root view of type UIView or UIViewController. You are also limited to UIKit classes that do not require updating. For more information, see Creating a Launch Screen File.

To set the launch screen, open the General information tab for your target, and select the launch screen file from the pop-up menu.

For more help icons, launch images, and the asset catalog, see *Asset Catalog Help*.

Create and Set iOS Launch Images for iOS 7 and Earlier

You can easily capture screenshots for launch images on a device. On the device, configure the screen the way you want it to appear. Then press the device Lock and Home buttons simultaneously. Your screenshot is saved in the Saved Photos album in the Photos app. Copy the screenshot from the device to your Mac. You can use the iPhoto app, for example, to import the screenshot from the device and then export the screenshot to your Mac as a PNG file.

To set the screenshot as a launch image, select the asset catalog file in the project navigator, and select the LaunchImage set. Drag your screenshot to the appropriate cell in the grid.

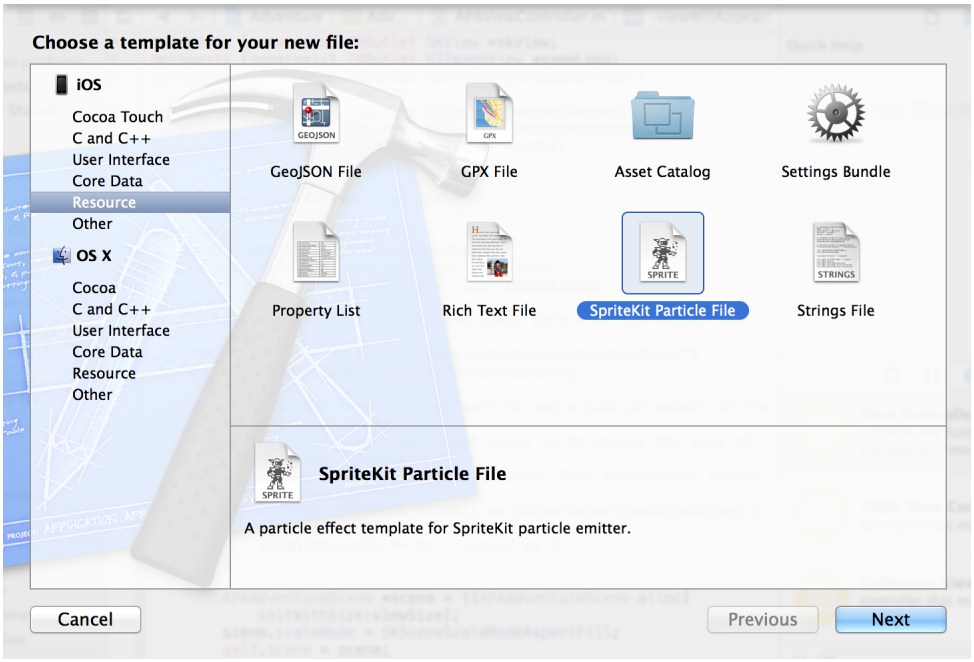
For more information on help icons, launch images, and the asset catalog, see *Asset Catalog Help*.

Add Particle Emitter Effects

Especially useful for developers of iOS and Mac games, Sprite Kit provides a graphics rendering and animation infrastructure. This infrastructure includes particle emitters. Particle emitters can range from a single image that barely moves, to thousands of small particles flying across the screen. You can use particle emitters to simulate fire, rain, smoke, snow, sparks, and other animated effects.

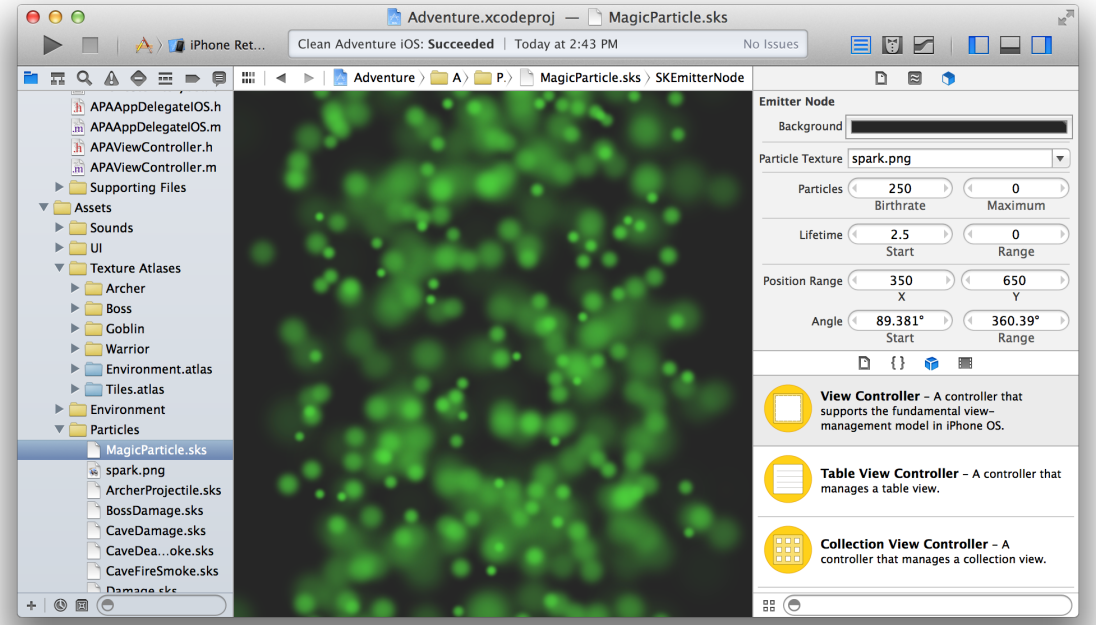
Xcode provides eight particle emitter templates and an editor for manipulating the appearance and behavior of particles.

Create a Sprite Kit–enabled game from the New Project template in Xcode, or use the General pane in the project editor to add the Sprite Kit framework to an existing target. To add a particle emitter to your project, choose File > New > File, and then choose Resource > SpriteKit Particle File.



Select the particle template from the drop-down menu, and click Next. Enter a name for the emitter in the Save As field. Select the checkbox associated with your project in the Targets area. Xcode creates a file with the extension `.sks`.

Select your particle emitter file in the project navigator, and Xcode opens the file in the particle emitter editor.



Modify the look and feel of the particles with the particle emitter inspector ( ). For example, you can change the rate at which particles are created, what a particle looks like, and how it acts after it is created. Changes made to the inspector take effect immediately and can be viewed in the editor.

For more detail, see the *Particle Emitter Editor Guide* .

Add 3D Scenes to Your App

Scene Kit is a 3D-rendering framework for iOS and Mac apps. Sprite Kit supports the import, manipulation, and rendering of 3D assets without requiring advanced 3D graphical programming skills on your part. With the Scene Kit editor, you can preview 3D scenes, inspect them for information needed for your source code, and adjust scene object parameters to enhance and fine-tune the rendering for your app.

To import a digital asset exchange (DAE) file into the project, use the project navigator. Select a folder in which you want to save the file. Choose File > Add Files, select the file, and click Add. To browse the 3D scene in Xcode, select the DAE file in the project navigator. Xcode opens the file in the Scene Kit editor.

To preview the scene and run animations, use the controls in the Scene Kit editor's main area. Press the Play button to play an animation. Click the Pause button to pause the animation, and drag the slider to scroll through it. Use the trackpad or mouse to manipulate the point of view.

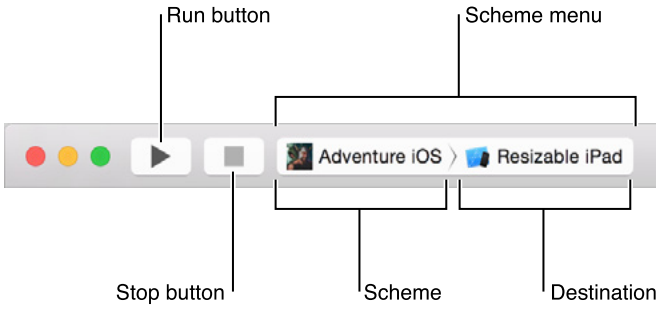
The inspectors in the utilities area allow you to view and edit information about the node in the scene graph list or the object in the entities list. For example, with the nodes attributes inspector, you can adjust camera, light, or geometry attributes, and with the materials inspector, you can adjust many settings on a material and its properties, such as by selecting a lighting model for it and colors and textures for its contents.

Find More Help

See *Asset Catalog Help*, *Particle Emitter Editor Guide*, and *SceneKit Editor Help*.

Run Your App

To build and run your iOS or Mac app, choose a scheme and a run destination in the workspace toolbar, and click the Run button. Clicking the Stop button causes your app to quit.



If you are running an iOS app, Xcode launches it either in iOS Simulator or on an iOS device connected to your Mac. If you are running a Mac app, Xcode launches it directly on your Mac.

Choose a Scheme to Build Your App

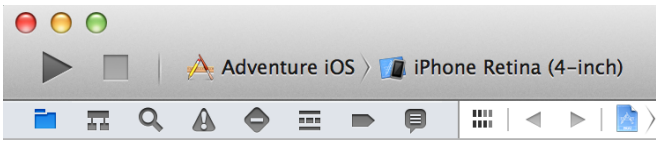
A **scheme** is a collection of settings that specify the targets to build for a project, the build configuration to use, and the executable environment to use when the product is launched. When you open an existing project (or create a new one), Xcode automatically creates a scheme for each target. The default scheme is named after your project and includes settings to perform five actions:

- Run the app.
- Run unit tests against the target.
- Profile the app's performance characteristics.
- Perform static analysis on the code.
- Archive the app for distribution, such as sending to testers or submitting to the App Store.

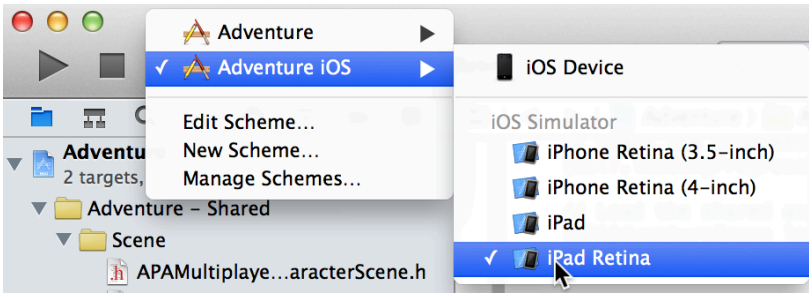
Each action includes building the app as an executable product. To choose the scheme, use the **Scheme menu** in the Xcode workspace toolbar. (You'll use the Scheme menu to choose a destination, too.)

Choose a Destination to Run Your App

When you build an app, the **destination** determines where the app runs after it's built. For Mac apps, the destination is the Mac on which the app is built. For iOS apps, the destination can be a provisioned iOS device connected to the Mac, or iOS Simulator. Installed as part of the Xcode tools along with the iOS SDK, **iOS Simulator** runs on your Mac and simulates an iPhone or iPad environment.



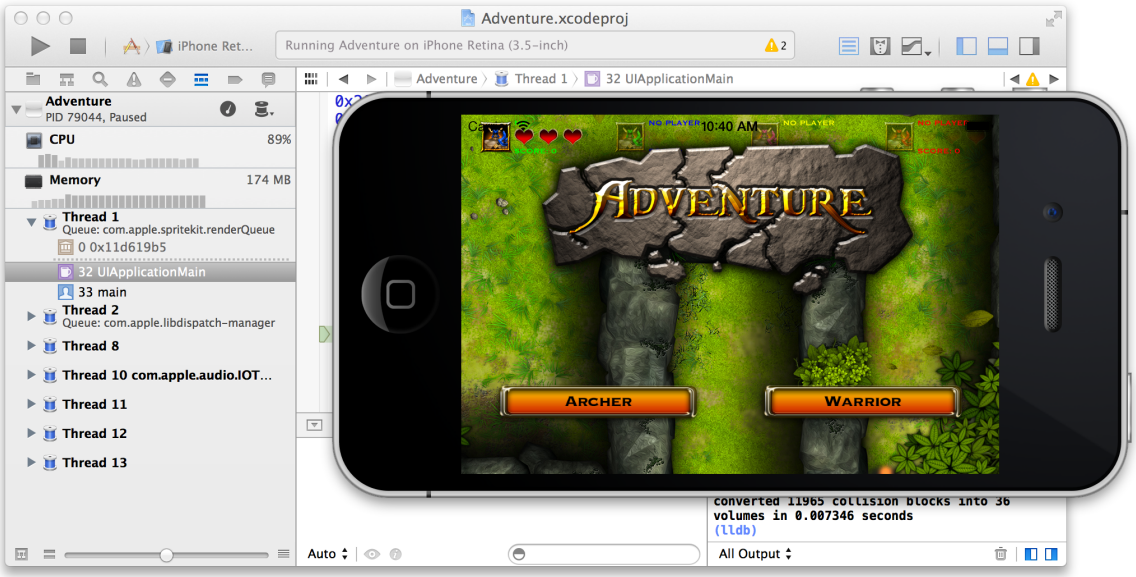
The Scheme menu lets you select a combination of scheme and destination, but the two settings are distinct. A scheme does not include a destination. In the screenshot above, Adventure iOS is selected as the scheme, and the iPhone Retina (4-inch) simulation environment is selected as the destination. As a result, the Adventure iOS scheme builds an iOS executable that runs on a simulated iPhone in OS Simulator. As shown below, the same scheme could be used to run the app on a different destination, such as a simulated iPad or a connected iOS device.



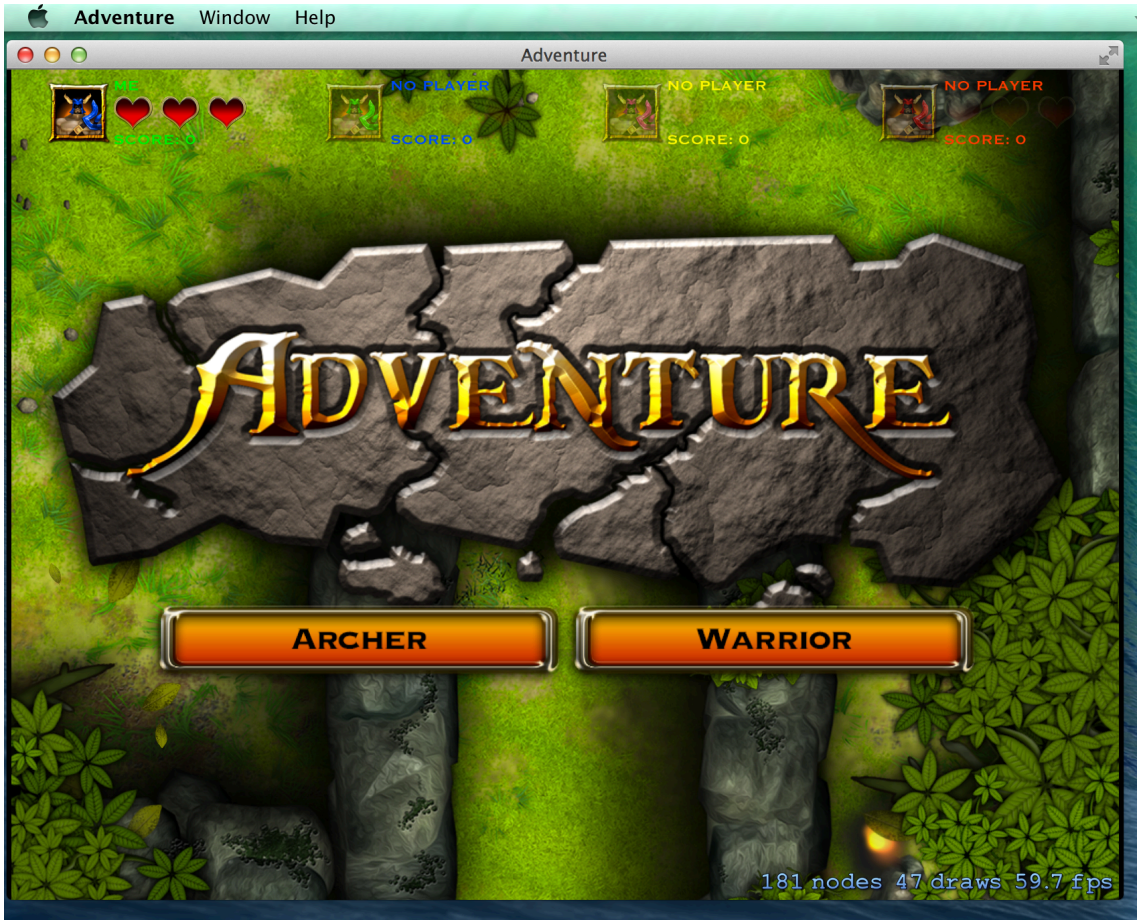
Run Your App

Click the Run button in the workspace toolbar to compile, link, and execute your code. If the app builds successfully, Xcode runs it and starts a debugging session.

Depending on your destination, Xcode launches your iOS app either in iOS Simulator or on a connected iOS device.



Xcode launches a Mac app on your development Mac.



Xcode displays any errors or warnings it encounters in the issue navigator, available by clicking in the navigator bar. If there are errors during the compilation or link phase, Xcode doesn't run your code.

Run Your App in iOS Simulator

iOS Simulator enables you to simulate several iPhone and iPad devices and several versions of the iOS operating system. You interact with iOS Simulator by using the keyboard and trackpad to emulate taps, device rotation, and other actions. For example, you can use the Hardware menu in iOS Simulator to:

- Rotate the simulator to the left and right
- Simulate a user shaking the device
- Send the frontmost app a simulated low-memory warning

As a preliminary tool for use before testing your app on devices, iOS Simulator allows you to prototype and test builds of your iOS app during the development process. Although you can test your app's basic behavior in iOS Simulator, the simulator is limited as a test platform. While developing your app, it is essential that you run and test it on connected iOS devices.

For more detail on using the simulator, see *iOS Simulator User Guide*.

Run Your App on a Connected Device

To run your iOS app on a device (an iPad, iPhone, or iPod touch) during development, four things are required:

- The device is connected to your Mac.
- You are a member of an Apple developer program.
- You have a valid signing identity for the developer program.
- The device is provisioned for development use by that developer program.

Xcode guides you through any missing parts of these requirements and can usually do the work of obtaining a signing identity and device provisioning profile.

To run your iOS app on a device (an iPad, iPhone, or iPod touch) during development, the device must be connected to your Mac, and the device must be provisioned for development by Apple. If your Mac app uses certain Apple technologies—such as iCloud, Game Center, and In-App Purchase—your Mac must be provisioned.

Apple implements an underlying security model to protect user data and to protect your app from being modified and distributed without your knowledge. Throughout the development process, you create assets and enter information that Apple uses to verify the identify of you, your devices, and your apps. These assets include provisioning profiles, which identify your development devices.

To obtain a provisioning profile for a device, you need an Apple Developer Program membership and associated signing identity. For detailed information on doing this, see *App Distribution Quick Start*.

Choose Your Device for the Run Destination

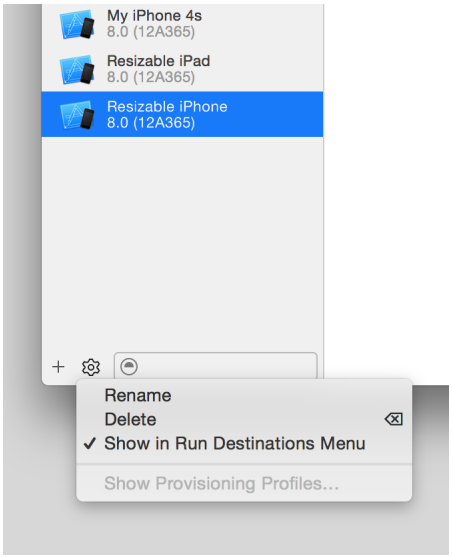
When you plug the device into your Mac, the device's name and the iOS release it is running appear as a destination in the Scheme menu. Choose your device as the destination, and then click the Run button to build and run your app on the device.

Create Custom Simulator Configurations

Choose Window > Devices to open the Devices organizer. Click the Add button (+) in the bottom-left of the organizer window. In the dialog that appears, type in a name for your custom simulator configuration, choose a device type, and then choose an iOS version. Click Create and your new custom simulator configuration is added to the Simulator list. By default, the new configuration appears in the Run Destinations menu.

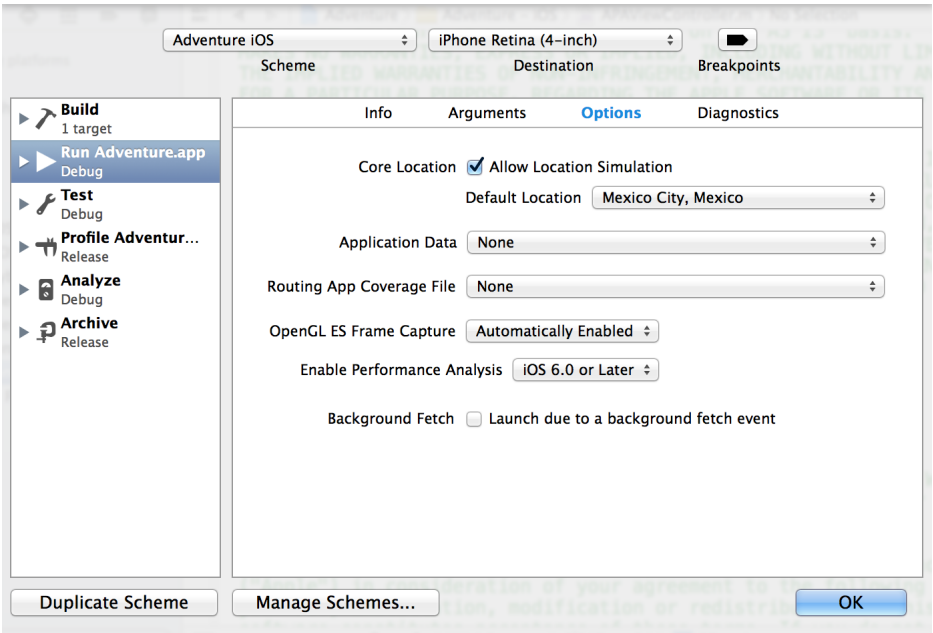
Show Simulators or Devices in the Run Destinations Menu

Choose Window > Devices. In the Devices organizer, select the item you want to add or remove from the target menu. Choose the Configuration button (⚙) in the bottom-left of the organizer window. Choose Show in Run Destinations Menu. A checkmark next to that menu item indicates that the simulator or device will be shown in the Run Destinations menu.



Edit, Create, and Manage Schemes

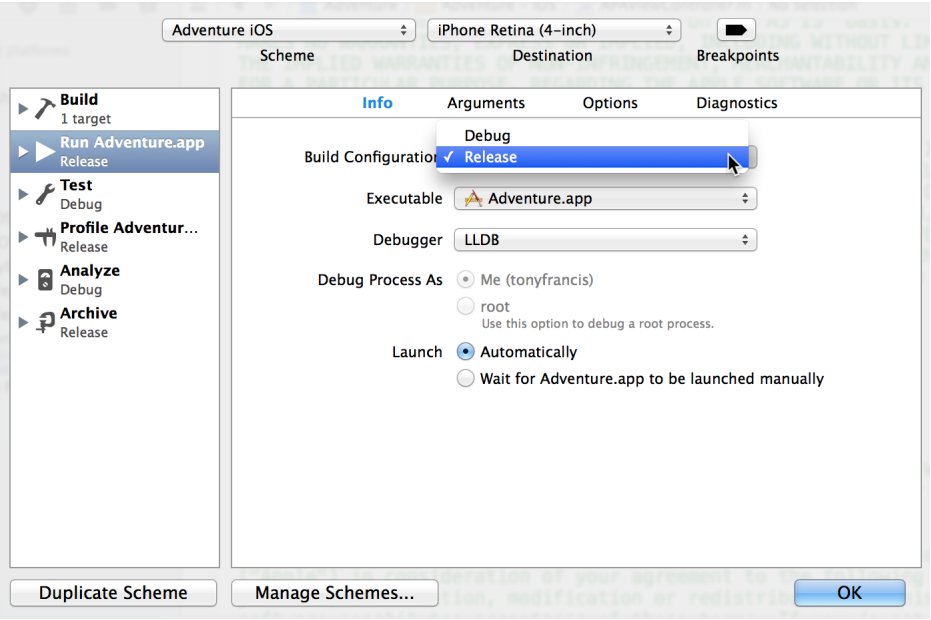
To edit a scheme, choose Edit Scheme from the Scheme menu. The left column of the scheme configuration dialog lists the actions that the scheme can perform. You can modify settings for each action. In the screenshot, the Run action is modified to simulate the location of Mexico City when Xcode launches the app.



You can edit a scheme so that it performs such actions as:

- Building multiple targets
- Executing scripts before or after any action
- Sending emails before or after any action
- Running with memory management diagnostics

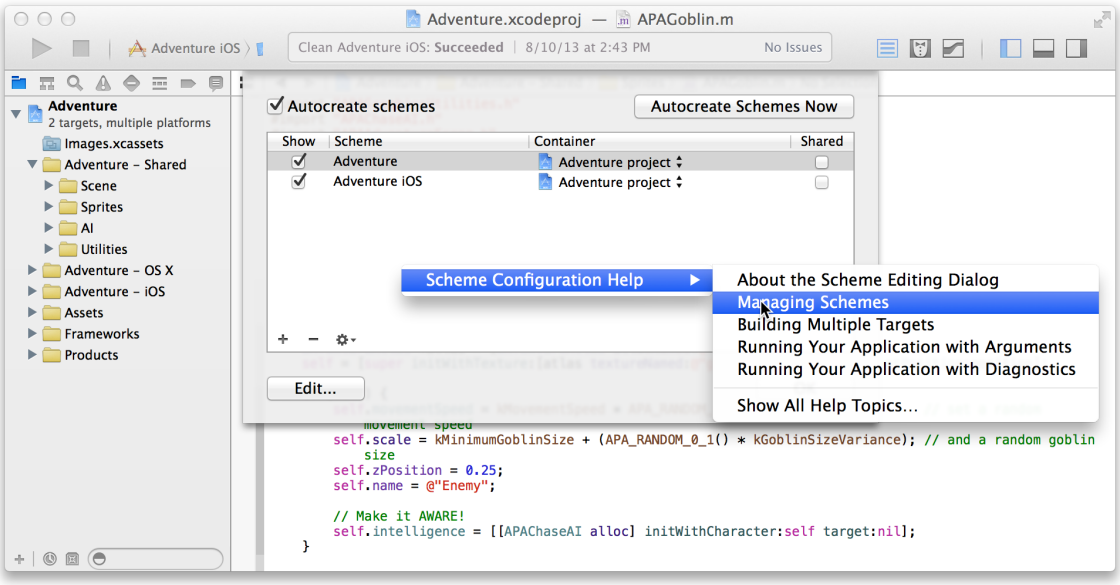
- Producing either a debug or release build for any action, such as for the Run action in the screenshot below



A convenient way to create a new scheme is to click the Duplicate Scheme button. This button uses the active scheme as a template for you to rename, edit, and save.

If you create schemes, you can manage them by clicking the Manage Schemes button in the scheme configuration dialog or by choosing Manage Schemes from the Scheme menu as shown in the screenshot below. You can rename or reorganize how schemes appear in the Scheme menu. You can also specify whether a scheme should be displayed in the menu, where a scheme is stored in the project or workspace, and whether

a scheme should be shared, such as with team members accessing a project from a source code repository. You can click the Autocreate Schemes Now button to make Xcode create schemes for any targets that don't have them.

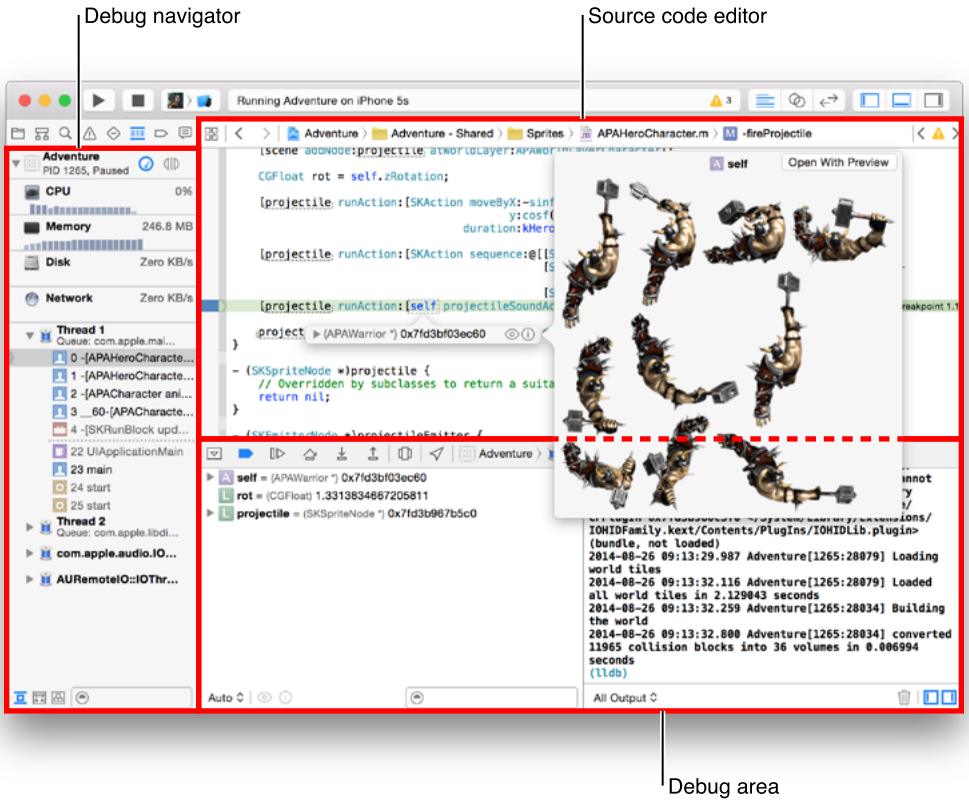


Get step-by-step instructions for creating, editing, and managing schemes by pressing Control-click anywhere in the scheme configuration dialog, or see *Scheme Editor Help* .

Debug Your App

After you click the Run button in the workspace toolbar and your app builds successfully, Xcode runs your app and starts a debugging session. You can debug your app directly within the source editor with graphical tools such as data tips and Quick Look for the value of variables.

The debug area and the debug navigator let you inspect the current state of your running application and control its execution.



Creating a quality app requires that you minimize your application's impact on your users' systems. Use the debug gauges in the debug navigator to gain insight into your app's resource consumption, and when you spot a problem, use Instruments to measure and analyze your app's performance.

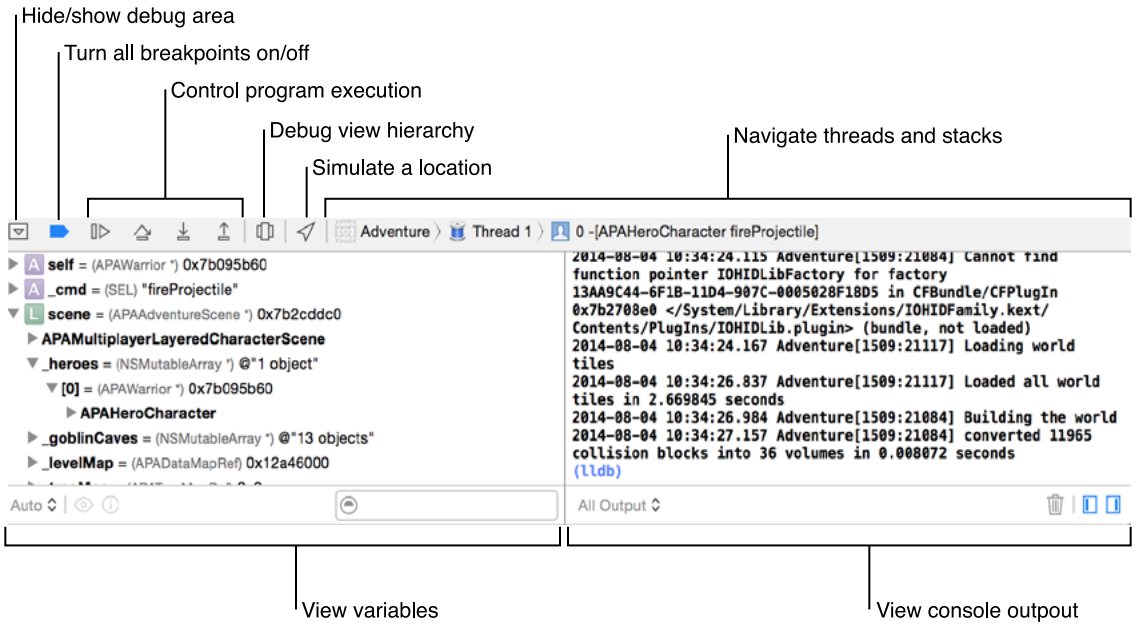
If you are developing an iOS app, use iOS Simulator to find major problems during design and early testing.

You can configure Xcode to help you focus on your debugging tasks. For example, when your code hits a breakpoint, you can make Xcode automatically play an alert sound and create a window tab named Debug, where Xcode displays the debug area, the debug navigator, and your code at the breakpoint.

Control Execution and View State Information

Xcode lets you step through your code line by line to view your program's state at a particular stage of execution. Use the debug area to control the execution of your code, view program variables and registers, view its console output, and interact with the debugger. You can also use the debug area to navigate the OpenGL calls that render a frame and to view the rendering-state information at a particular call.

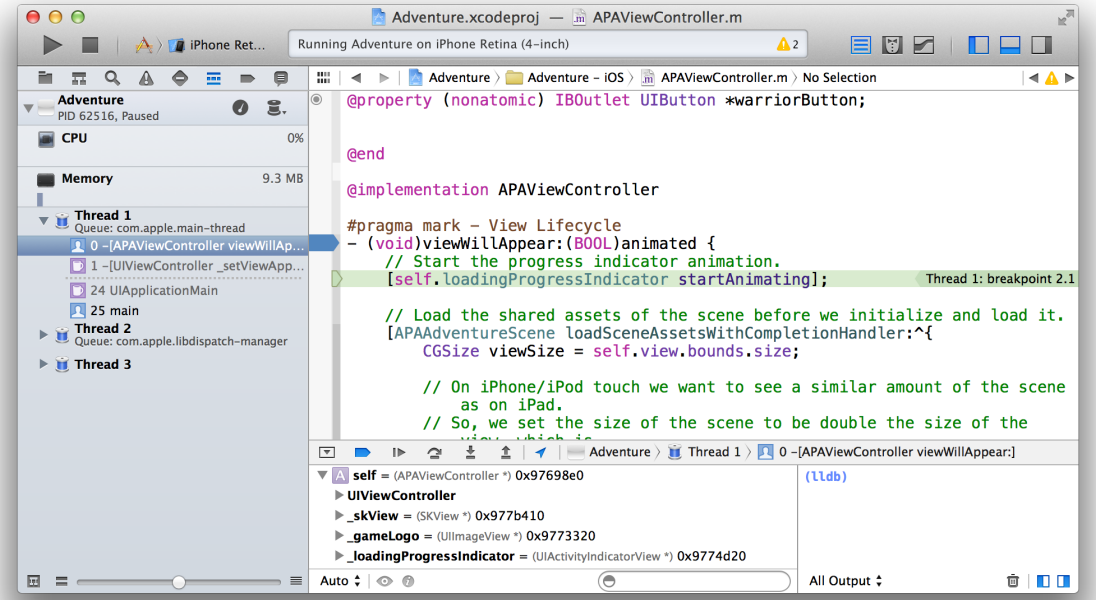
Display the debug area by clicking the center button ( ) in the view selector in the workspace window toolbar.



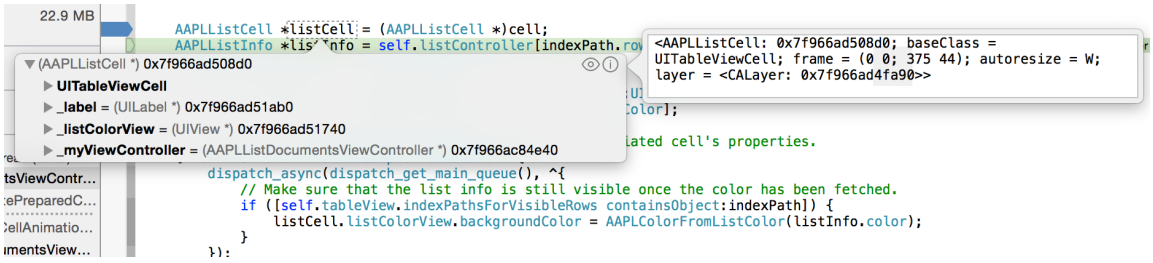
You can suspend the execution of your app by clicking the pause button (which toggles between to pause and to continue) in the debug area toolbar. To set a breakpoint, open a source code file and click the gutter next to the line where you want execution to pause. A blue arrow ( ) in the gutter indicates the breakpoint. For more information on breakpoints, including how to set breakpoint actions and the different kind of breakpoints, see *Breakpoint Navigator Help* .

When your app is paused, the currently executing line of code is highlighted in green. You can step through execution of your code using the Step Over ( ), Step Into ( ), and Step Out ( ) buttons located in the bar at the top of the Debug area. Step over will execute the current line of code, including any methods. If the current line of code calls a method, step into starts execution at the current line, and then stops when it reaches the first line of the called method. Step out executes the rest of the current method or function.

When execution pauses, the debug navigator opens to display a stack trace. Select an item in the debug navigator to view information about the item in the editor area and in the debug area. As you debug, expand or collapse threads to show or hide stack frames.



Hover over any variable in the source code editor to see a data tip displaying the value for the variable. Click the Inspector icon ( ) next to the variable to print the Objective-C description of the object to the debug area console and to display that description in an additional popover.

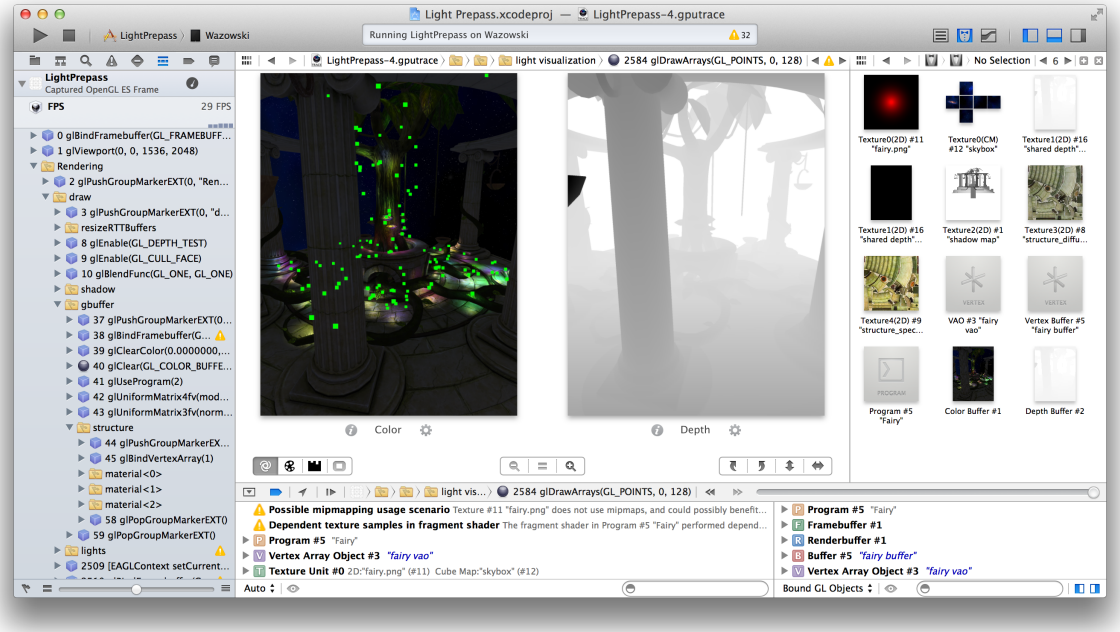


Click the Quick Look icon ( ) to see a graphical display of the variable's contents. You can implement a custom Quick Look display for your own objects. See *Quick Look for Custom Types in the Xcode Debugger*.

When you build and run an OpenGL ES application on a connected device, the debug area toolbar includes a Frame Capture button ( ). Click that button to capture a frame. You can use OpenGL ES frame capture to:

- Inspect OpenGL ES state information
- Introspect OpenGL ES objects such as view textures and shaders
- Step through the state calls that precede each draw call and watch the changes with each call
- Step through draw calls to see exactly how the image is constructed
- See in the assistant editor which objects are used by each draw call
- Edit shaders to see the effect upon your application

The screenshot shows the use of the debugger to view components of a rendered frame. The debug navigator on the left shows parts of the rendering tree, and the main debug view shows the color and depth sources for the rendered frame as well as other image sources.



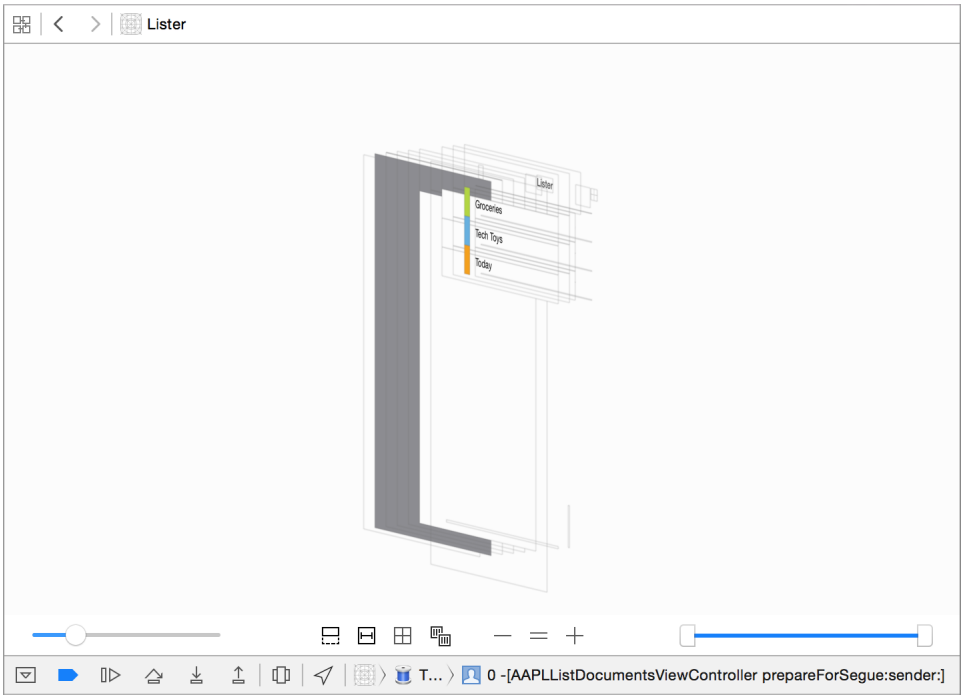
For more help debugging OpenGL ES, see related items in *Debug Navigator Help* and *Debug Area Help*.

Examine Your App's View Hierarchy at Runtime

Click the Debug View Hierarchy button ( ) in the bar at the top of the debug area to inspect a 3D rendering of the view hierarchy of your paused app. You can:

- Rotate the rendering by clicking and dragging in the canvas.
- Increase or decrease the spacing between the view layers using the slider on the lower left.
- Change range of visible views using the double ended slider on the lower right. Move the left handle to change the bottom-most visible view. Move the right handle to change the top-most visible view.
- Reveal any clipped content of the selected view by pressing the Show clipped content button ( ).
- Reveal any Auto Layout constraints of the selected view by pressing the Show constraints button ( ).

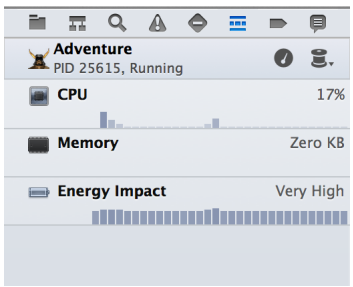
- Increase and decrease the magnification using the Zoom In (+) and Zoom Out (-) buttons.



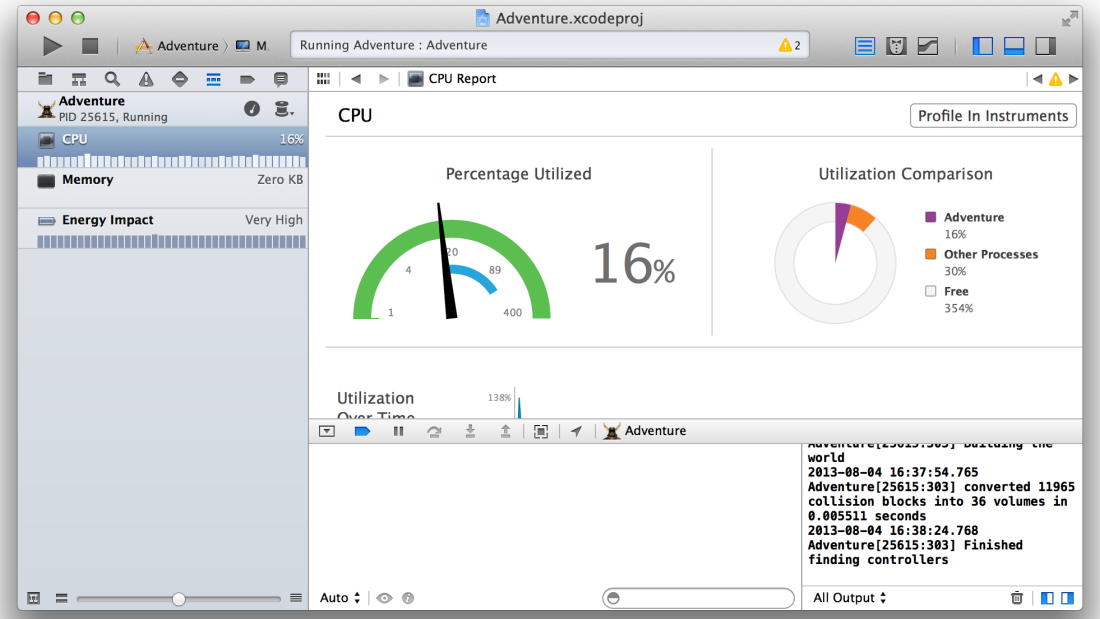
For more help debugging views, see Debugging Views

Examine Your App's Impact on System Resources

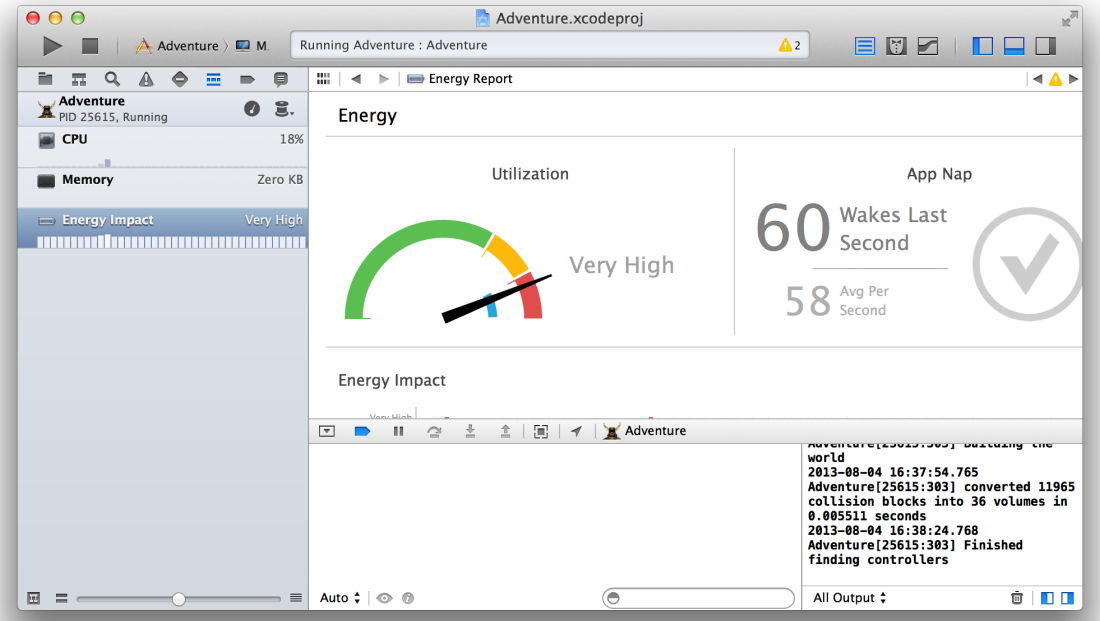
The debug navigator displays gauges that provide insight into how your app is performing. For example, the CPU gauge shows a readout of your app's CPU usage, making it easy to spot unexpected spikes. Depending on the capabilities of your app and the characteristics of its destination, gauges can report your app's impact on memory, iCloud, OpenGL ES, energy, and the CPU.



To see a full report, click a gauge in the debug area. To perform a deeper analysis of your app's performance, click the Profile in Instruments button.



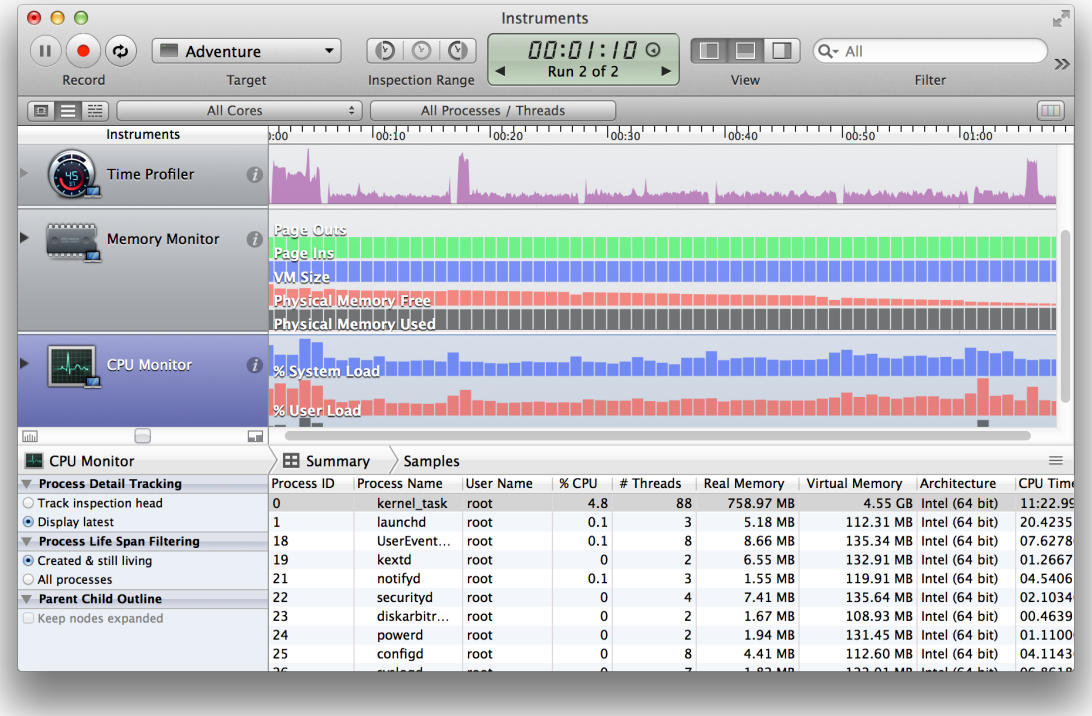
For problem areas, the energy report offers a preliminary diagnosis of what may be plaguing your app.



For more help, see Using Debug Gauges.

Measure Your App's Performance

The Instruments app, which is included with Xcode, gathers data from your running app and presents it in a graphical timeline. With Instruments, you can gather data about performance areas such as your app's memory usage, disk activity, network activity, and graphics operations. By viewing the data together, you can analyze different aspects of your app's performance to identify potential areas of improvement. You can also automate the testing of your iOS app's user interface elements.



There are several ways to start Instruments from Xcode. For example:

- Click the Profile in Instruments button from a debug gauge report.
- Choose Product > Profile.
- Specify an Instrument in the Profile action for a scheme.

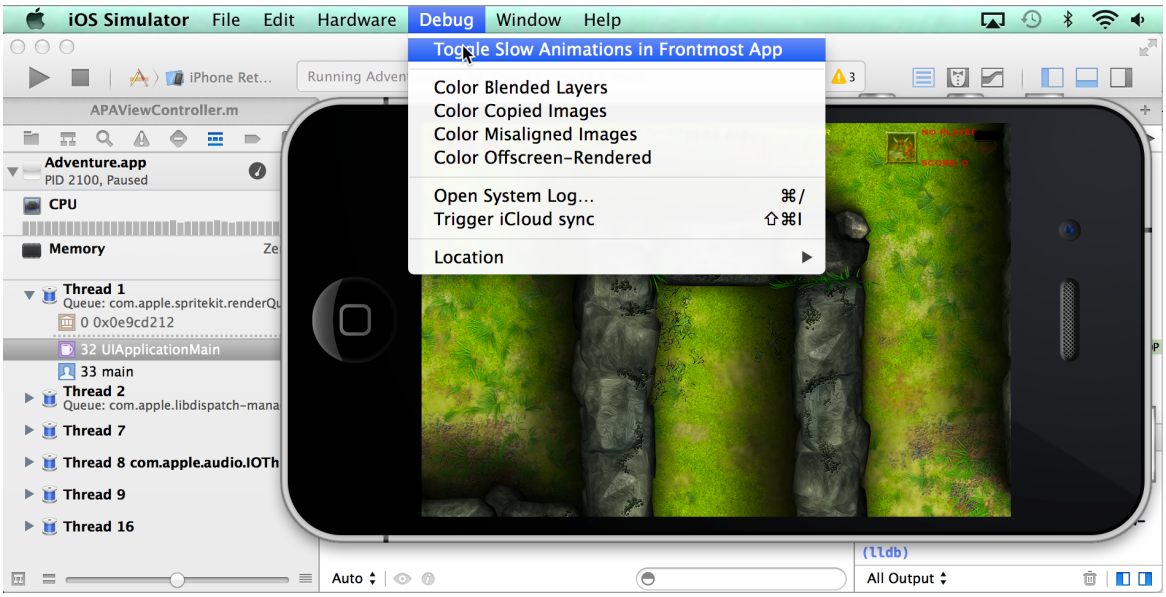
The Instruments app uses individual data collection modules, known as *instruments*, to gather data about a process over time. The Instruments app includes a library of templates. Each template contains instruments for obtaining a set of related information.

For more detailed information, see *Performance Overview* and *Instruments User Guide*.

Perform Early Testing in iOS Simulator

iOS Simulator helps you find major problems in your app during design and early testing. For example, the Debug menu in iOS Simulator offers tools that help you:

- Slow an animation to spot any problems
- Trigger iCloud sync
- Identify blended view layers that harm app performance
- Identify images whose source pixels aren't aligned to the destination pixels
- See what content is rendered offscreen
- Simulate different locations



In every simulated environment in iOS Simulator, the Home screen provides access to apps—such as Safari, Contacts, Maps, and Passbook—that are included with iOS on the device. You can perform preliminary testing of your app's interaction with these apps in iOS Simulator. For example, if you are testing a game, use iOS Simulator to test that the game uses Game Center correctly.

The Accessibility Inspector in iOS Simulator helps you test the usability of your app regardless of a person's limitations or disabilities. The Accessibility Inspector displays information about each accessible element in your app and enables you to simulate VoiceOver interaction with those elements. To start the Accessibility Inspector, click the Home button on iOS Simulator. Click Settings and go to General > Accessibility. Slide the Accessibility Inspector switch to On.

You can test your app's localizations in iOS Simulator by changing the language. In Settings, go to General > International > Language.

Although you can test your app's basic behavior in iOS Simulator, the simulator is limited as a test platform for multiple reasons. For example:

- Because iOS Simulator is an app running on a Mac, iOS Simulator has access to the computer's memory, which is much greater than the memory found on a device.
- The iOS Simulator runs on the Mac CPU rather than the processor of an iOS device.
- iOS Simulator doesn't run all threads that run on devices.
- iOS Simulator can't simulate hardware features like the accelerometer, gyroscope, camera, or proximity sensor.

While developing your app, run and test it on all of the iOS devices and iOS versions that you intend to support.

For more detailed information, see Testing and Debugging in iOS Simulator.

Customize Your Debugging Workflow

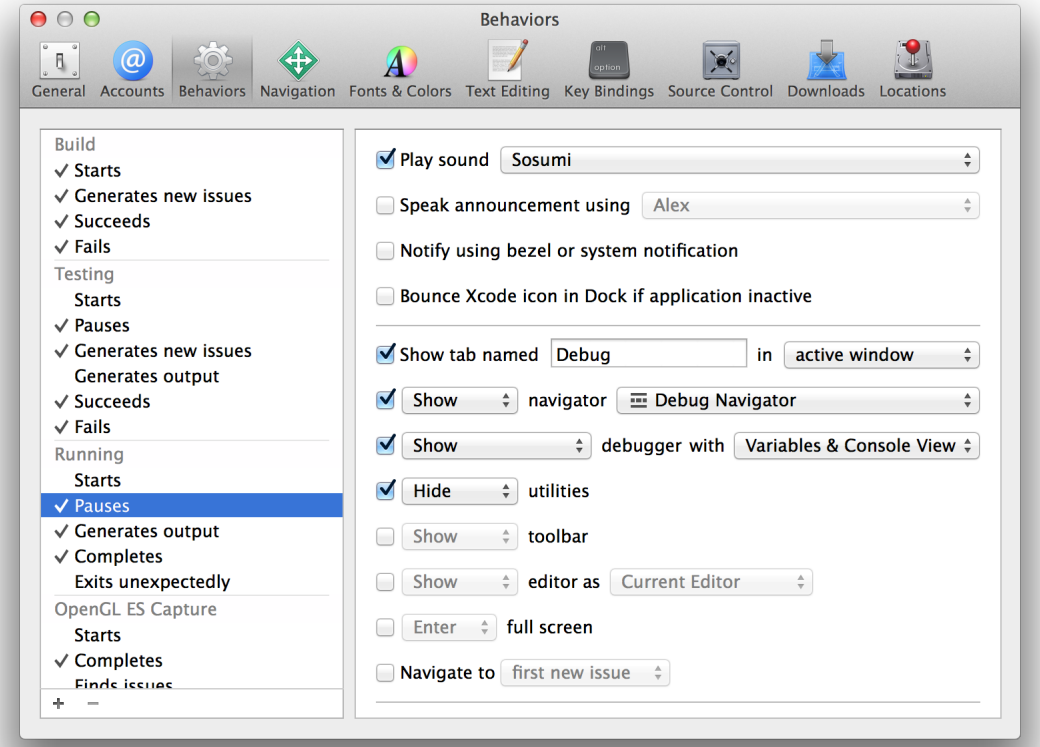
Specify behaviors that affect your workflow through the Xcode Behaviors preferences. Choose Xcode > Behaviors to specify what should happen when a variety of events occur while building, running, and debugging your app.

For example, Xcode can display the debug area when your code pauses at a breakpoint, and it can display the issue navigator when a build fails.

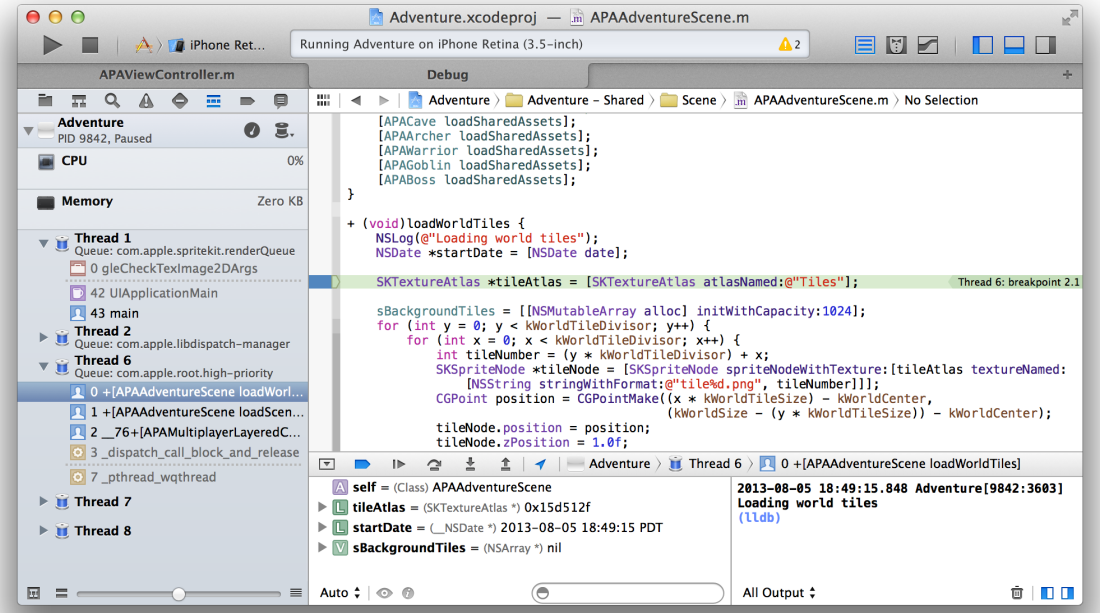
In the screenshot below, behaviors are customized for whenever the code pauses. Here are some examples of customized behaviors:

- Play an alert sound at every pause.
- Create a tab named Debug in the workspace window for displaying the debug navigator.
- Show both the variables view and the console view in the Debug tab.

- Hide the utilities area in the Debug tab.



As a result, when the code in the project hits a breakpoint, Xcode creates a Debug tab in the workspace window with the specified content.



You can design custom behaviors that are triggered by menu items or their keyboard equivalents. Choose Xcode > Preferences, select the Behaviors preferences pane, and click the Add button (+) at the bottom of the pane. Type the name of the new behavior, and press Return. Select checkboxes to specify what should happen when you invoke this behavior. For example, create a Unit Testing behavior that saves a snapshot of your project and runs your unit tests. After you've created a behavior, it appears in the Xcode > Behaviors menu.

To assign a keyboard equivalent to a custom behavior, choose Xcode > Preferences and click Key Bindings. In the Key Bindings preferences pane, select the Customized tab to find the custom behavior you want. In the text field, enter the keys you want to use for the key binding in the text field, and click outside the text field to complete the operation.

For more detail on types of breakpoints and breakpoint actions, see *Breakpoint Navigator Help*.

Test Your App

Create tests that automatically exercise the features and test the performance of your application. Monitor the results of the tests and fix any issues from the test navigator.

You can use the Xcode service, available in OS X Server, to automate the execution of tests. From Xcode on your development Mac, you create *bots* that run on a separate server. In addition to running unit tests, bots automatically perform static analysis on your code, build your app, and archive it for distribution to testers or the App Store. Bots help you ensure that your product is always in a releasable state—and when there's a failure, the service notifies you or the person whose code change caused the failure.

Create and Run Tests

Xcode supports two main types of testing. *Functional tests* focus on code functionality. *Performance tests* focus on measuring execution time. Both kinds of tests are functions that you write. Each function sets up an environment for the test, executes the targeted parts of the app, and tears down the test environment.

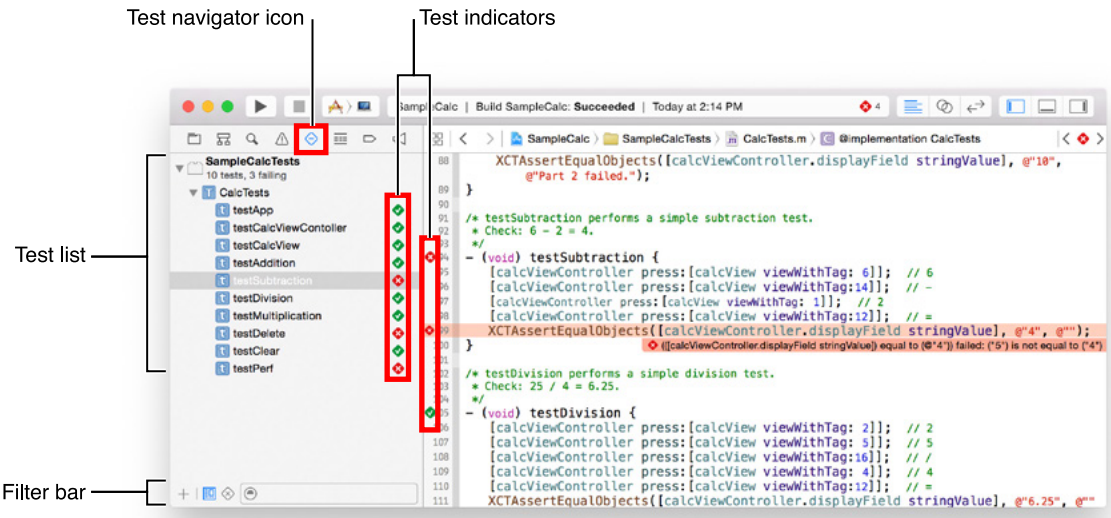
The most common type of functional testing is unit testing. A *unit* of code is the smallest testable component of your project—for example, a method in a class or a set of methods that accomplish an essential purpose. Unit tests are often used to detect regressions introduced by code changes to a project. Some developers write unit tests first and then implement methods that pass the tests.

Performance tests measure the time it takes your app to complete a task on different types of devices. Xcode tracks times for different configurations and you choose baselines from measured values.

A *test case* exercises a unit of code in a specific way or measures a specific part of your app's performance; if the result of the test is different from the expected result, the test case fails. A *test suite* is made up of a set of test cases.

When you create a project or a target, Xcode includes a unit test target in the scheme that builds the app. The implementation file for the target includes stubs for the `setUp`, `tearDown`, and `testExample` methods. Complete these stub implementations and add other code as necessary to perform unit tests on your app.

Run all tests by choosing Product > Test. Click the Test Navigator icon to view the status and results of the tests. You can add a test target to a project (or add a class to a test) by clicking the Add button (+) in the bottom left corner of the test navigator. To view the source code for a particular test, select it from the test list. The file opens in the source code editor.



To run a test suite, click the arrow to the right of the name. To run a subset of test methods, select them in the test navigator and choose Product > Perform Action > Run Test Methods. To run an individual test method, click the arrow to the right of the method name. Choose Product > Test to run all tests in the active scheme.

When a test succeeds, a green diamond with a checkmark denoting success appears to the right of the test name. When a test fails, a red diamond with an X denoting failure appears to the right of the test name and the issue is displayed in the issue navigator. To see the issue, click the Issue Navigator button in the navigator bar.

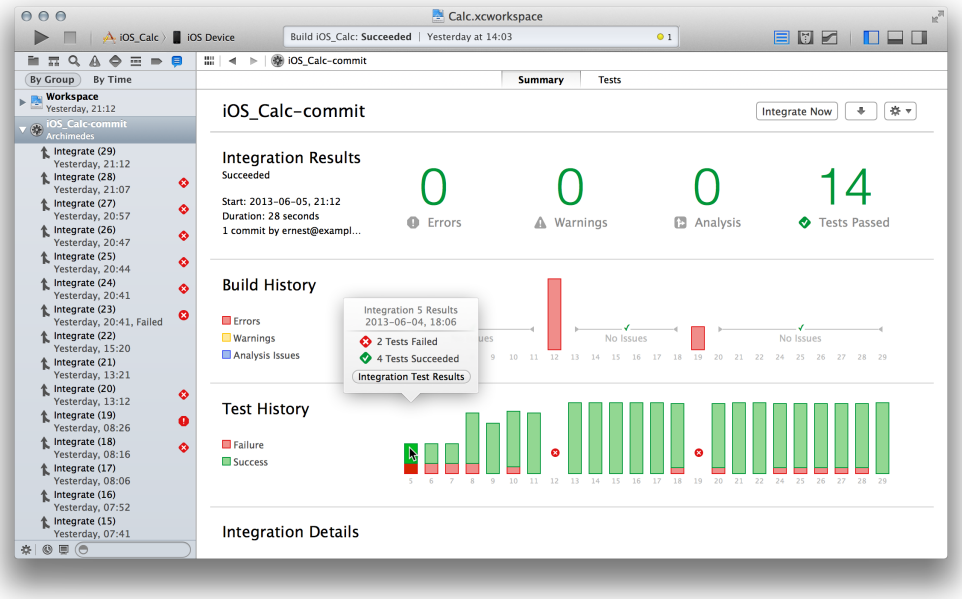
To view only the failed tests, click the Failed Test button ( ) at the bottom of the test navigator. Select a failed method to examine it in the source code editor. After addressing the reason for the failure, click the failed test indicator (a red diamond with an X) to rerun the test.

Show related test methods in an assistant editor by choosing either the Test Classes or Test Callers categories from the Assistant pop-up menu.

For more detail on writing, running, and viewing tests, see *Testing with Xcode*.

Automate Unit Testing as Part of a Continuous Integration Workflow

Xcode supports a continuous integration workflow through the Xcode service. The *Xcode service*, available in OS X Server, automates the integration process of building, running unit tests, performing static analysis, and archiving your product. The service reports build errors and warnings, static analyzer problems, and unit test failures. All tests, analysis, and archiving are performed on the server.



For information on setting up and using the service, see *Xcode Server and Continuous Integration Guide*.

Save and Revert Changes

Xcode automatically saves changes to source, project, and workspace files as you work. This feature requires no configuration, because Xcode continuously tracks your changes and saves them in memory. Xcode then writes these changes to disk whenever you:

- Build and run your app
- Commit files to a source code repository
- Close the project
- Quit Xcode
- Create a snapshot

You can also manually save changes to disk by choosing File > Save.

Xcode lets you revert files and entire projects to a previous state; you can also discard those changes. You use source control management to keep track of changes at a fine-grained level.

Revert to the Last Saved Version of a File

To discard all changes you've made to a file since it was last saved to disk, choose File > Revert Document. The Revert Document command operates only on the file that has the editing focus. Give editing focus to a file either by clicking its editor pane or by selecting it in the project navigator. For example, you experiment with a new user interface layout and then decide to revert to the previous layout. Or you need to undo some code changes because they introduced a problem.

The Revert Document command always returns the contents of the file to the last saved version on disk. If you prefer to back out changes one change at a time, use the Undo command in the Edit menu.

Undo File Changes Incrementally

To back out changes to a file incrementally, choose Edit > Undo *change*. The Undo command is contextualized by your last operation. For example, the command appears as Undo Typing if you make an edit to an implementation file; the command changes to Undo Add Button if you add a button object to a storyboard.

With the Undo command, you can back out every change to a file since the start of your editing session. An editing session begins when you open a project and ends when you close the project. Xcode lets you undo *all* the edits in that session, even those already saved to disk. (Note, however, that the Revert Document command clears the Undo history, and you cannot undo a revert operation.)

After you've chosen the Undo command, you can choose Edit > Redo to reverse the last undo operation.

Use Snapshots to Restore Projectwide Changes

Snapshots provide an easy way to back up the current version of your project or workspace. If something goes wrong because of a code change you make you can restore your entire workspace, including all project files, to a previous state.

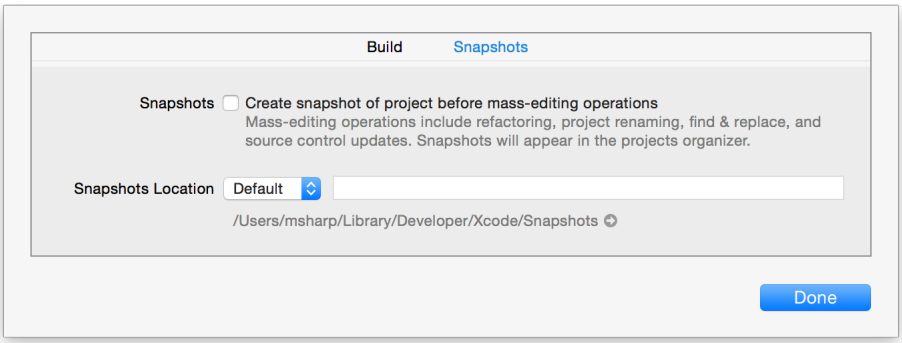
A snapshot is an archive that includes all document files in the project or workspace and all project and workspace settings. Snapshots support reverting from three changes that are not supported by the Revert Document and Undo commands:

- Xcode operations that involve changes to many document files and potentially to project settings. These operations include refactoring code, performing project validation, and adding Automatic Reference Counting to an existing project.
- Adjustments to the workspace and project settings.
- Global search and replace operations.

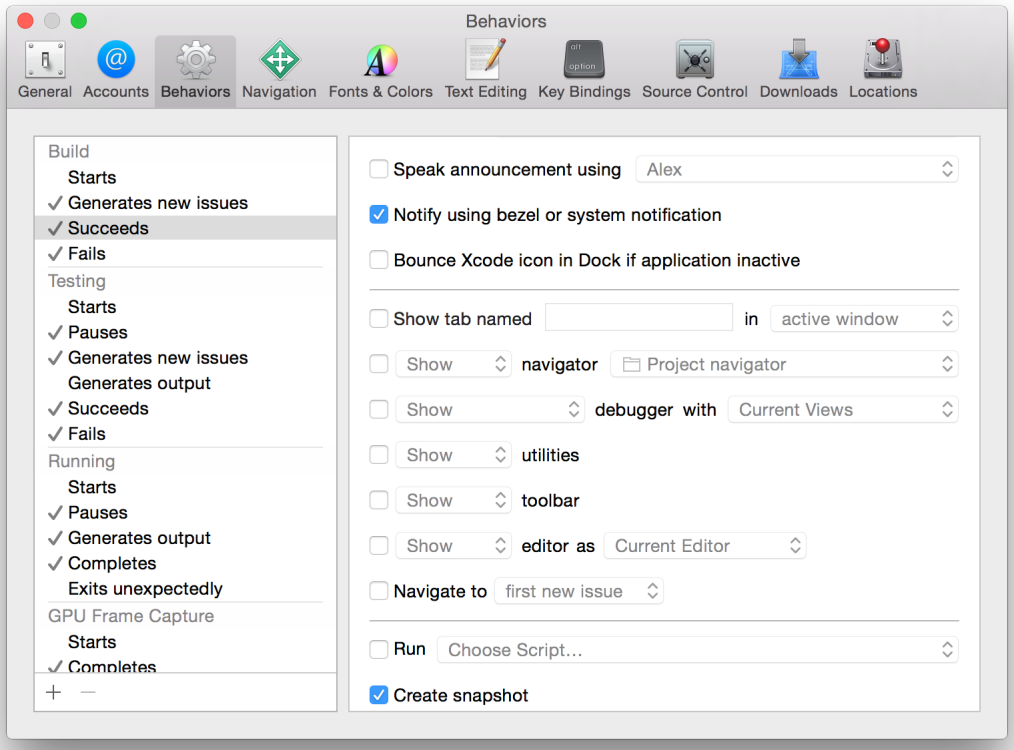
You create a snapshot manually by choosing File > Create Snapshot.

Xcode can automatically create snapshots. The first time you perform a mass editing operation for a project or workspace, Xcode prompts you to turn on automatic snapshots for mass editing operations. Also, you can configure automatic creation of mass editing snapshots in the project or workspace preferences. Choose File

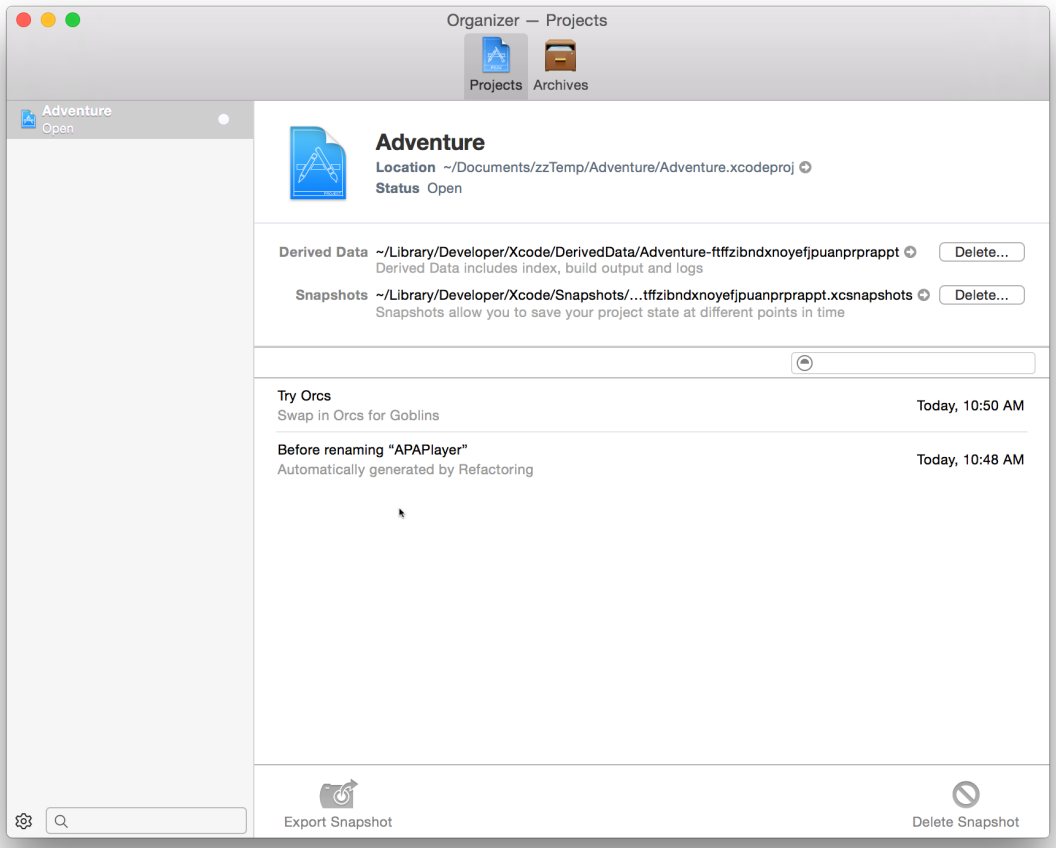
> Project Settings or File > Workspace Settings. In the settings window that appears, select the Snapshots tab and set the "Create snapshot of project before mass-editing operations" checkbox to the desired state. The screenshot shows the settings window with automatic snapshots turned off.



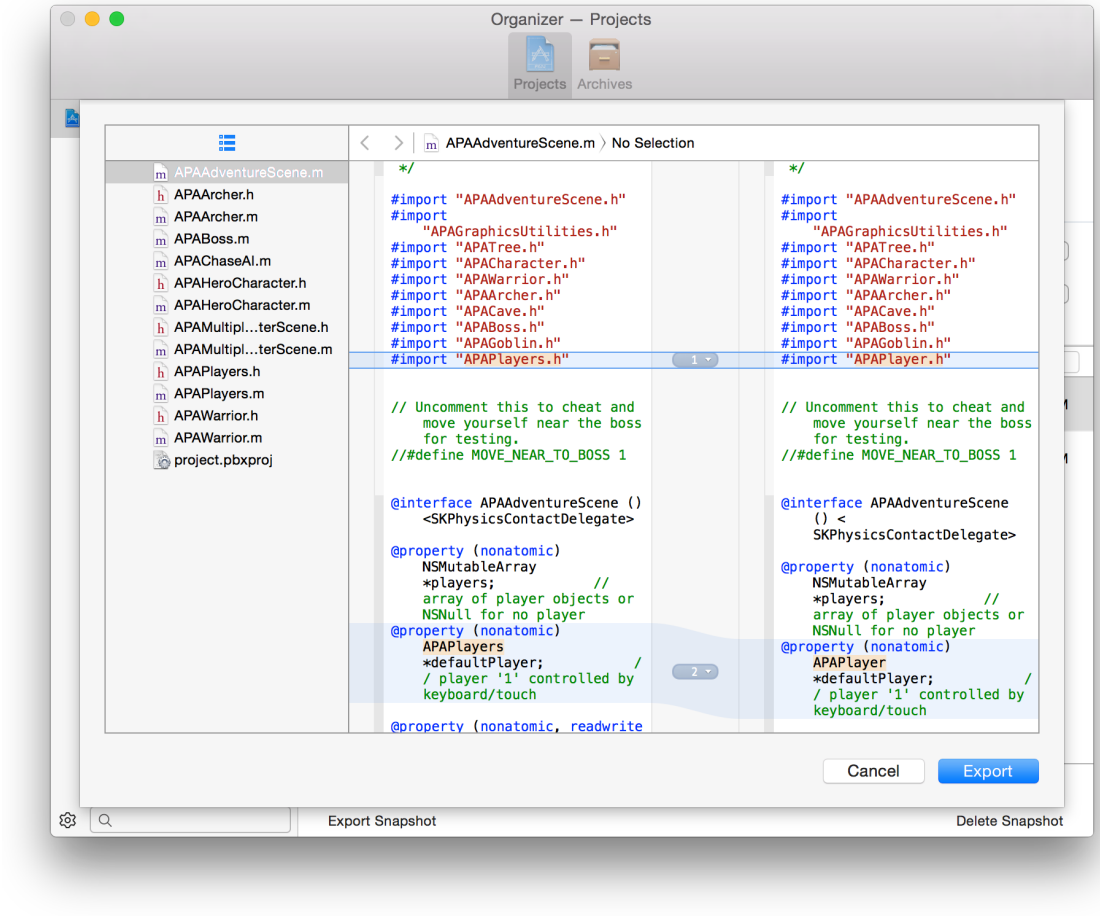
Set Xcode to automatically create snapshots in other circumstances by choosing Xcode > Preferences, selecting the Behaviors pane, and selecting the Create Snapshot option for any of the behaviors. For example, the screenshot shows adding the creation of a snapshot when a build succeeds. The Create snapshot checkbox is at the bottom of the configurable behaviors on the left side of the window.



To see the snapshots for a project or workspace, choose Window > Organizer, select Projects to open the Projects organizer, and click the project.



Recover an earlier state of a project or workspace by exporting a snapshot from the Projects organizer. Choose Windows > Organizer, select a project or workspace, select a snapshot, and click the Export Snapshot button at the bottom of the window. Xcode displays a preview dialog in which you can review the differences between the snapshot version of a file on the left and the current version on the right.



To export a snapshot, click Export, select a destination folder for your snapshot, and click Export again.

You can also restore a snapshot on top of the current project by choosing File > Restore Snapshot from the project's workspace window. Xcode displays a preview dialog in which you can review the differences between the current version of the project and the snapshot version. When you click Restore, Xcode replaces the current version of the project with the version in the snapshot. Xcode takes a snapshot of the current version before replacing it.

You can restore a deleted project from a snapshot because Xcode keeps track of all your projects, even deleted ones, and displays them in the Projects organizer.

Store and Track Changes with Source Control

Use commands in the Source Control menu to manage your project files with a source code repository. A repository saves multiple versions of each file onto disk, storing historical metadata about each version of each file. Source control allows you to keep track of file changes at a finer level of detail than snapshots allow. Source control also helps you coordinate efforts if you work with a team of programmers.

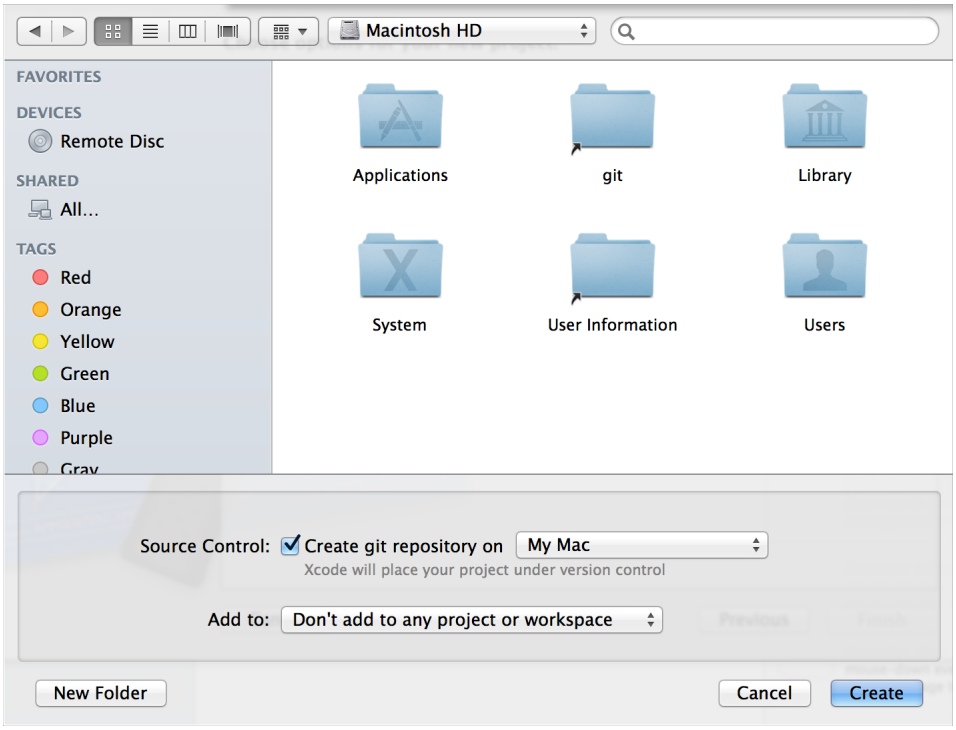
A source control system helps you reconstruct past versions of a project. You can commit a file to your repository each time you make a major change. If you introduce bugs, you can use the Xcode version editor to compare the new version of the file with a past version that worked correctly, to locate the source of the trouble.

When multiple people work on a project, source control helps prevent conflicts and helps resolve conflicts should they arise. By keeping a central repository that holds the master copy of the software, the source control system allows each programmer to work on a local copy with no risk of corrupting the master. With a file checkout system, you can ensure that two people don't work on the same code at the same time. If two people do change the same code, the system helps you merge the two versions.

You can also branch from a stable version of your project, add new features and make other changes to the branch, and then merge and reconcile those changes back into the stable version of your project.

Xcode supports two popular source control systems: Git and Subversion. Subversion (often abbreviated svn) is always server based. The server is typically on a remote computer (although it is possible to install the server locally). Git can be used purely as a local repository, or you can install a Git server on a remote computer to share a repository among team members.

If you are working alone, it's easiest to use Git, because you won't need to set up a server. When you create a project, Xcode automatically sets up a Git repository for you.

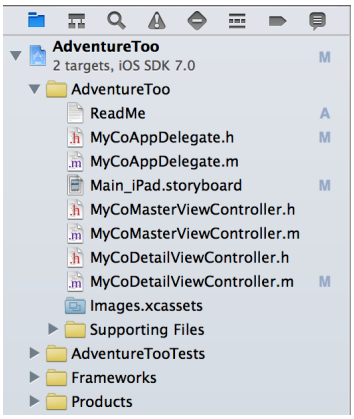


In addition to performing continuous integrations, the Xcode service, available with OS X Server, hosts Git repositories.

If your repository is on a server, choose Source Control > Check Out to create a local working copy of the project on your computer. If you use a local Git repository, you don't check out a working copy, because your local repository is your master copy.

When you are satisfied with changes you've made to a file, choose Source Control > Commit to ensure that those changes are preserved in the repository. You are required to provide a comment explaining the nature of your commit. If your Git repository is on a server, the commit operation adds your changes to your local repository. Perform a push operation to add your committed changes to the Git repository on the server. For example, when you choose Source Control > Commit on your development Mac, select the "Push to remote" option, specify the remote repository in the pop-up menu, and click Commit Files.

You can see the source control status of your files in the project navigator. The status is shown as a badge next to the filename.

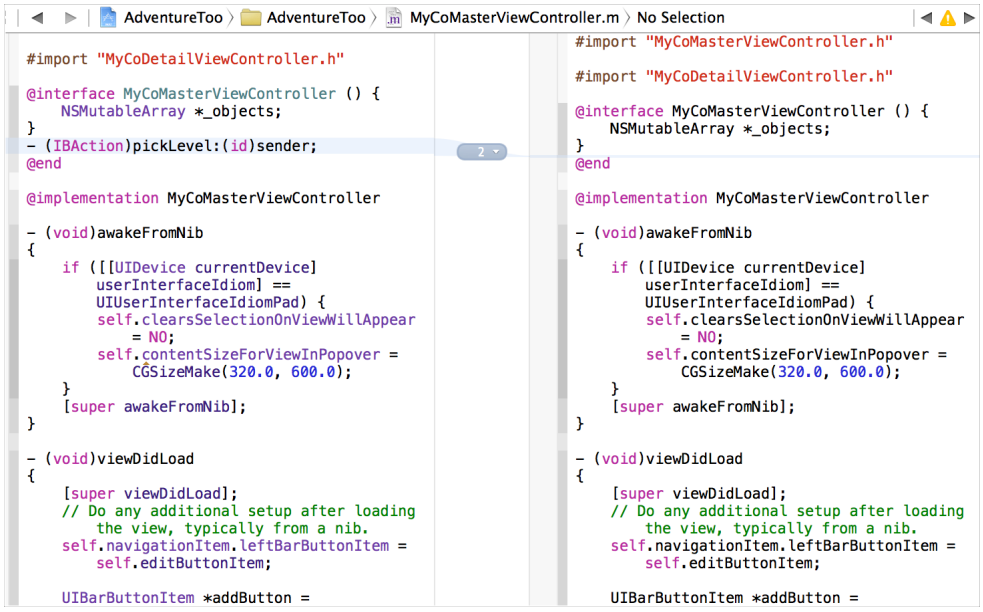


Badge	SCM status
M	Locally modified
U	Updated in repository
A	Locally added
D	Locally deleted
I	Ignored
R	Replaced in the repository
–	The contents of the folder have mixed status; display the contents to see individual status
?	Not under source control

For help on connecting to and working with source code repositories, see *Source Control Management Help*

Compare File Versions to Revert Lines of Code

Choose View > Version Editor > Show Comparison View to compare versions of files saved in a repository. Use the jump bars to choose file versions based on their position within a repository. Each jump bar controls the selection for the content pane above it. To display a version, browse through the hierarchy to find it, then click to choose it. Shaded areas indicate changes between versions.



Use the version timeline to choose file versions based on their chronological order. Click the Timeline Viewer icon ( ) at the bottom of the center column to display the timeline between the two editing panes. Move the pointer up or down through the timeline to browse the available versions. When you find the version you want, click the left or right disclosure button to display that version in the corresponding editor pane.

You can edit the current working copy of the file in the version editor. If you want to revert changes between versions, you can copy code from an older version and paste it into the current version.

Create a Branch to Isolate Risky Changes

After you've worked on a project for a while, you are likely to have a body of reliable, stable code. You can choose Source Control > *Working Copy* > New Branch to create a copy of that code. Then you can work on new features and other changes without destabilizing your existing code base. When you are satisfied with your changes, you can merge them back into the body of stable code. Use Source Control > *Working Copy* > Merge from Branch and Source Control > *Working Copy* > Merge into Branch to combine and reconcile differences between versions of your project.

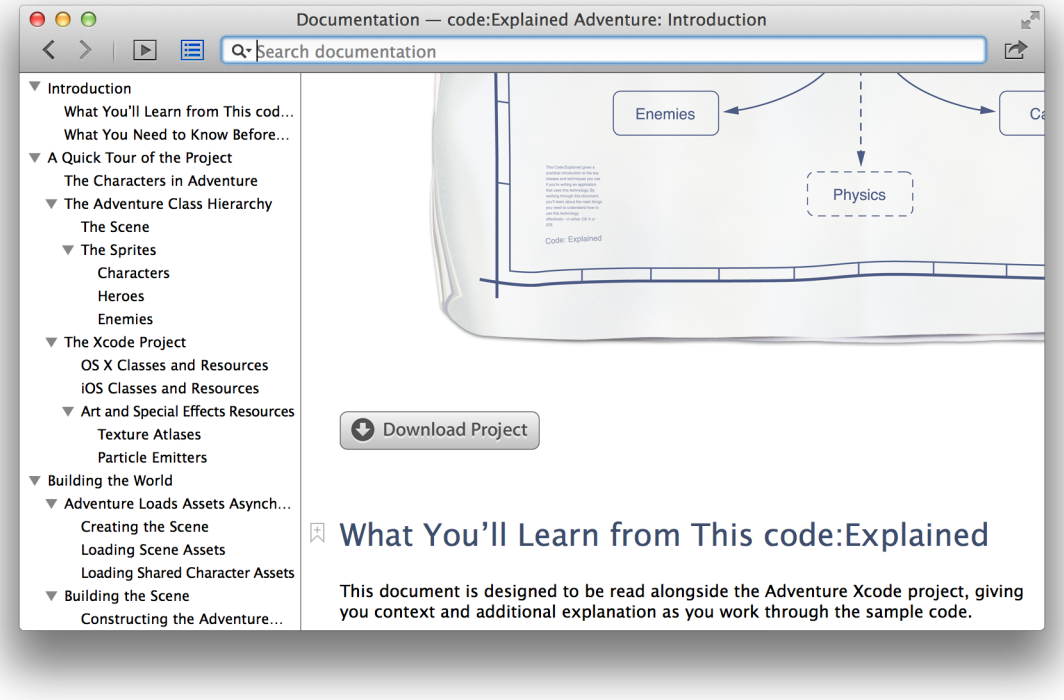
Learn More About Xcode

This guide introduces you to features and capabilities of Xcode. To help you become an expert Xcode user, Apple provides additional documentation within Xcode. You'll find identical Xcode documentation online in the iOS Developer Library and Mac Developer Library.

Get a Hands-On Introduction

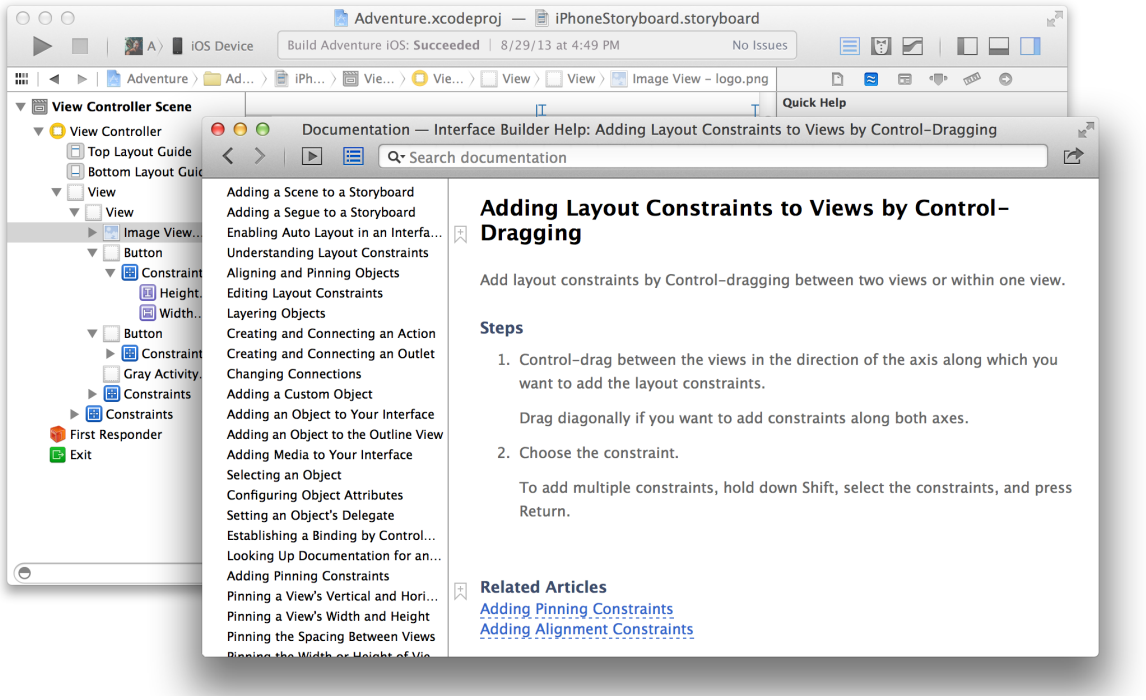
If you're new to iOS or Mac programming, work through *Start Developing iOS Apps Today* or *Start Developing Mac Apps Today*. Each document provides a starting point for app development. In each, you create a simple app, and you learn the basics of programming with Objective-C using either the Cocoa Touch or Cocoa framework.

For a guided tour through the development of a game project that can be built for both iOS and Mac, look at *code:Explained Adventure*, which includes a link to download the complete Xcode project.



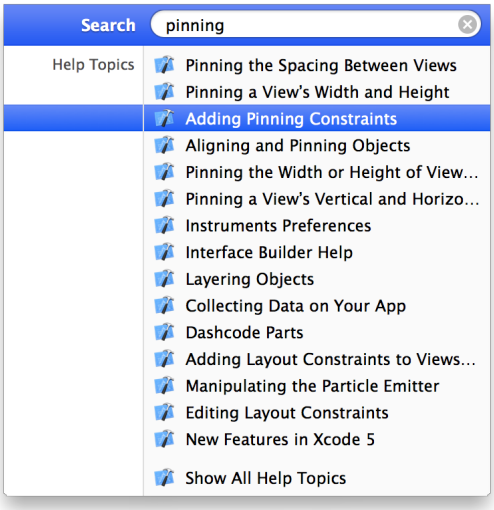
Find Step-by-Step Instructions

As you saw earlier, step-by-step instructions for performing common tasks are available directly in Xcode. Control-click areas of the Xcode user interface to see a short list of the most common operations. Choose Show All Help Topics to see a larger list. Select an operation from the list, and a help article appears in the Xcode documentation viewer window.

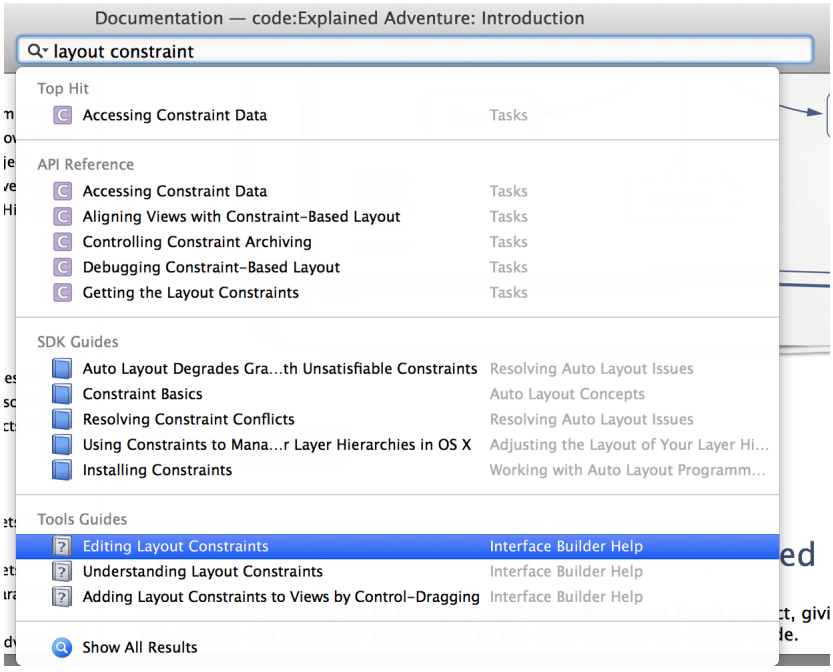


Many articles include links to additional articles describing related tasks. Many articles also include links to lengthier discussions in user guides, which offer additional context and details about performing tasks in Xcode.

The Help menu also offers quick access to help articles. Type a search term or phrase, and a list of relevant help articles and related documents appears directly in the menu.



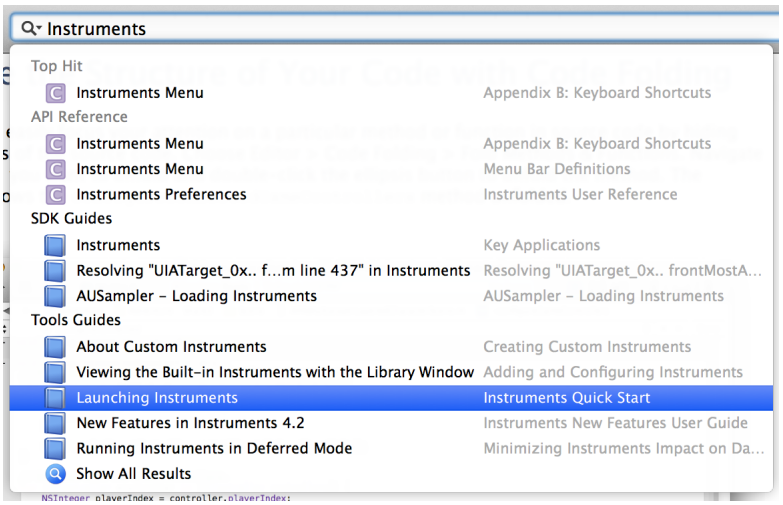
You can also use the search bar in the Xcode documentation viewer to locate help for a task. In Xcode, choose Help > Documentation and API Reference to display the documentation viewer. As you type into the search field, a menu appears of top search results. Choose a document directly from this menu, or click Show All Results at the bottom it to see a comprehensive list of search results.



Xcode obtains these results by searching through all of the documentation relevant to your project's SDK. You'll find results for API symbols listed under API Reference, programming guide results under SDK Guides, and Xcode documentation results under Tools Guides.

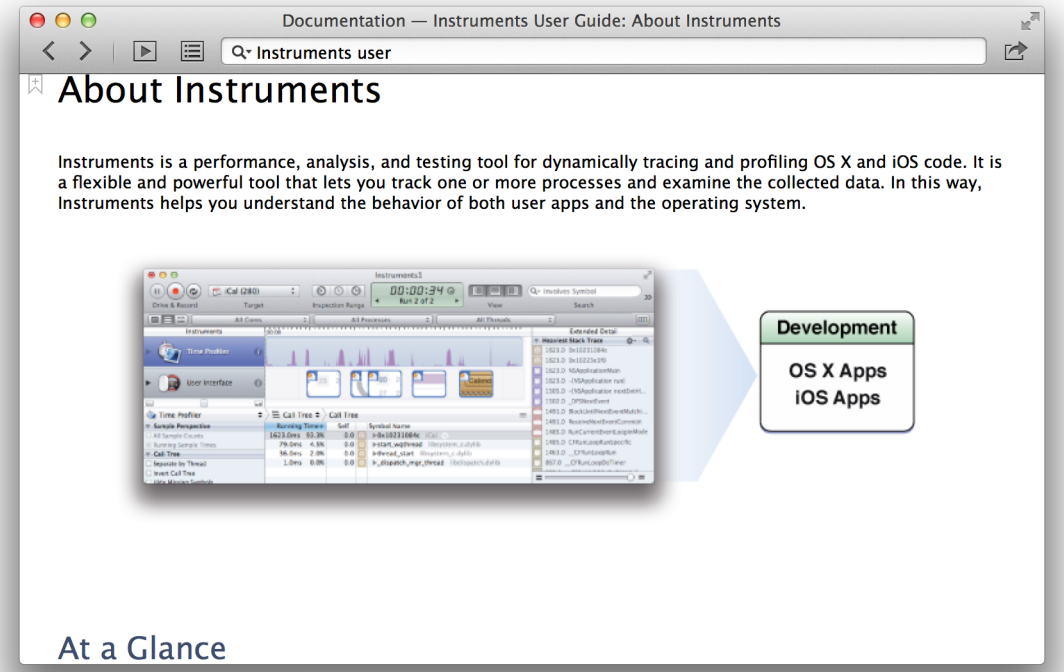
Learn from Detailed User Guides

Apple provides detailed teaching guides for developer tool topics, including iOS Simulator, Instruments, app distribution, and continuous integration. To locate user guides for features in Xcode, use the search bar in the Xcode documentation viewer and look for results under Tools Guides.

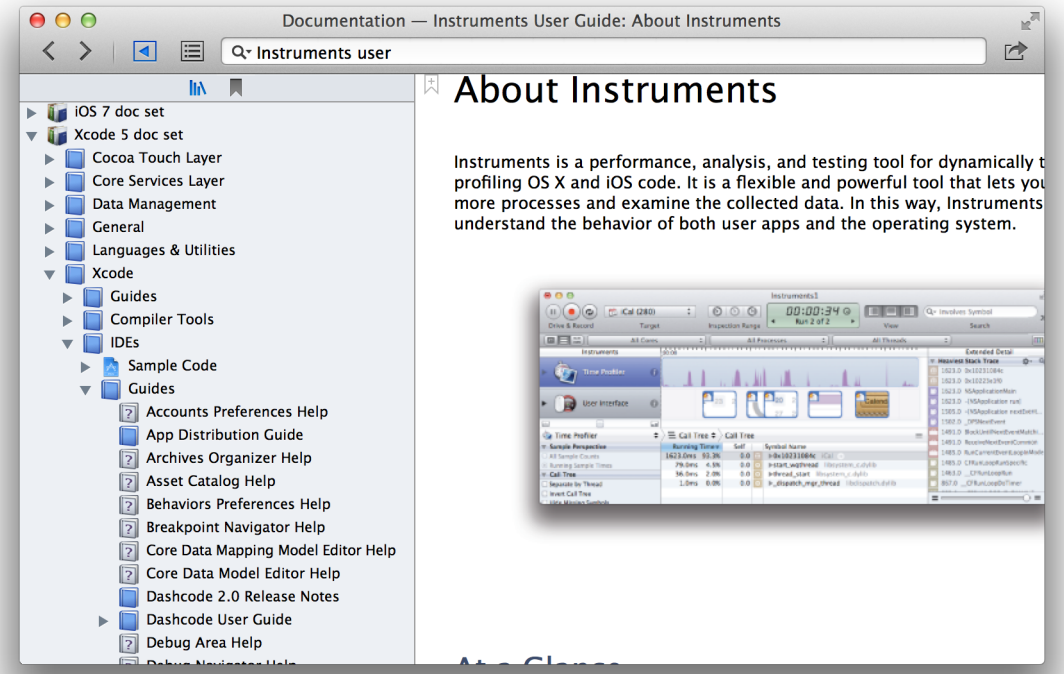


The *Instruments User Guide* , for example, explains how to use the Instruments app to examine program behavior. Like many developer tool guides, it begins with a quick start tutorial.

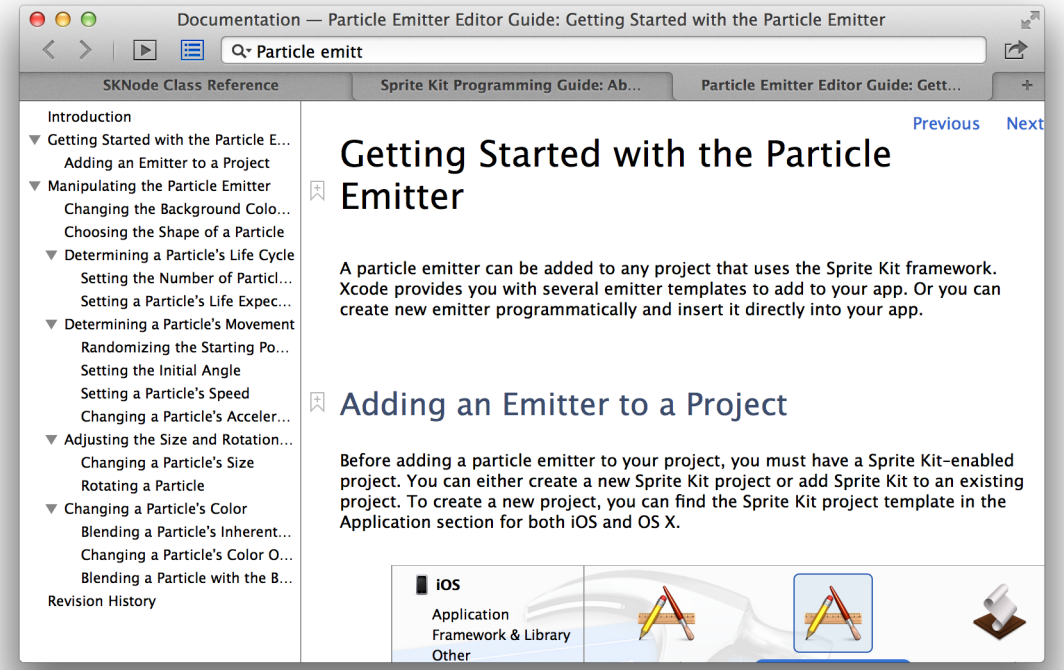
To hide or show a list of the chapter and section titles for the document, click the “Table of Contents” button ( ) to the left of the search bar .



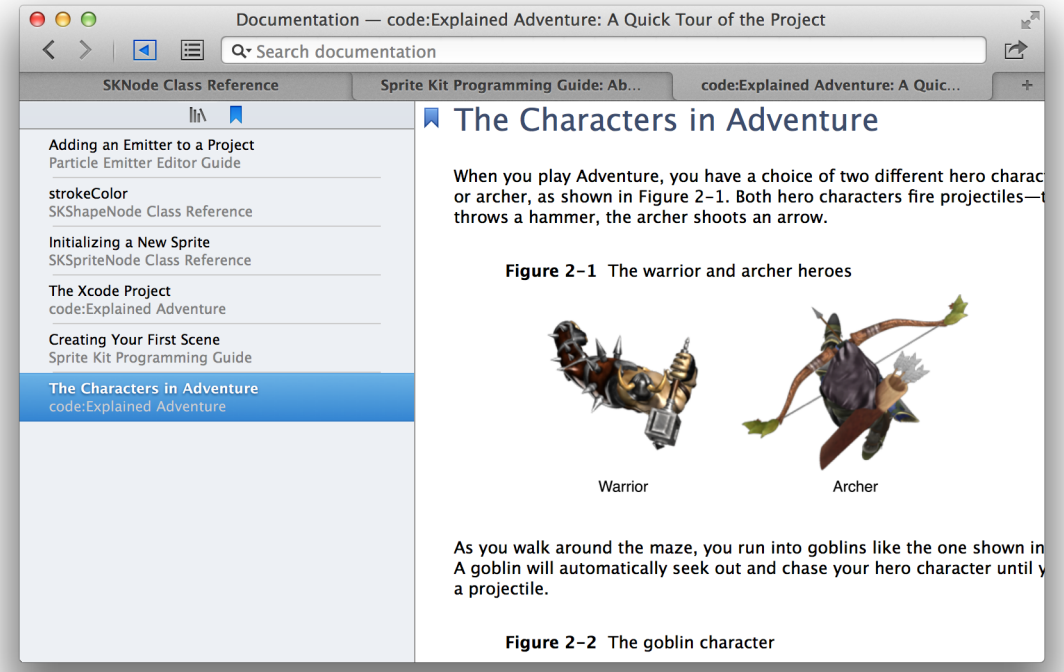
Click the Documentation Navigator button ( ) to the left of the “Table of Contents” button to display a navigation sidebar. To browse the list of Apple developer documentation installed in Xcode, click the Documentation Library button ( ).



Use the tab bar in the doc viewer to keep multiple related documents open at once. To create a tab, choose File > New > Tab (or click the Add button (+) in the tab bar).



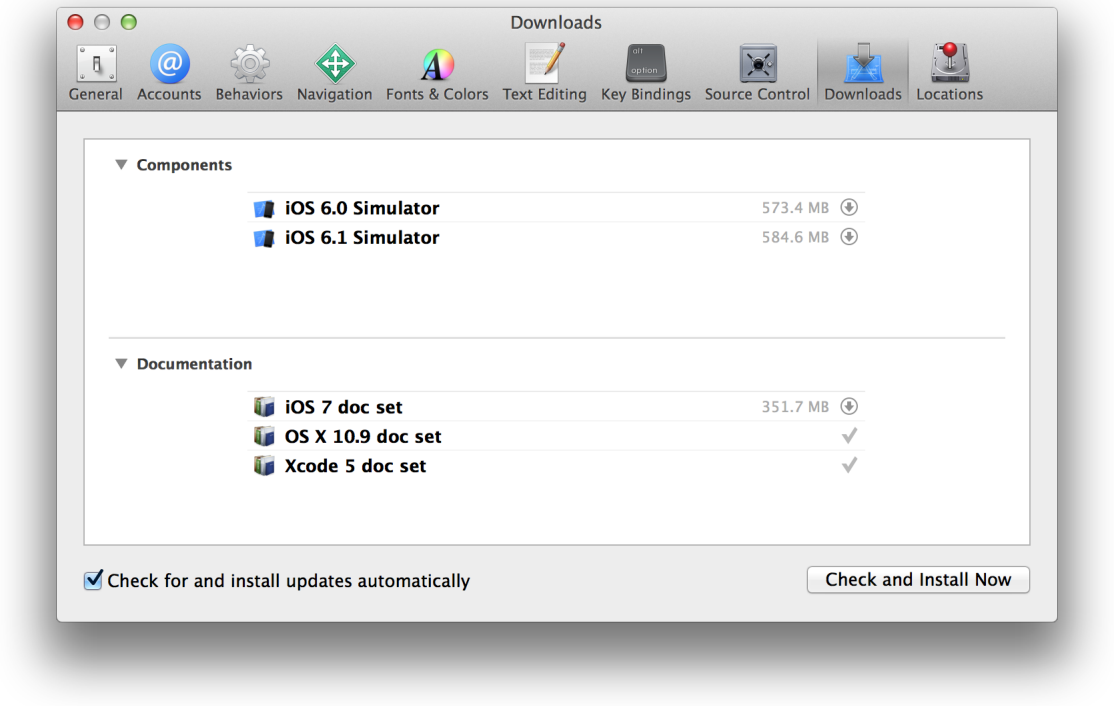
For quicker access to documents that you'll return to, click the Bookmark icon ( ) next to a chapter or section title. To display all of your bookmarks, click the Documentation Navigator button ( ) to display the navigation sidebar, then click the Bookmark button ( ) at the top of the sidebar.



Stay Up to Date

Apple continually produces new and updated help articles, user guides, programming guides, and API reference. As updated documentation becomes available, it downloads to Xcode in the background. Be sure to keep your documentation up to date by leaving the default download behavior intact or by manually checking for documentation updates on a regular basis.

Documentation is installed in the form of documentation sets, also called *doc sets*. Apple doc sets associated with your projects' SDKs are installed with Xcode, and access to updates for them is controlled by subscription. For your convenience, Xcode can keep these subscriptions up to date. This feature is controlled by the option "Check for and install updates automatically," which you can select in the Downloads pane (available by choosing Xcode > Preferences).



To check for updates manually, click "Check and Install Now." If no new updates are available, Xcode displays a message to that effect. When an update for a doc set is available but not yet installed on your system, Xcode displays a Download button on the subscription line for that doc set. Click the Download button ( ) to download and install the updated doc set on your system.

Document Revision History

This table describes the changes to *Xcode Overview*.

Date	Notes
2014-10-20	Updates for OS X plus other minor updates.
2014-10-16	Updates for OS X plus other minor updates.
2014-09-17	Updated for Xcode 6 and iOS 8, including the new look of the Xcode UI, updates to Build a User Interface for Size Classes and recent Auto Layout additions, updates to Debug Your App for new features, and other updates and changes throughout.
2014-03-10	Added minor clarifications and fixed typos.
2014-02-11	Revised the descriptions of actions and outlets in the chapter "Build a User Interface."
2013-10-22	Updated the information about documentation viewing to reflect product improvements.
2013-09-18	Changed the title from "Xcode User Guide" and updated for Xcode 5.
2013-04-23	Made minor content fixes.
2013-01-28	Incorporated content from Tools Workflow Guide for iOS.
2012-09-19	Made available in PDF.
2012-06-11	Added chapter about saving and reverting changes to files.
2011-10-12	Added features new in Xcode 4.2.
2011-08-10	Added features new in Xcode 4.1.

Date	Notes
2011-07-07	Added features new in Xcode 4.1.
2011-05-07	Made editorial and format changes throughout.
2011-03-02	New document that explains how to use Xcode 4 to develop software for iOS and OS X.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010