Configuring Your Xcode Project for Distribution

You can edit your project settings anytime, but some settings are necessary during development. Others are recommended when you distribute your app for beta testing and required when you submit your app to store.

During development, your app must be provisioned and code signed to use certain app services and run on a device. If you assign a bundle ID and team to your project, Xcode can create the necessary certificates, identifiers, and profiles for you in your developer account. You can enter this information now using the project editor or as needed when you add app services and launch your app.

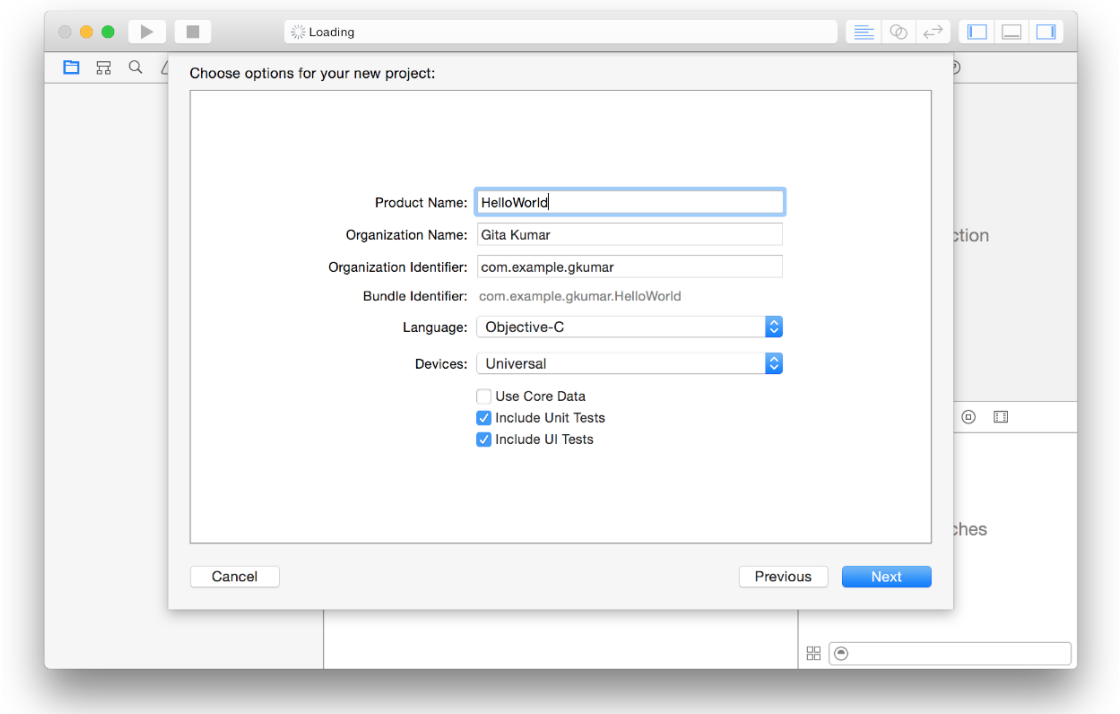
Before you distribute your app for testing or to the store, provide all the required information about your app. For example, set app icons to pass iTunes Connect validation tests.

Before you upload your app to iTunes Connect, verify your build settings and set the copyright key for Mac apps.

Setting Properties When Creating Your Xcode Project

An assistant guides you through the process of creating an Xcode project. First, you select a template for your project. Starting with the right template helps speed the development process. The assistant also prompts you to enter information about your app that's used to determine your app's capabilities and distribute it to customers. If you don't have this information when you create the project, you can set these properties later. If you are a member of the Apple Developer Program, some of the data in the Xcode project is similar to the data you enter in iTunes Connect, but only the bundle ID needs to match the bundle ID you enter in iTunes Connect before you can upload your app to iTunes Connect.

For iOS apps, a dialog similar to this appears when you create an Xcode project from a template:



The *product name* is the name of your app as it will appear to customers in the store and should be similar to the app name you enter later in iTunes Connect. Most importantly, a customer should instantly associate the app name and icon in the store with the product name and app icon that installs on their devices. The product name is also the name that will appear in Springboard when the app is installed. The product name can't be longer than 255 bytes and can be no fewer than 2 characters. (Read Best Practices in *iTunes Connect Developer Guide* for guidelines on choosing an app name.)

The *organization name* is an attribute of the Xcode project and is used in boilerplate text throughout your project folder. For example, the organization name is used in the source and header file copyright strings. The organization name in your Xcode project isn't the same as the organization name that you enter later in iTunes Connect.

The product name and *organization identifier* you enter are concatenated to create the default bundle ID using reverse domain name service (reverse DNS) notation. The *bundle ID* needs to be unique to your app, so it's important to set the organization identifier to a unique string as well.

For iOS apps, you can choose the types of devices you support from the Devices pop-up menu. For Mac apps, you can choose the Mac App Store categories from a pop-up menu.

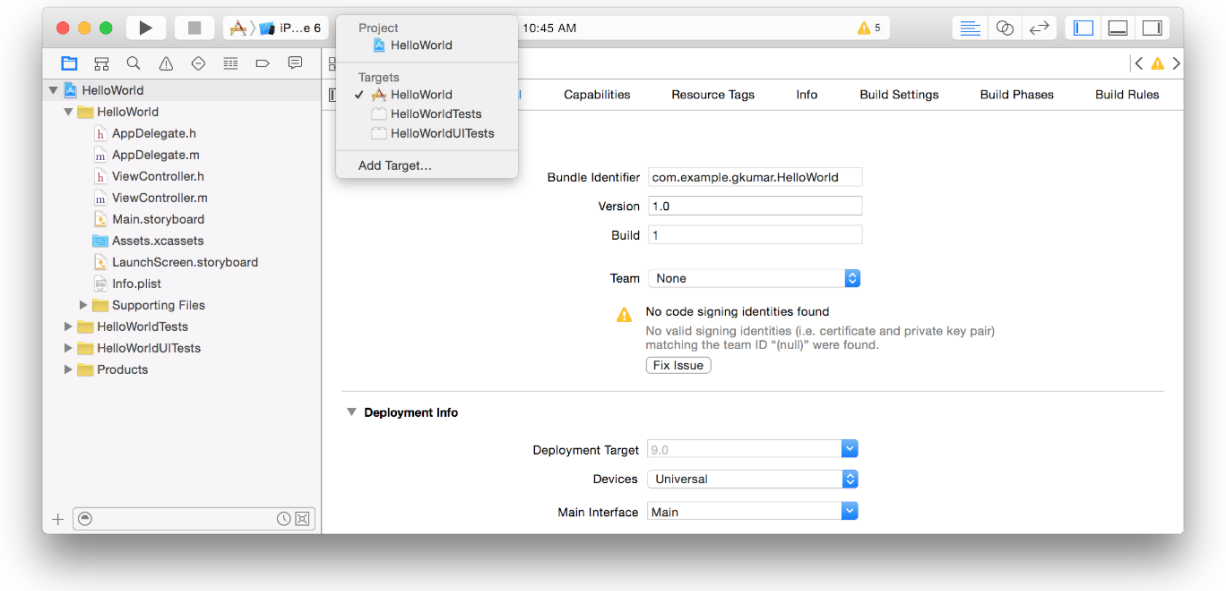
The other values used by the Xcode template are sufficient for building and running your app locally, but later you'll need to finalize properties, such as the bundle ID. Also, the assistant doesn't set all required properties for the store. You complete the basic store configuration before you submit. Ideally, you'll complete this configuration before you distribute your app for testing too.

After your app is released, you can't change some of this metadata, so it's important to choose your settings carefully. To learn which app states cause some properties to be locked in iTunes Connect, refer to iTunes Connect App Properties in *iTunes Connect Developer Guide*.

Before You Begin Configuring Your Project

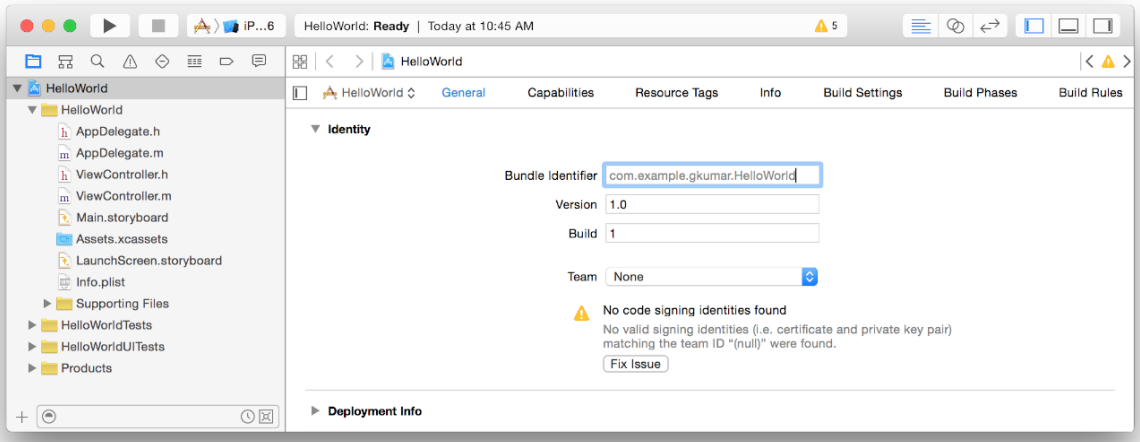
All of the options discussed in this chapter are located in the General pane in the project editor for your target. To open the project navigator, choose View > Navigators > Show Project Navigator. Choose the target from the Project/Targets pop-up menu or in the Targets section of the second sidebar if it appears. Click General to view settings discussed in this chapter.

The screenshot below shows the General pane for an iOS app.



Configuring Identity and Team Settings

For Xcode to create the team provisioning profile, the app's bundle ID needs to be unique and the project assigned to a team. Later, you provide other information that identifies this version of your app. The identity settings appear in the Identity section of the target's General pane. For iOS apps, the Identity section appears as shown here:

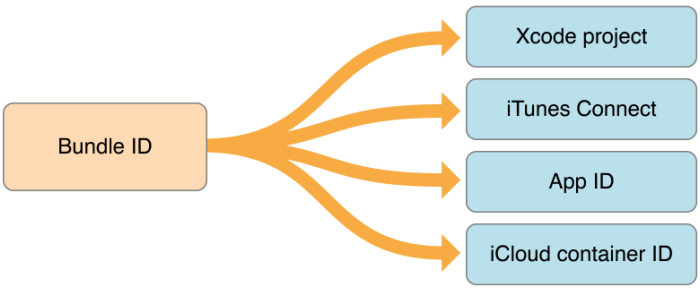


About Bundle IDs

A *bundle ID* precisely identifies a single app. A bundle ID is used during the development process to provision devices and by the operating system when the app is distributed to customers. For example, Game Center and In-App Purchase use a bundle ID to identify your app when using these app services. The preferences system uses this string to identify the app for which a given preference applies. Similarly, Launch Services uses the bundle ID to locate an app capable of opening a particular file, using the first app it finds with the given identifier. The bundle ID is also used to validate an app's signature.

The bundle ID string must be a uniform type identifier (UTI) that contains only alphanumeric characters (A-Z,a-z,0-9), hyphen (-), and period (.). The string should be in reverse-DNS format. For example, if your organization's domain is `Acme.com` and you create an app named Hello, you could assign the string `com.Acme.Hello` as your app's bundle ID.

During the development process, you use an app's bundle ID in many different places to identify the app.



Specifically, the bundle ID is located and used as follows:

- In the Xcode project, the bundle ID is stored in the information property list file (`Info.plist`). This file is later copied into your app's bundle when you build the project.
- In iTunes Connect, you enter the bundle ID to identify your app. After your first build is uploaded to iTunes Connect, you can't change your bundle ID or delete the associated explicit App ID.
- In your developer account, Xcode creates an App ID that matches the app's bundle ID. If the App ID is an explicit App ID, it exactly matches the bundle ID. However, unlike domain names, bundle IDs are case sensitive. If the App ID is lowercase, your bundle ID needs to be lowercase, too.
- In iCloud, the container IDs you specify in your Xcode project are based on the bundle IDs of one or more apps.

> **Note:** A Mac app can't share the same bundle ID with other types of apps, such as iOS and tvOS apps.

Setting the Bundle ID

The default bundle ID in your Xcode project is a string formatted as a reverse-domain—for example, `com.MyCompany.MyProductName`. To create the default bundle ID, Xcode concatenates the organization

identifier with the product name you entered when creating the project from a template, as described in Setting Properties When Creating Your Xcode Project. (Xcode replaces spaces in the product name to create the default bundle ID.) It may be sufficient to replace the organization identifier prefix in the bundle ID, or you can replace the entire bundle ID. For example, change the organization identifier prefix to match your organization domain name, or replace the entire bundle ID to match an explicit App ID.

For tvOS apps that share the iTunes Connect app record of an iOS app (described in Creating a Universal Purchase (iOS, tvOS)), the tvOS app must have the same bundle ID as the iOS app.

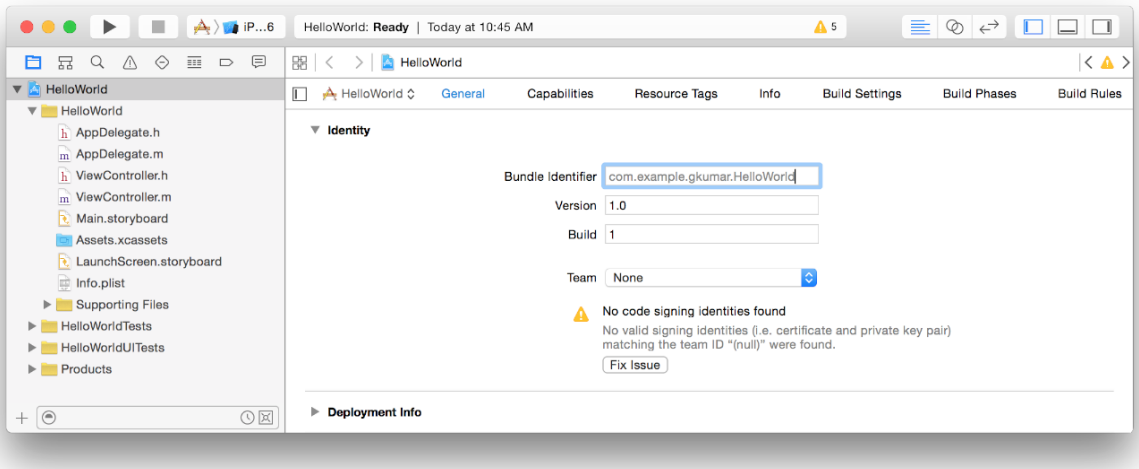
For watchOS apps, the embedded WatchKit app and WatchKit extension targets must have the same bundle ID prefix as the iOS target. The WatchKit app target must have the format `[Bundle ID].watchkitapp` and the WatchKit extension target must have the format `[Bundle ID].watchkitextension`. If you change the iOS app's bundle ID, you must change the WatchKit app and WatchKit extension target's bundle ID to match.

For Mac apps, ensure that every bundle ID is unique within your app bundle. For example, if your app bundle includes a helper app, make sure that its bundle ID is different from your app's bundle ID.

Follow these steps to change the bundle ID prefix in the General pane in the project editor.

To set the bundle ID prefix

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Identity to reveal the settings.

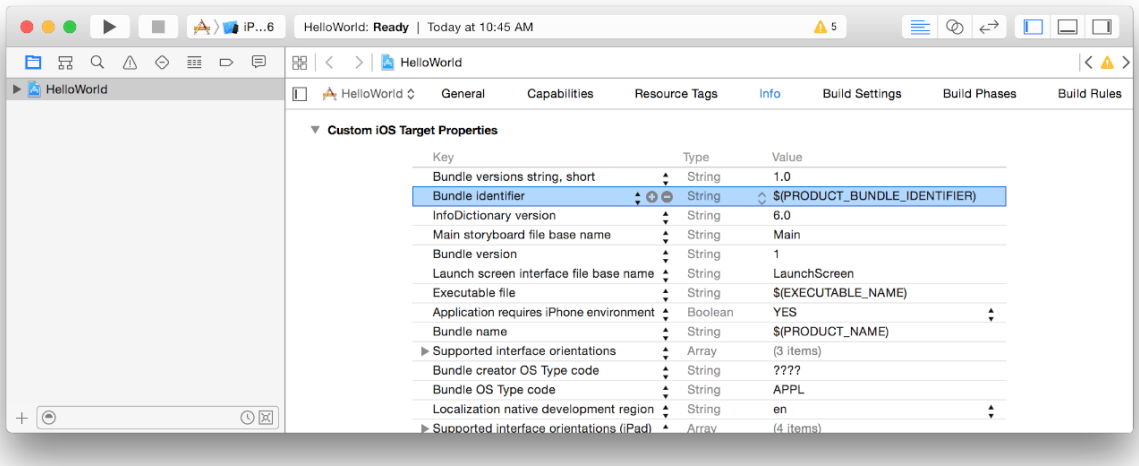


3. Enter the bundle ID prefix in the Bundle Identifier field.

Alternatively, follow these steps to change the entire bundle ID in the Info pane in the project editor.

To set the bundle ID

1. In the project navigator, select the project and your target to display the project editor.
2. Click Info.
3. Enter the bundle ID in the Value column of the "Bundle identifier" row.

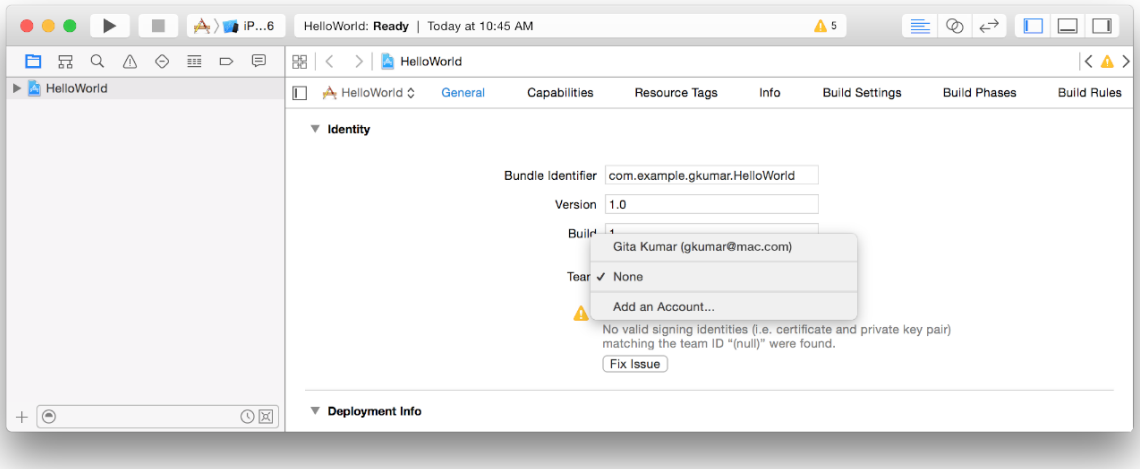


Assigning the Xcode Project to a Team

Each Xcode project is associated with a single team. If you enroll as an individual, you're considered a one-person team. The developer account is used to store the certificates, identifiers, and profiles needed to provision your app. All apps need to be code signed and provisioned to run on a device and use certain app services. Xcode creates these assets for you when needed, but you can avoid warnings and dialogs later if you set the team now.

To assign the project to a team

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Identity to reveal the settings.
3. Choose your team from the Team pop-up menu.
   - If you're an individual, choose your name from the pop-up menu.
   - If you're an organization, choose your organization name from the pop-up menu.



   Xcode may attempt to create a team provisioning profile. For iOS, tvOS, and watchOS apps, Xcode creates a team provisioning profile if you have a device connected to your Mac or have previously registered a device. For Mac apps, Xcode registers your development Mac. To resolve issues related to the team provisioning profile, read Creating the Team Provisioning Profile.
4. If a team doesn't appear in the Team pop-up menu, choose "Add an Account" and follow the steps described in Adding Your Apple ID Account in Xcode.

Choosing a Signing Identity (Mac)

You have the choice of submitting your Mac app to the store, signing it with a Developer ID certificate to distribute it outside of the store, signing it using an Apple ID that doesn't belong to the Apple Developer Program, or not code signing it at all. If you select Mac App Store, you assign your Xcode project to a team and can enable app services, as described in Adding Capabilities. If you select Developer ID or Apple ID, the available capabilities are limited. If you select None, the Team pop-up menu is disabled and you don't need to read this guide.

To select a signing identity

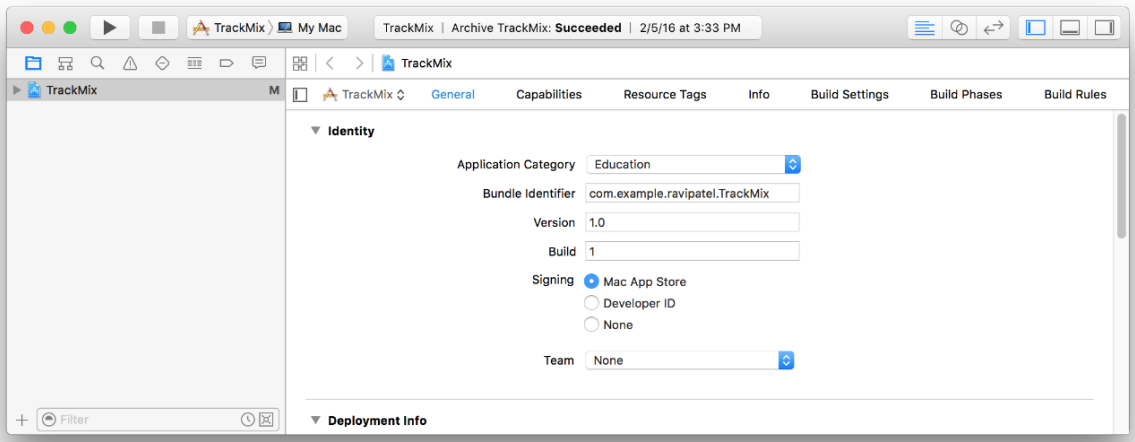
1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Identity to reveal the settings.
3. Select the type of signing identity you want to use.

   The options change depending on the Apple ID you choose from the Team pop-up menu.

   - If you want to submit your app to the store, choose the Apple ID that is a member of the Apple Developer Program from the Team pop-up menu, and select Mac App Store.
   - If you want to distribute your Mac app outside of the store, choose the Apple ID that is a member of the Apple Developer Program from the Team pop-up menu, and select Developer ID.
   - If you want to use an Apple ID that is not a member of the Apple Developer Program, choose the Apple ID from the Team pop-up menu, and select Apple ID.

- If you don't want to sign your app or use app services, select None.

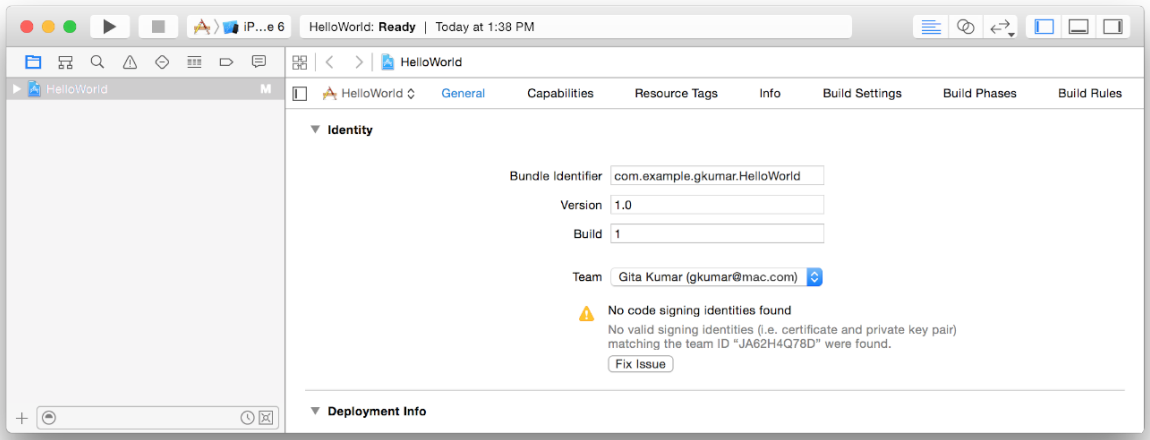
The capabilities available to your app depends on the signing identity you choose, described in Supported Capabilities.



Read Distributing Apps Outside the Mac App Store for other steps to create a Developer ID–signed app.

Creating the Team Provisioning Profile

After you select a team, Xcode attempts to create a specialized development provisioning profile called a *team provisioning profile* that it manages for you. A team provisioning profile allows an app to be signed and run by all team members on all their devices. If Xcode fails to create the team provisioning profile, a warning and Fix Issue button appear below the Team pop-up menu.



You can assist Xcode and avoid common problems by creating the team provisioning profile now. In most cases, you can just click Fix Issue below the warning message and Xcode performs these steps for you:

1. Creates your development certificate

   Xcode creates the development certificate for the team you selected from the Team pop-up menu.

2. Registers the device chosen in the Scheme toolbar menu or your Mac

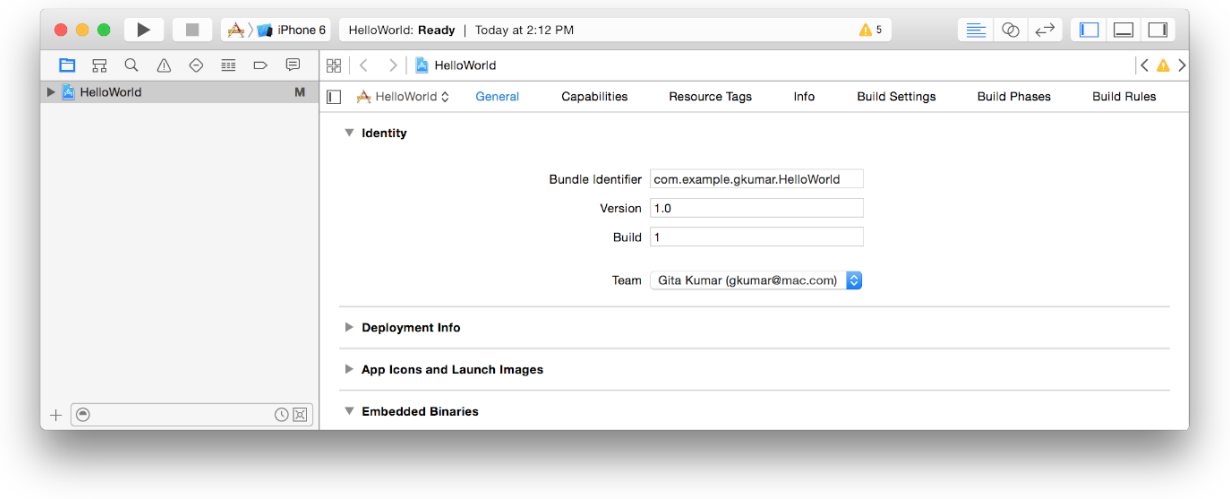
   Xcode requires one or more registered devices in your account before it can create a team provisioning profile. For Mac apps, Xcode automatically registers the Mac that's running Xcode. For iOS and tvOS apps, connect a device you want to use for development. For watchOS apps, connect an iPhone that is paired with an Apple Watch. If the device matches the deployment target, Xcode registers the device for you. If the device isn't eligible because it doesn't match the deployment target, upgrade the OS on the device or change the deployment target, as described in Setting Deployment Info.

3. Creates an App ID that matches your app's bundle ID and enables app services

Depending on your project's configuration, Xcode may create either a wildcard App ID or an explicit App ID. Because Xcode uses the bundle ID to register an explicit App ID, the bundle ID also needs to be unique during development. To avoid an App ID registration error, enter a unique bundle ID, as described in Setting the Bundle ID.

4. Creates a team provisioning profile containing these assets
5. Sets your project's code signing build settings accordingly

After Xcode performs these steps, the warning message under the Team pop-up menu disappears.



If dialogs or warnings appear, use the information to correct the problem and click Fix Issue again. If a "Your account already has a valid … certificate" dialog appears, choose one of these actions.

- If you previously exported your signing identity, as described in Exporting and Importing Certificates and Profiles, click Import Developer Profile and follow the instructions.
- If you don't have a backup of your signing identity, click Reset.

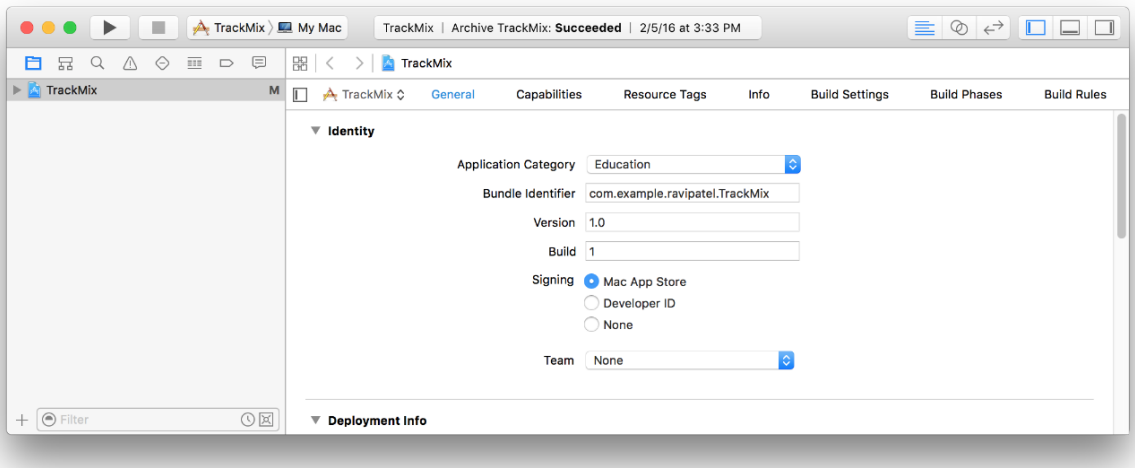
  Xcode revokes your development certificate and recreates the signing identity for you.

Setting the Application Category (Mac)

Set the category under which your Mac app will be listed on the Mac App Store. The category you select should match the category you later select in your iTunes Connect app record.

To set the application category

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Identity to reveal the settings.
3. Choose the category from the Application Category pop-up menu.



iOS, tvOS, and watchOS app categories are set in iTunes Connect only. For more details on app categories, read Best Practices in *iTunes Connect Developer Guide*.

Setting the Version Number and Build String

It's important to initial set, and later update, the version number and build string before uploading your app to iTunes Connect because these numbers are used by the store to identify the build.

The version number is a two-period-separated list of positive integers, as in `4.5.2`. The first integer represents a major revision, the second a minor revision, and the third a maintenance release. The version number is shown in the store and that version should match the version number you enter later in iTunes Connect. For details on possible values, see CFBundleShortVersionString in *Information Property List Key Reference*.

The build string represents an iteration (released or unreleased) of the bundle and is also a two-period-separated list of positive integers, as in `1.2.3`. For Mac apps, the user can click the version number in the About window to toggle between the version number and the build string. For details on possible values, see CFBundleVersion in *Information Property List Key Reference*.

> **Important:** Update the version number when you create a new app version in iTunes Connect. Update the build string before you upload a new build of your app to iTunes Connect. The number represented by the build string should be incremented. For iOS apps, iTunes will recognize that the build string changed and properly sync the new app version to the device. The version number and build string are also used to identify crash reports and `.dSYM` files for apps distributed through TestFlight or the store.

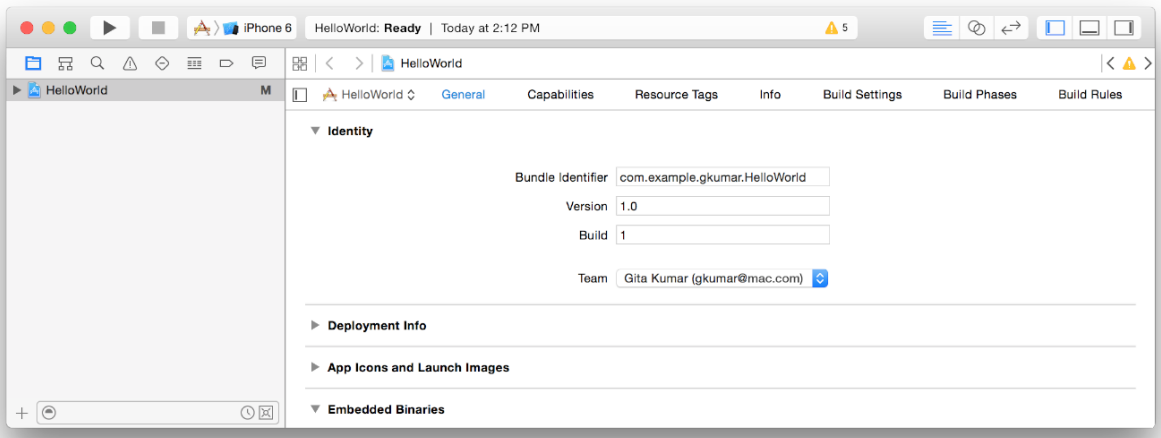
Set the version number and build string in the General pane in the project editor.

To set the version number and build string

1. In the project navigator, select the project and your target to display the project editor.

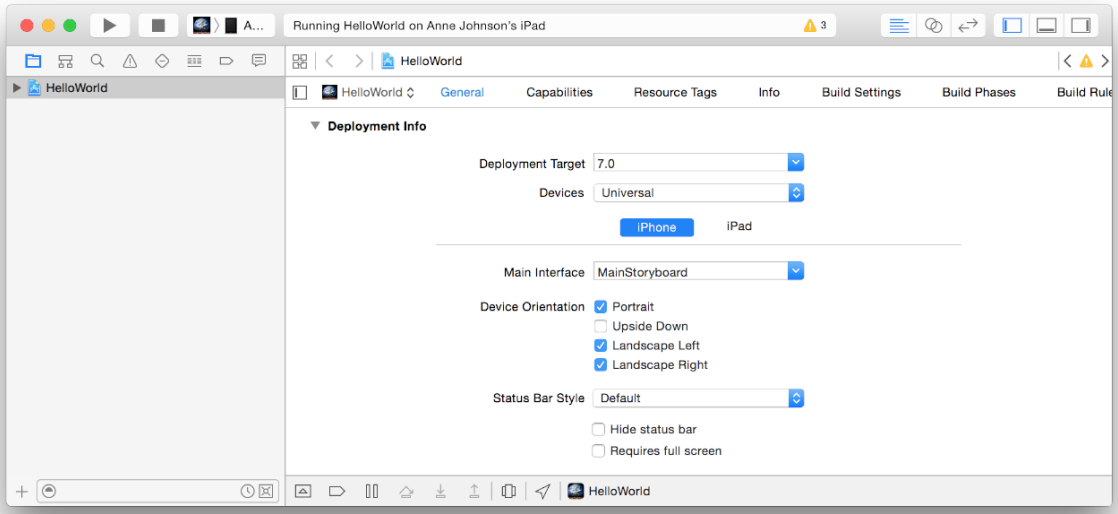
   > **Important:** For watchOS apps, the embedded WatchKit app and WatchKit extension must have the same version number and build string as the enclosing iOS app.

2. Click General and, if necessary, click the disclosure triangle next to Identity to reveal the settings.
3. Enter the version number in the Version field, and enter the build string in the Build field.



Setting Deployment Info

The default deployment settings are sufficient for development, but it's best to review these settings before you distribute your app. Some settings must match values you enter in iTunes Connect later. For iOS apps, the deployment settings appear as shown here:



Setting the Deployment Target

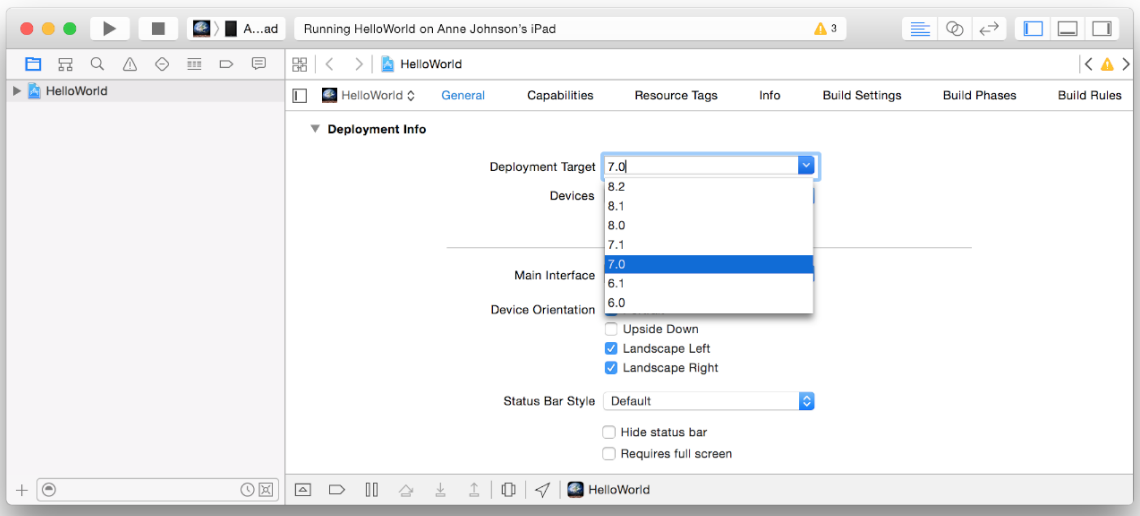
The deployment target setting specifies the lowest operating system version that your app can run on. For example, the lowest available setting for iPad apps is iOS 4.3.

There are several strategies for choosing the deployment target when developing your app. Each version of an OS includes features and capabilities not present in earlier versions. As new versions are published, some users may upgrade immediately, while other users may wait before moving to the latest version. You can target the latest version, taking full advantage of all the new features but limiting the app to only users running the latest version. Or you can target an earlier version, making your app available to more users but limiting the features you can use in the app. Another approach is to target an earlier version but use weak linking to determine at runtime whether later version features are available before using them.

For details on weak linking, read Weak Linking and Apple Frameworks in *SDK Compatibility Guide*.

To set the target version

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Deployment Info to reveal the settings.
3. Choose the version you want to target from the Deployment Target pop-up menu.



Xcode sets the Minimum System Version key in the app's information property list to the deployment target you choose. When you publish your app to the store, it uses this property value to indicate which versions your app supports.

Note: The SDK version, not the deployment target, determines which features you can use in an app. If the SDK you're using to build the app is more recent than the app's deployment target, Xcode displays build warnings when it detects that your app is using a feature that's unavailable in the deployment target.

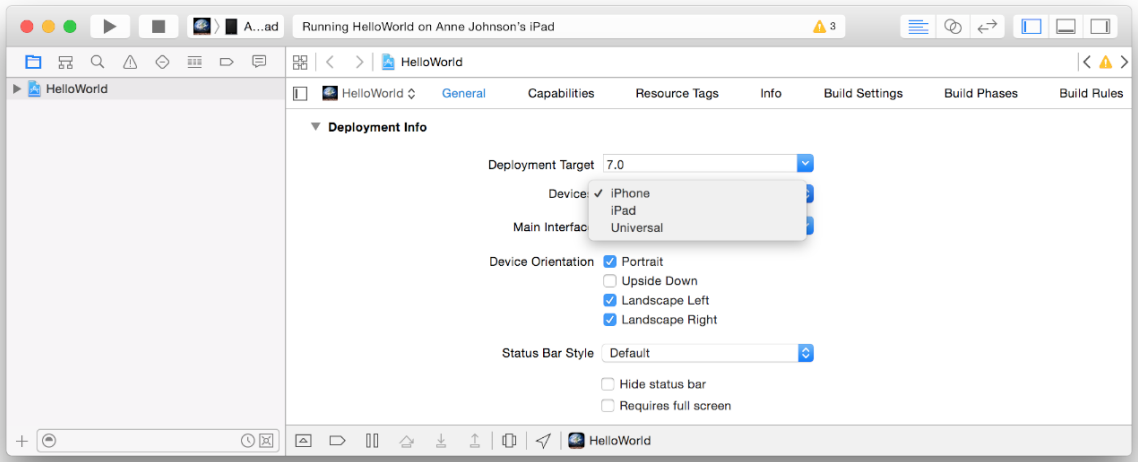
Ensure that the symbols you use are available in the app's runtime environment. To check for their availability, use the techniques described in *SDK Compatibility Guide*.

Setting the Target Devices (iOS, watchOS)

The Devices setting identifies the type of devices you want an app to run on. There are two device types: iPhone and iPad. The iPhone type includes iPhone and iPod touch devices. The iPad type includes all iPad and iPad mini devices.

To set the target devices

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to Deployment Info to reveal the settings.
3. From the Devices pop-up menu, choose iPhone, iPad, or Universal (to target both families).



For more information on configuring your app for iPhone, iPad, or both device families, see Advanced App Tricks in *App Programming Guide for iOS*.

Adding App Icons and a Launch Screen File

App icons and the launch screen file are stored in the app bundle, not uploaded as separate assets to iTunes Connect. The operating system uses these images and the launch screen file (a `.xib` file) in various locations on a device to represent your app. In general, artwork displayed by the operating system resides in the bundle, and artwork displayed by iTunes or the store is uploaded to iTunes Connect. Your app needs an app icon to represent it and pass validation tests.

You use an asset catalog that manages the app icons for you or maintain individual image files yourself. If you create a new project, asset catalogs are used by default to store app icons and a launch screen file is used as a splash screen. For iOS 7 deployment targets, you can supply both a launch screen file and launch images. In iOS 8, the launch screen file is used, and in iOS 7, the launch images are used. If you have an older project, you can migrate from managing individual icon image files to an asset catalog. You can also add a launch screen file to an older Xcode project.

If you prefer to manage your assets as individual image files, read Setting Individual App Icon and Launch Image Files. If you want to migrate to an asset catalog, read Migrating Your Images to an Asset Catalog. To learn more about creating and managing asset catalogs, read *Asset Catalog Help*.

Preparing Your Artwork

For all artwork, keep the file size as small as possible for a positive purchase experience for your users.

For iOS apps, see Icon and Image Sizes in *iOS Human Interface Guidelines* for the sizes of all required app icons, launch images, and other icons.

For iOS and tvOS apps, a launch image matching the device resolution appears as soon as the user taps your app icon. Use screenshots to create your app's launch images. To take advantage of Retina displays, provide high-resolution images for each device you support. For guidance on creating launch images, read Launch Images.

For the required Mac app icons, see Creating Great Icons for Any Resolution in *OS X Human Interface Guidelines*. This table includes icon sizes that may be used on the Mac App Store. To create your app icon files, read Provide High-Resolution Versions of All App Graphics Resources in *High Resolution Guidelines for OS X*.

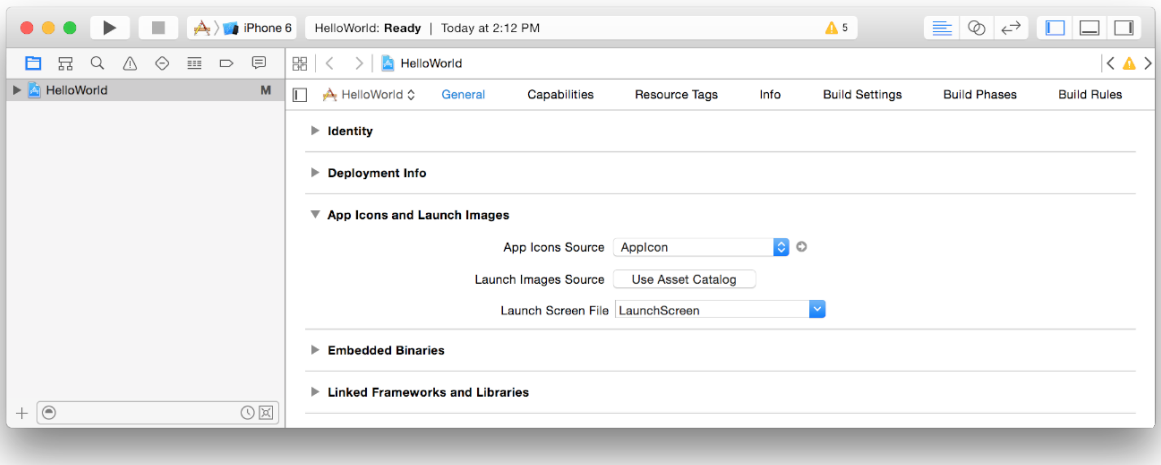
For the specification of screenshots and other artwork that you upload later using iTunes Connect, read iTunes Connect App Properties in *iTunes Connect Developer Guide*.

Adding App Icons to an Asset Catalog

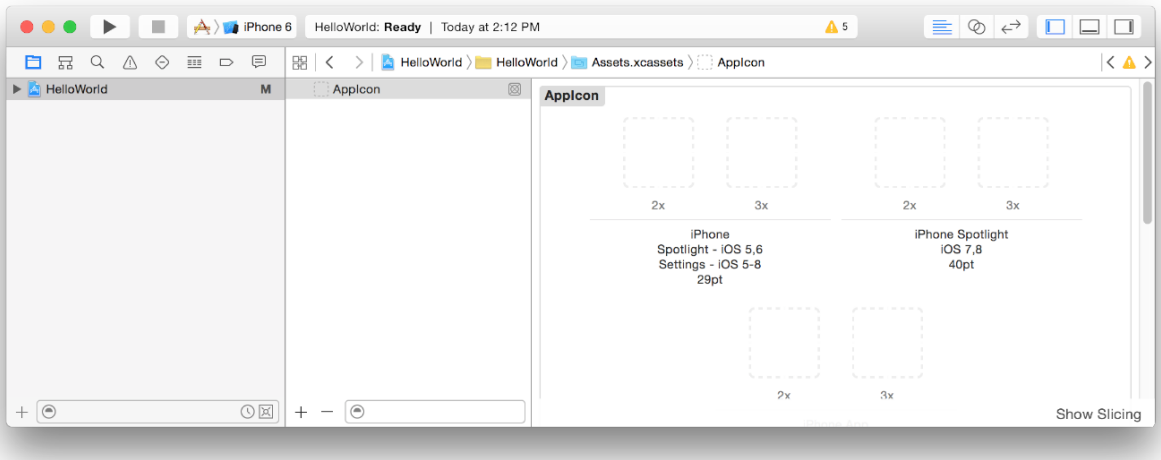
Versions of your app icons are organized into image sets in an asset catalog. Xcode automatically creates image sets for the app's target device—for example, both iPhone and iPad image sets appear if your iOS app's target is universal.

To add an app icon to an asset catalog

1. In the project navigator, select the project and your target to display the project editor.
2. In the "App Icons and Launch Images" section of the General pane, click the arrow button to the right of the App Icons Source pop-up menu.



3. In the Finder, drag an app icon to the image well that matches its resolution in the project navigator.



Creating a Launch Screen File

The launch screen file is displayed as a splash screen while your app is launching. It's a single, atomic `.xib` file that uses size classes to support different device resolutions. It contains basic UIKit views, such as UIImageView and UILabel objects, and uses Auto Layout constraints. Xcode adds a default launch screen file, called `LaunchScreen.xib`, to your project.

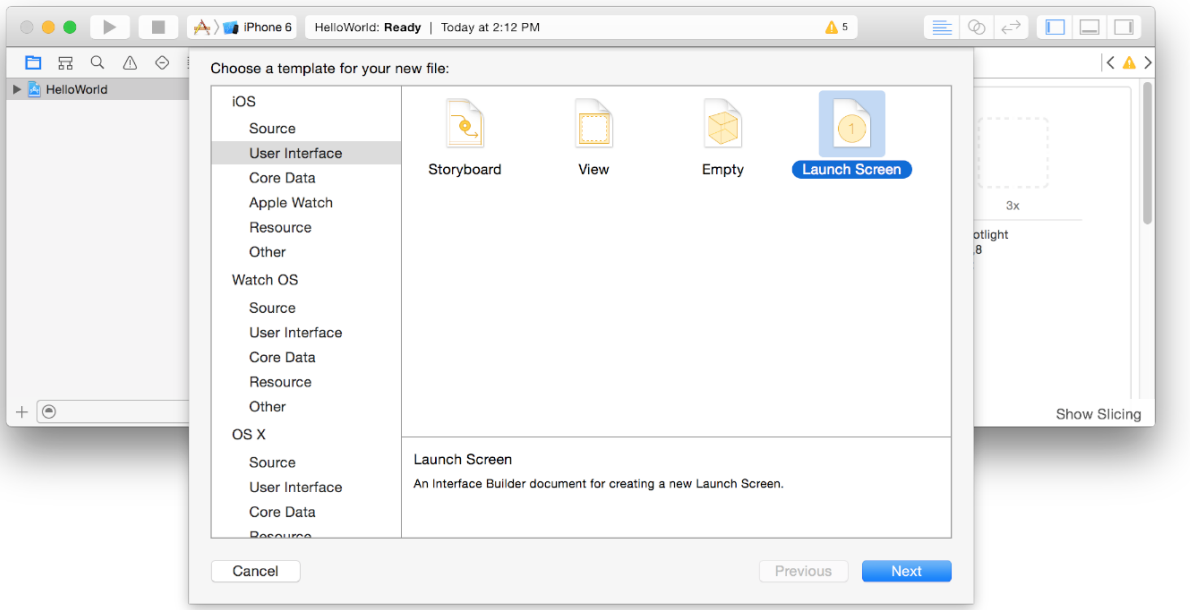
Follow these guidelines when creating a launch screen file:

- Use only UIKit classes.
- Use a single root view that is a UIView or UIViewController object.
- Don't make any connections to your code (don't add actions or outlets).
- Don't add UIWebView objects.
- Don't use any custom classes.
- Don't use runtime attributes.

You can add a launch screen file to an older Xcode project.

To create a launch screen file for an existing project

1. Choose File > New > File.
2. Under iOS, select User Interface.
3. Click Launch Screen and click Next.



4. Enter a filename in the Save As text field, and click Create.

To set the launch screen file

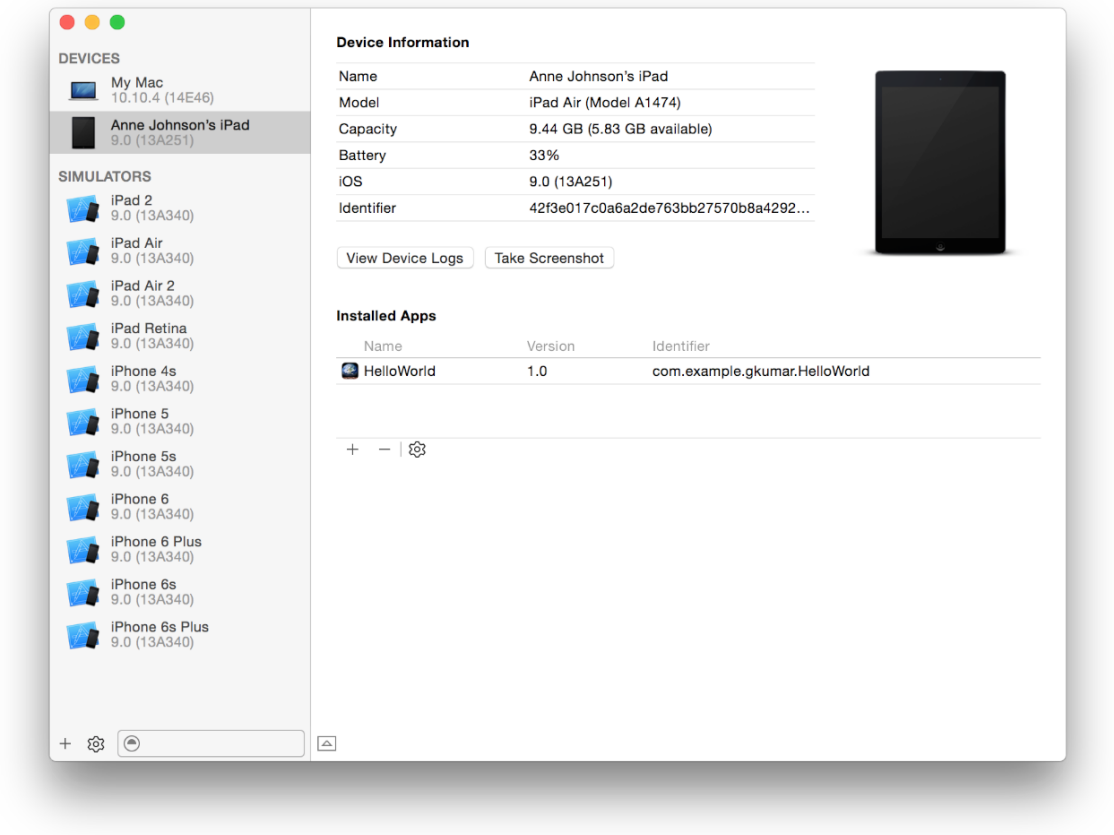
1. If necessary, open the "App Icons and Launch Images" section of the General pane.
2. From the Launch Screen File pop-up menu, choose a launch screen file.

Adding Launch Images and Capturing Screenshots

Follow these steps to capture a screenshot of your app while your device is connected to your Mac. Although the screenshot of an iOS app includes the status bar as it looks when the screenshot is captured, iOS replaces it with the current status bar when your app launches.

To capture a screenshot of your app running on a device

1. Connect the device to your Mac.
2. Choose Window > Devices, and select the device under Devices.
3. In the Device Information section, click Take Screenshot.



The screenshot appears on your desktop.

Note: For iOS 7 deployment targets, add an asset catalog for launch images, described in Migrating Your Images to an Asset Catalog. Capture screenshots to create the launch images and add them to the asset catalog, similar to adding app icons to an asset catalog, described in Adding App Icons to an Asset Catalog.

Setting Individual App Icon and Launch Image Files

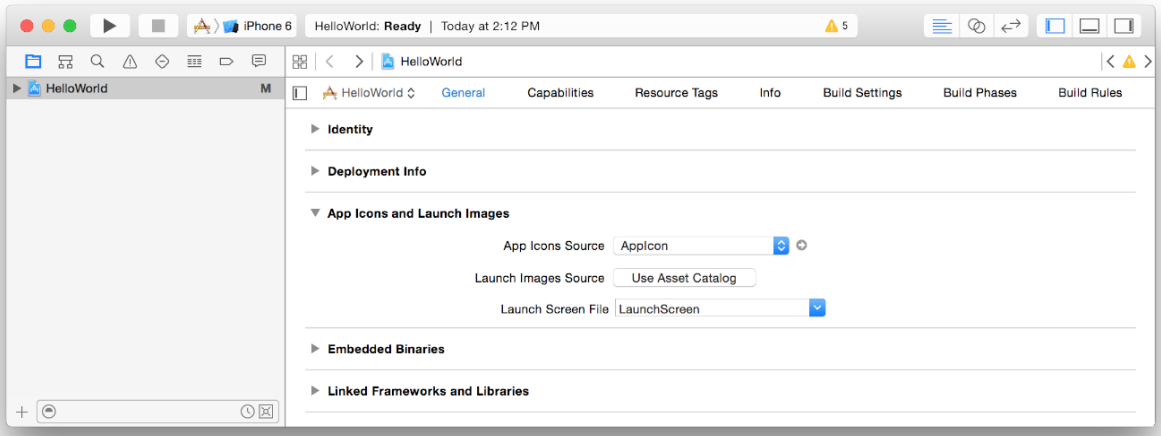
If you don't want to use an asset catalog, choose "Don't use asset catalogs" from either the App Icons Source or Launch Images Source pop-up menu in the "App Icons and Launch Images" section of the General pane. If a Use Asset Catalog button appears, you are not using an asset catalog.

Migrating Your Images to an Asset Catalog

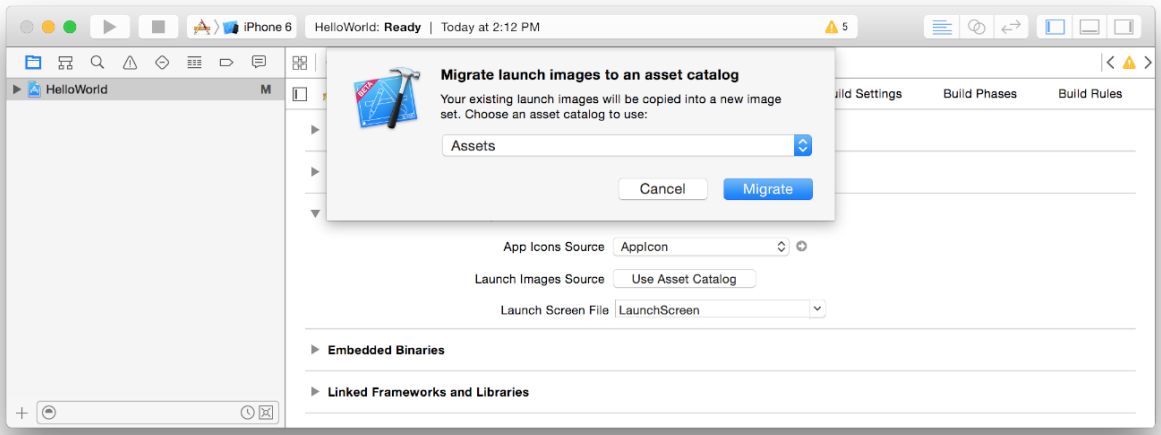
If you have an older project, you can migrate your image files to an asset catalog that manages your app icons and launch images for you. Xcode moves the image files from the tables to the new asset catalog. You create a separate asset catalog for app icons and launch images, but the steps are identical.

To migrate to an asset catalog

1. In the project navigator, select the project and your target to display the project editor.
2. Click General and, if necessary, click the disclosure triangle next to "App Icons and Launch Images."
3. Click the Use Asset Catalog button next to either App Icons Source or Launch Images Source.



4. In the dialog that appears, choose either Assets or New Asset Catalog from the pop-up menu, and click Migrate.



To view the asset catalog, click the arrow button to the right of the App Icons Source or Launch Images Source pop-up menu.

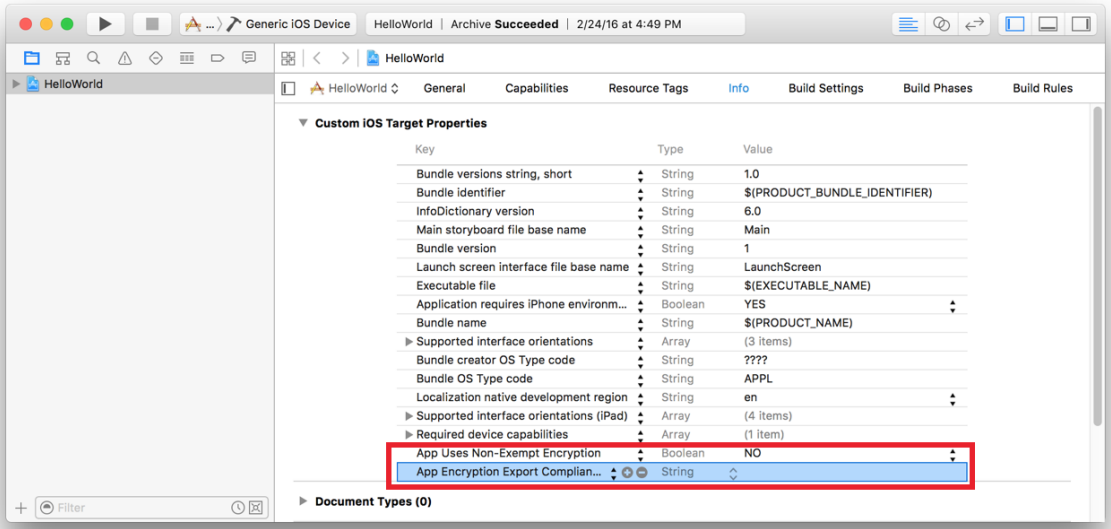
Adding Export Compliance Information to Your Project (Optional)

You can optionally specify export compliance information for each build that you upload to iTunes Connect. If your app uses encryption and requires export compliance documentation, you upload the document and submit it for review using iTunes Connect, as described in Cryptography and U.S. Export Compliance in *iTunes Connect Developer Guide*. Once the document is approved, iTunes Connects will provide key-value pairs that you can add to the `Info.plist` file in your Xcode project. If you provide the export compliance information in the build, then you don't need to provide this information later when you distribute your app using TestFlight or submit it to App Review.

To add the export compliance keys to your Xcode project

1. In the Project window, choose the target from the Project/Targets pop-up menu or in the Targets section of the second sidebar if it appears.
2. Click Info.
3. In the last row of the target properties table, click the Add button (+).
4. In the search field, enter the export compliance key followed by the Return character and set its value.

- If your app is not using encryption, add the `App Uses Non-Exempt Encryption` key and set the value to `NO`.
- If your app uses encryption, add the `App Uses Non-Exempt Encryption` key and set the value to `YES`.
- If your app requires export compliance, add the `App Encryption Export Compliance Code` key and enter the value for this key provided by iTunes Connect.



Note: In Xcode 7.2 and earlier, enter both the keys and values manually.

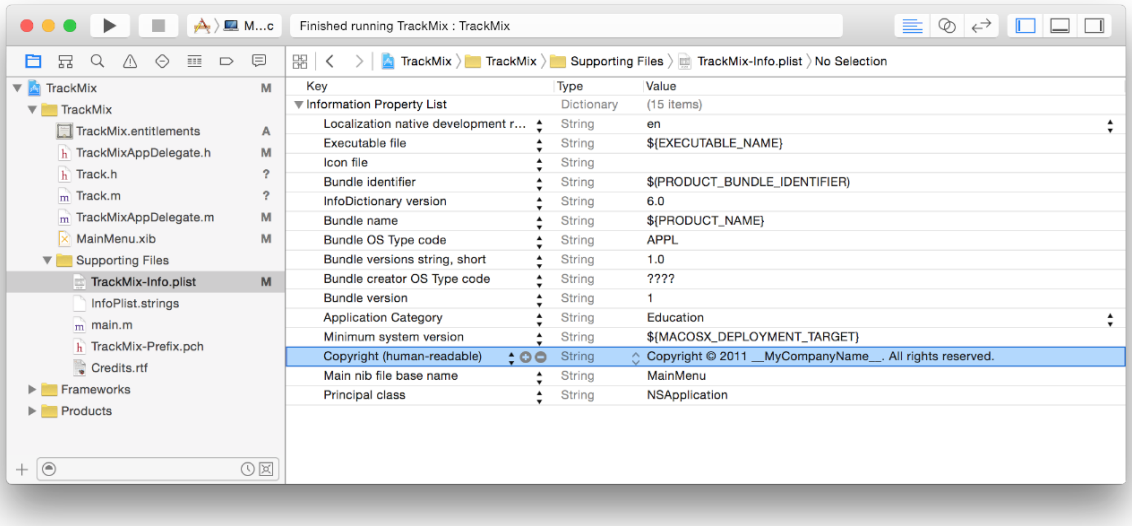
Setting the Copyright Key (Mac)

Make sure that your information property list file contains a valid value for the `Copyright` key. For details on possible values, see NSHumanReadableCopyright in *Information Property List Key Reference*.

To edit the copyright key in the information property list

1. In Xcode, select the project in the project navigator.
2. Click the disclosure triangle next to the *ProjectName* folder to reveal its contents.
3. Click the disclosure triangle next to the `Supporting Files` subfolder to reveal its contents.
4. Select the *ProjectName*`-Info.plist` file.

   The information property list is displayed to the right in a property list editor.



5. Double-click in the Value column and in the row of the Copyright key.
6. Enter a new value for the key.

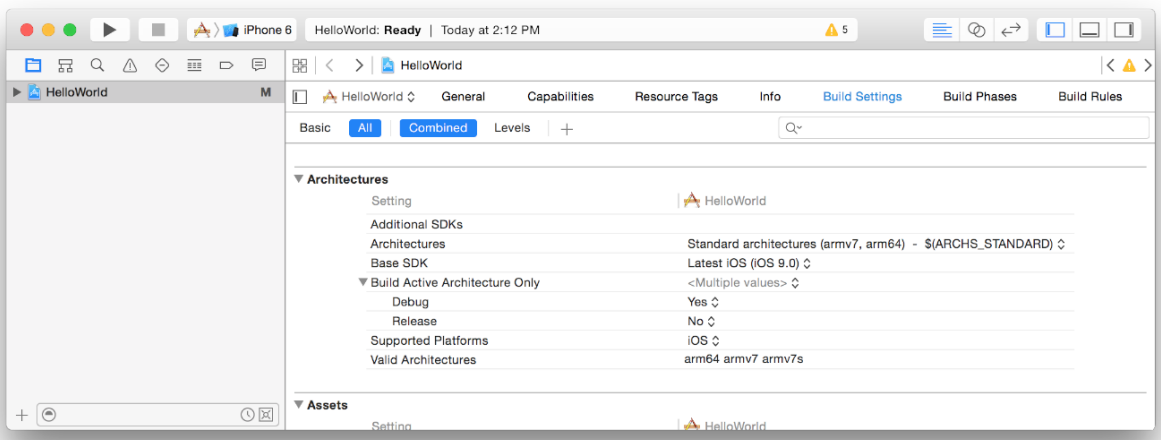
For how to edit other cells in a property list, refer to *Property List Editor Help*.

Verifying Your Build Settings

If you changed the default build settings, verify some of the settings before submitting your app to the store. You do this in the Build Settings pane of the project editor.

To edit a build setting

1. In the project editor, select the project or target whose build setting you want to edit.
2. At the top of the project editor, click Build Settings.



3. Locate the build setting in the left column, or enter the name of the build setting in the search field in the upper-right corner.
4. If some build settings don't appear, click All.
5. Set the value for the build setting in the right column.

Setting Architectures for iOS Apps

The Architectures build setting identifies the architectures for which your app is built. An iOS device uses a set of architectures, which include `armv7` and `arm64`. You have two options for specifying the value of this setting:

- **Standard.** Produces an app binary with a common architecture, compatible with all supported devices. This option generates the smallest app, but it may not be optimized to run at the best possible speed for all devices.
- **Other.** Produces an app binary for a specified set of architectures.

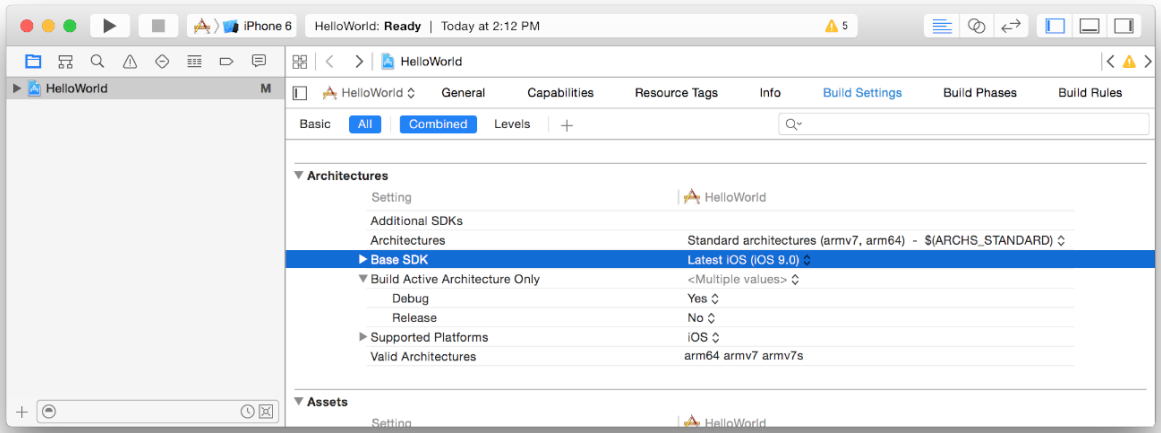
If you select Other from the Architectures build-setting value list, click the Add button (+) to enter the custom device architecture names you support.

> **Important:** The store rejects a build that supports only `armv7s`. If `armv7s` is included in the Valid Architectures list, `armv7` must also be included.

Setting the Base SDK

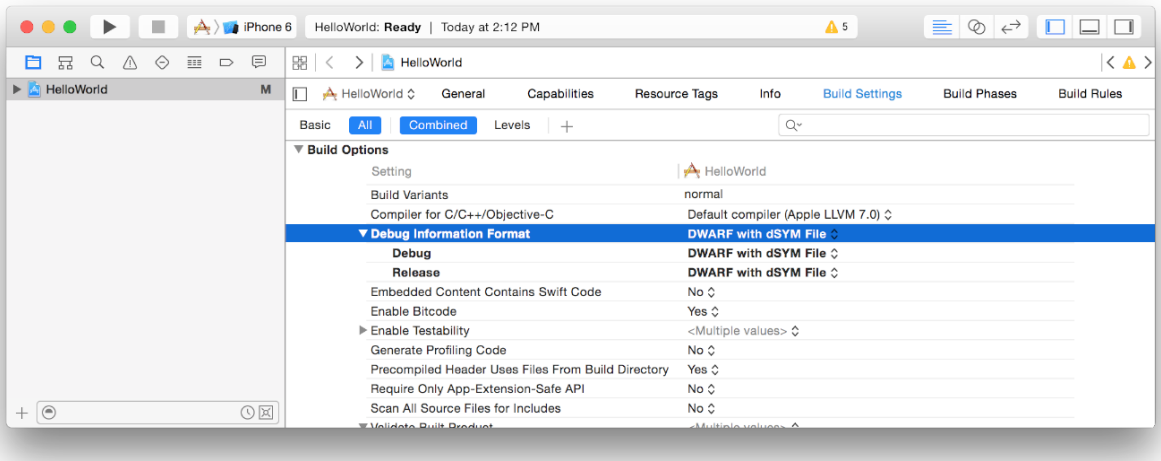
The Base SDK version number must be greater than or equal to the software version number on your development device; otherwise, Xcode can't initiate a debugging session with the device. Set the Base SDK for your project and targets to the latest operating system, which is the default value. The Base SDK property is located in the Architectures area in the Build Settings pane. For iOS apps, set Base SDK to Latest iOS. For Mac apps, set Base SDK to Latest OS X. If you select another value, download and install the latest SDK version that's greater than or equal to your device software version.

To go to the Architectures area, select the project or target and click Build Settings. The Architectures area appears first in the Build Settings pane.



Setting the Debug Information Format

Set the Debug Information Format build setting to "DWARF with dSYM File." This is required to symbolicate crash reports, as described in Analyzing Crash Reports.



Recap

In this chapter, you learned how to configure your Xcode project from a template, set the app's identity information, and create a team provisioning profile for development. During development, refer to this chapter as needed. Later, use this chapter as a checklist for the settings that are required by the store.