Introduction

- "Setup" (page 6)
- "Tutorial: Basics" (page 8)

Setup

Start Developing iOS Apps Today provides the perfect starting point for iOS development. On your Mac, you can create iOS apps that run on iPad, iPhone, and iPod touch. View this guide's four short modules as a gentle introduction to building your first app—including the tools you need and the major concepts and best practices that will ease your path.

The first three modules each end with a tutorial, where you'll implement what you've learned. At the end of the last tutorial, you'll have created a simple to-do list app.

After you've built your first app in this guide and are considering your next endeavor, read the fourth module. It explores the technologies and frameworks you might consider adopting in your next app. You'll be on your way to keeping your customers engaged and looking forward to the next great thing.

Even though this guide takes you through every step of building a simple app, to benefit most it helps to be acquainted with computer programming in general and with object-oriented programming in particular.

Get the Tools

Before you can start developing great apps, set up a development environment to work in and make sure you have the right tools.

To develop iOS apps, you need:

- A Mac computer running OS X 10.7 (Lion) or later
- Xcode
- iOS SDK

Xcode is Apple's integrated development environment (IDE). Xcode includes a source editor, a graphical user interface editor, and many other features. The iOS SDK extends the Xcode toolset to include the tools, compilers, and frameworks you need specifically for iOS development.

You can download the latest version of Xcode for free from the App Store on your Mac. (The App Store app is installed with OS X version 10.7 and later. If you have an earlier version of OS X, you need to upgrade.) The iOS SDK is included with Xcode.

To download the latest version of Xcode

1. Open the App Store app on your Mac (by default it's in the Dock).
2. In the search field in the top-right corner, type `Xcode` and press the Return key.
3. Click Free FREE.

   Xcode is downloaded into your `/Applications` directory.

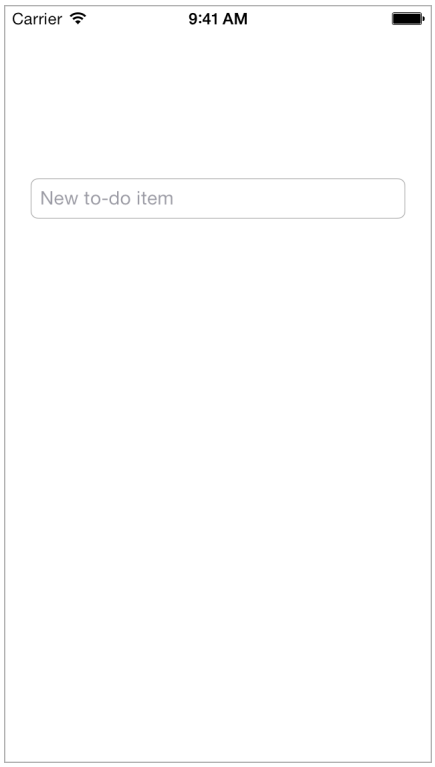
Tutorial: Basics

This tutorial takes you through the basics of what an app is, to the process of creating a simple **user interface**, and on to adding the custom behavior that transforms the interface into a working app.

Follow this tutorial to learn the basics of iOS app development, including:

- How to use Xcode to create and manage a project
- How to identify the key pieces of an Xcode project
- How to add standard user interface elements to your app
- How to build and run your app

After you finish the tutorial, you'll have an app that looks something like this:



To keep things simple, the tutorial project has only an iPhone interface, but you use the exact same tools and techniques to develop an iPad app. This tutorial uses Xcode 5.0 and iOS SDK 7.0.

Create a New Project

To get started developing your app, create a new Xcode project.

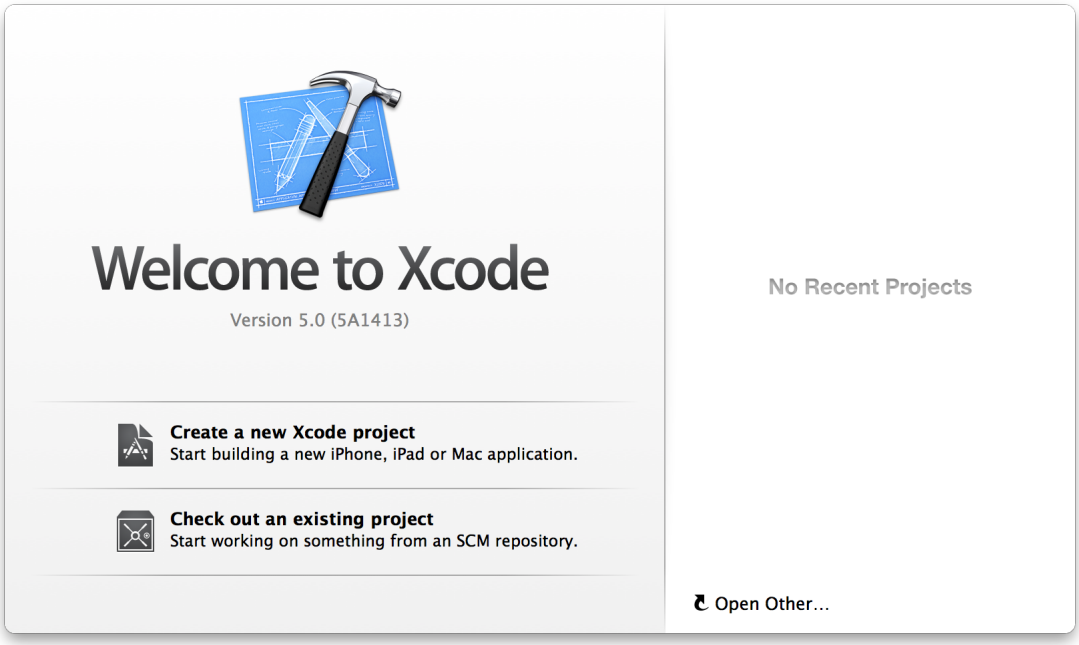
Xcode includes several built-in app **templates** that you can use to develop common styles of iOS apps, such as games, apps with tab-based navigation, and table-view-based apps. Most of these templates have preconfigured interface and source code files for you to start working with. For this tutorial, you'll start with the most basic template: Empty Application.

Working with the Empty Application template will help you understand the basic structure of an iOS app and how content gets onscreen. After you've learned how everything works, you can use one of the other templates for your own app to save yourself some configuration time.

To create a new empty project

1. Open Xcode from the `/Applications` directory.

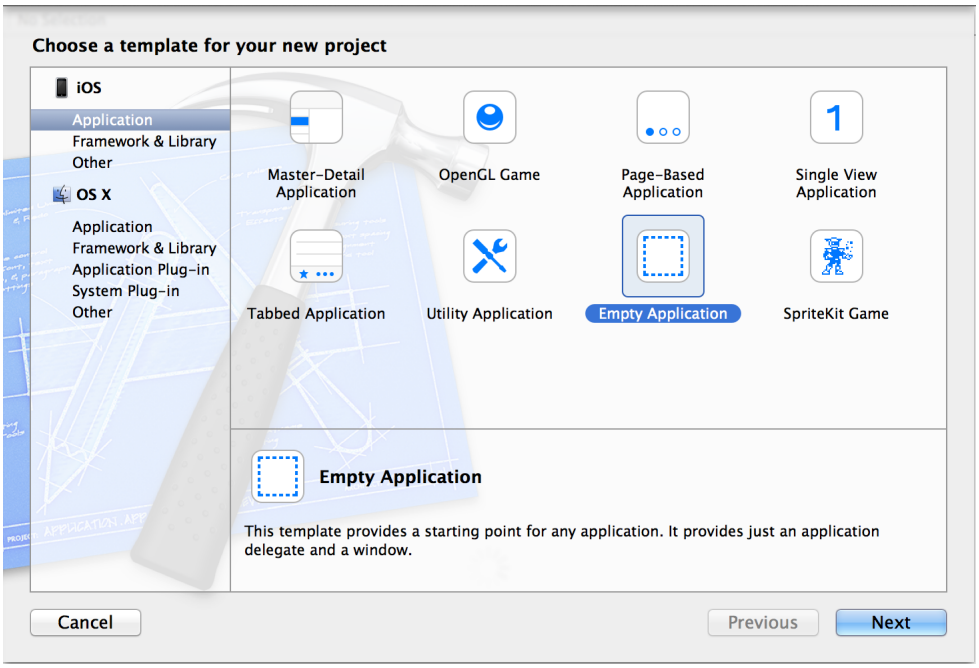
   The Xcode welcome window appears.



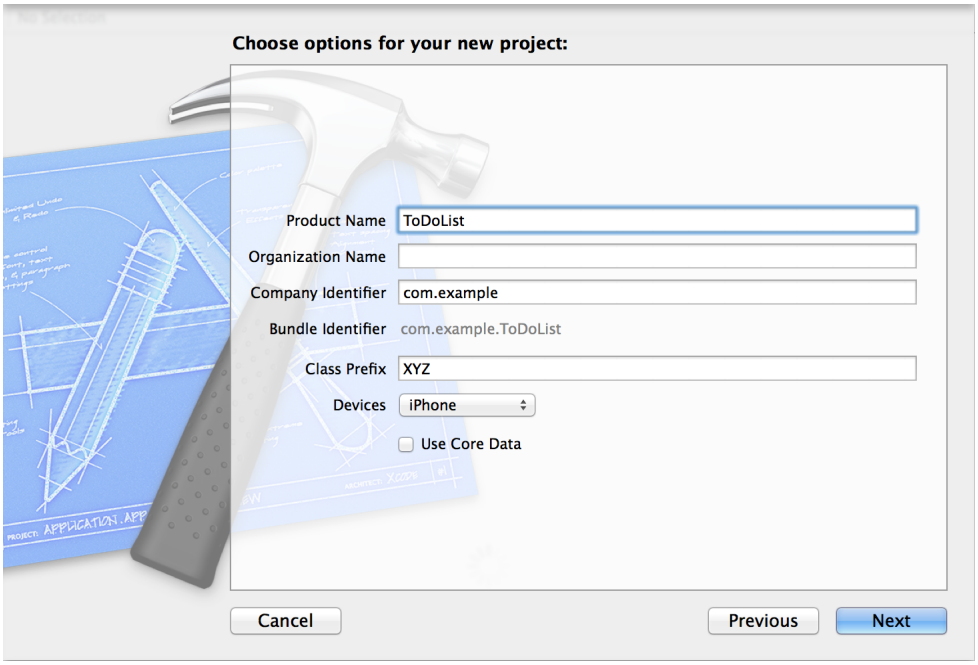
   If a project window appears instead of the welcome window, don't worry—you probably created or opened a project in Xcode previously. Just use the menu item in the next step to create the project.

2. In the welcome window, click "Create a new Xcode project" (or choose File > New > Project).

Xcode opens a new window and displays a dialog in which you can choose a template.



3. In the iOS section at the left of the dialog, select Application.
4. In the main area of the dialog, click Empty Application and then click Next.
5. In the dialog that appears, name your app and choose additional options for your project.



Use the following values:

- Product Name: `ToDoList`

Xcode uses the product name you entered to name your project and the app.

- Company Identifier: Your company identifier, if you have one. If you don't, use `com.example`.
- Class Prefix: `XYZ`

  Xcode uses the class prefix name to name the classes it creates for you. Objective-C classes must be named uniquely within your code and across any frameworks or bundles you might be using. To keep class names unique, the convention is to use prefixes for all classes. Two-letter prefixes are reserved by Apple for use in framework classes, so use something that's three letters or longer.

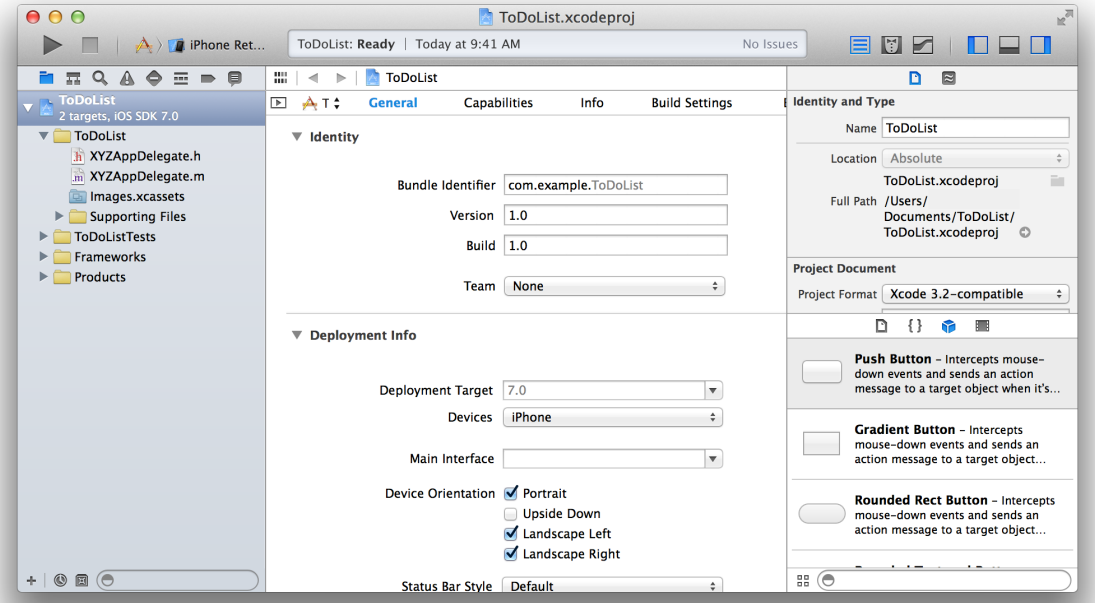
6. Choose iPhone from the Devices pop-up menu.

   As already mentioned, creating an app with an iPhone interface is the simplest way to start. The techniques used are the same for an iPad or universal app.

7. Click Next.

8. In the dialog that appears, choose a location for your project and click Create.

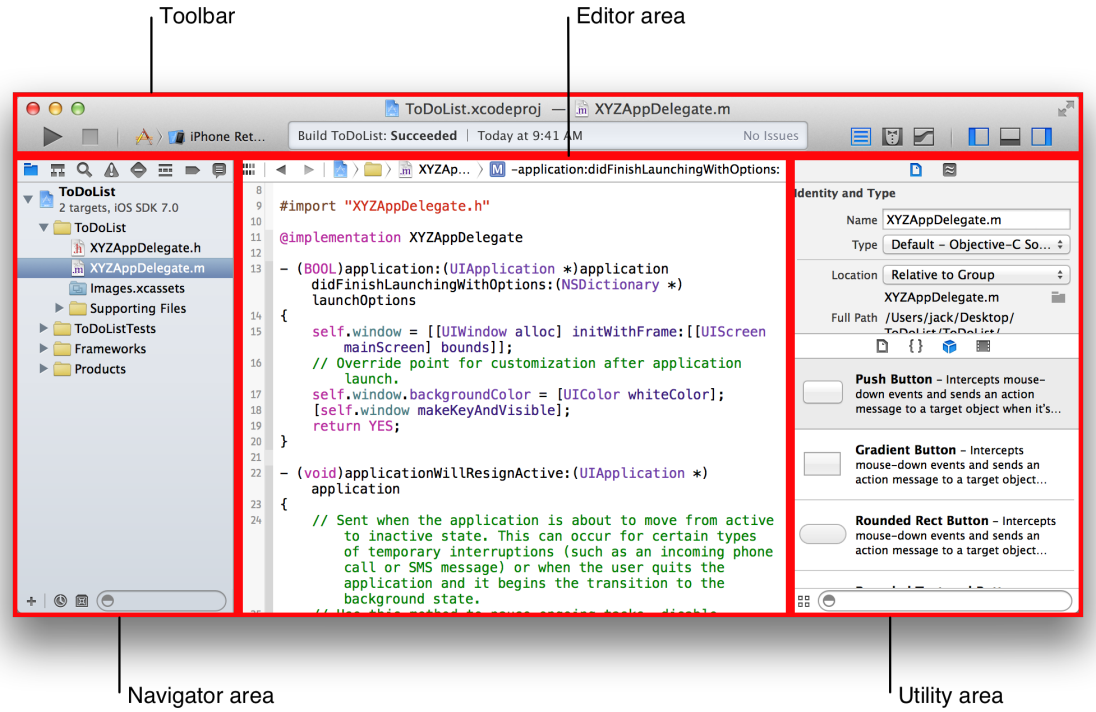
   Xcode opens your new project in a window (called the **workspace window**), which should look similar to this:



Get Familiar with Xcode

Xcode includes everything you need to create an app. It not only organizes the files that go into creating an app, it provides editors for code and interface elements, allows you to build and run your app, and includes a powerful integrated debugger.

Take a few moments to familiarize yourself with the Xcode workspace. You'll use the controls identified in the window below throughout the rest of this tutorial. Click different buttons to get a feel for how they work. If you want more information on part of the interface, read the help articles for it—you find them by Control-clicking an area of Xcode and choosing the article from the shortcut menu that appears.



Run iOS Simulator

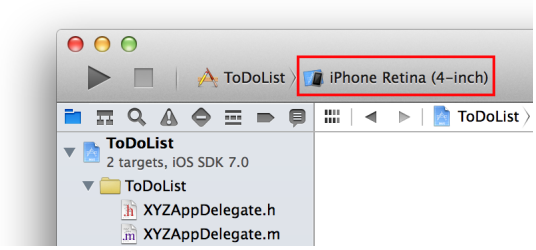
Because you based your project on an Xcode template, the basic app environment is automatically set up for you. Even though you haven't written any code, you can build and run the Empty Application template without any additional configuration.

To build and run your app, you can use the **iOS Simulator** app that's included in Xcode. As its name implies, iOS Simulator gives you an idea of how your app would look and behave if it were running on an iOS device.

iOS Simulator can model a number of different types of hardware—iPad, iPhone with different screen sizes, and so on. As a result, you can simulate your app on every device you're developing for. In this tutorial, use the iPhone Retina (4-inch) option.

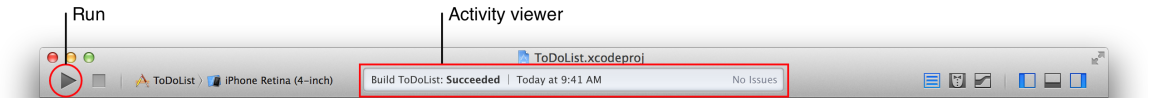
To run your app in iOS Simulator

1. Choose iPhone Retina (4-inch) from the Scheme pop-up menu in the Xcode toolbar.



   Go ahead and look through the menu to see what other hardware options are available in iOS Simulator.

2. Click the Run button, located in the top-left corner of the Xcode toolbar.



   Alternatively, you can choose Product > Run (or press Command-R).

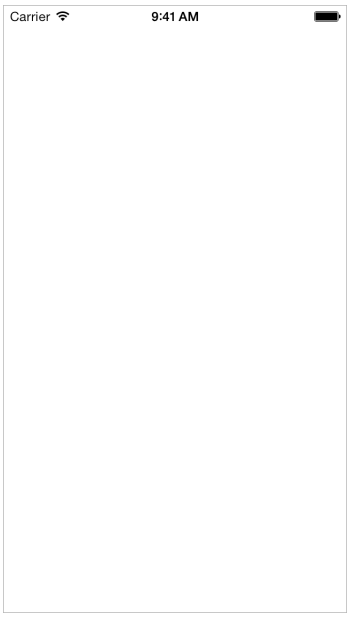
   If this is the first time you're running an app, Xcode asks whether you'd like to enable developer mode on your Mac. Developer mode allows Xcode access to certain debugging features without requiring you to enter your password each time. Decide whether you'd like to enable developer mode and follow the prompts. If you choose not to enable it, you may be asked for your password later on. The tutorials assume developer mode is enabled.

3. Watch the Xcode toolbar as the build process completes.

   Xcode displays messages about the build process in the **activity viewer**, which is in the middle of the toolbar.

After Xcode finishes building your project, iOS Simulator starts automatically. It may take a few moments to start up the first time.

iOS Simulator opens in iPhone mode, just as you specified. On the simulated iPhone screen, iOS Simulator opens your app.



As the name Empty Application implies, the template doesn't do much—it just displays a white screen. Other templates have more complex behavior. It's important to understand a template's uses before you extend it to make your own app. Running the template with no modifications is a good way to start developing that understanding.

After you've explored the app, quit iOS Simulator by choosing iOS Simulator > Quit iOS Simulator (or pressing Command-Q).

Review the Source Code

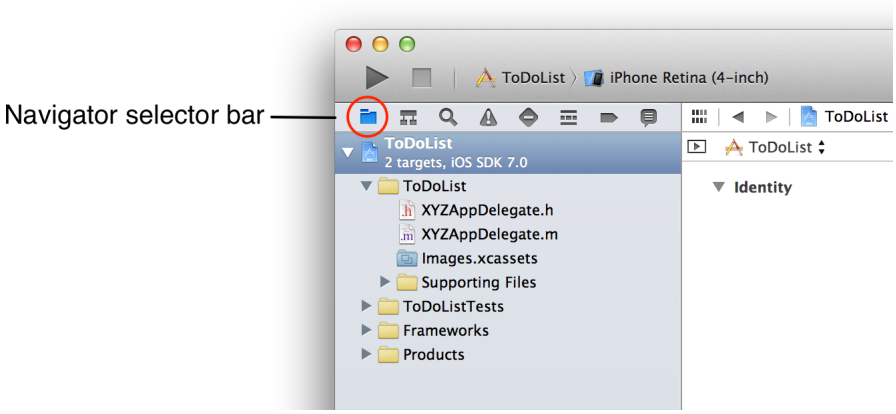
The Empty Application template comes with a few existing source code files that set up the app environment. Most of the work is done by the `UIApplicationMain` function, which is automatically called in your project's `main.m` source file. The `UIApplicationMain` function creates an application object that sets up the infrastructure for your app to work with the iOS system. This includes creating a **run loop** that delivers input events to your app.

You won't be dealing with the `main.m` source file directly, but it's interesting to understand how it works.

To look at the main.m source file

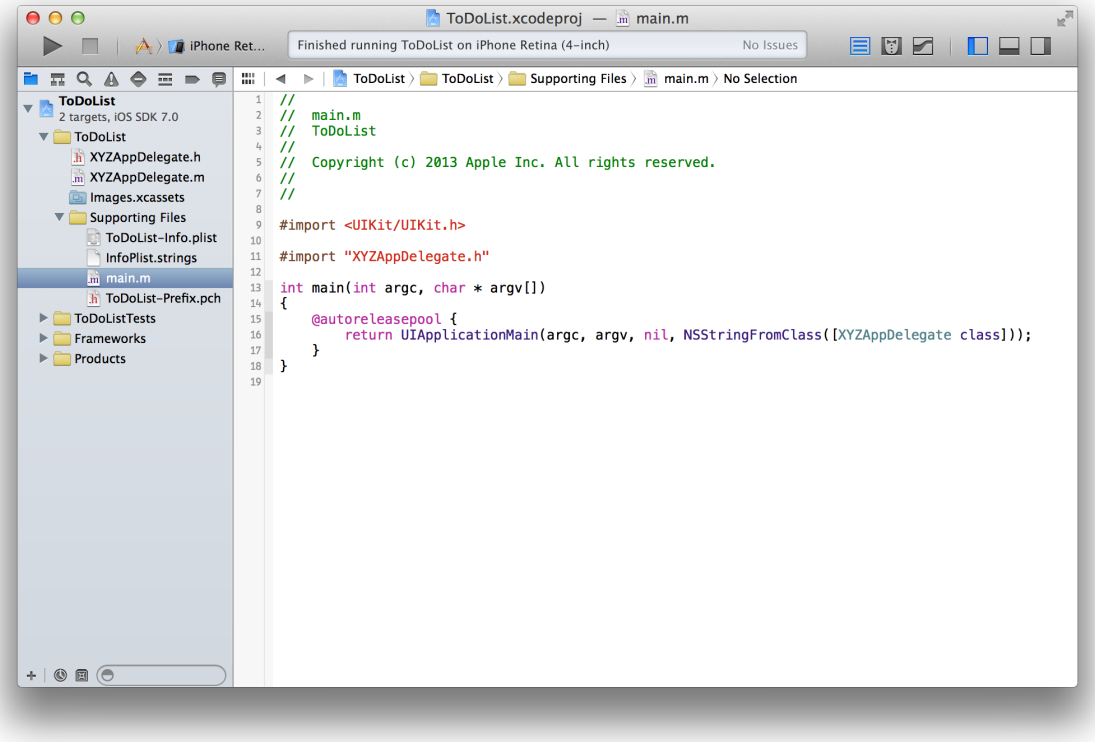
1. Make sure the project navigator is open in the navigator area.

The **project navigator** displays all the files in your project. If the project navigator isn't open, click the leftmost button in the navigator selector bar.



2. Open the Supporting Files folder in the project navigator by clicking the disclosure triangle next to it.
3. Select `main.m`.

   Xcode opens the source file in the main editor area of the window, which looks similar to this:



If you double-clicked the file, you'll notice that it opened in a separate window. You can choose to have a file open in the main project window by clicking it once, or you can open it in a separate window by double-clicking it.

The `main` function in `main.m` calls the `UIApplicationMain` function within an autorelease pool.

```
@autoreleasepool {
   return UIApplicationMain(argc, argv, nil, NSStringFromClass([HelloWorldAppDelegate
 class]));
}
```

The `@autoreleasepool` statement is there to support memory management for your app. Automatic Reference Counting (ARC) makes memory management straightforward by getting the compiler to do the work of keeping track of who owns an object; `@autoreleasepool` is part of the memory management infrastructure.

The call to `UIApplicationMain` creates two important initial components of your app:

- An instance of the `UIApplication` class, called the **application object**.

  The application object manages the app event loop and coordinates other high-level app behaviors. This class, defined in the UIKit framework, doesn't require you to write any additional code to get it to do its job.

- An instance of the `XYZAppDelegate` class, called the **app delegate**.

  Xcode created this class for you as part of setting up the Empty Application template. The app delegate creates the window where your app's content is drawn and provides a place to respond to state transitions within the app. This window is where you write your custom app-level code. Like all classes, the `XYZAppDelegate` class is defined in two source code files in your app: in the interface file, `XYZAppDelegate.h`, and in the implementation file, `XYZAppDelegate.m`.

Here's how the application object and app delegate interact. As your app starts up, the application object calls defined methods on the app delegate to give your custom code a chance to do its job—that's where the interesting behavior for an app is executed. To understand the role of the app delegate in more depth, view your app delegate source files, starting with the interface file. To view the app delegate interface file, select `XYZAppDelegate.h` in the project navigator. The app delegate interface contains a single property: `window`. With this property the app delegate keeps track of the window in which all of your app content is drawn.

Next, view the app delegate implementation file. To do this, select `XYZAppDelegate.m` in the project navigator. The app delegate implementation contains "skeletons" of important methods. These predefined methods allow the application object to talk to the app delegate. During a significant runtime event—for example, app launch, low-memory warnings, and app termination—the application object calls the corresponding method in the app delegate, giving it an opportunity to respond appropriately. You don't need to do anything special to make sure these methods get called at the correct time—the application object handles that part of the job for you.

Each of these automatically implemented methods has a default behavior. If you leave the skeleton implementation empty or delete it from your `XYZAppDelegate.m` file, you get the default behavior whenever that method is called. Use these skeletons to put additional custom code that you want to be executed when the methods are called. For example, the first method in the `XYZAppDelegate.m` file contains some lines of code that set up the app window and give it the white background color you saw when you ran your app for the first time. In this tutorial, you won't be using any custom app delegate code, so go ahead and remove the code that sets the window to have a plain white background.

To configure the app delegate implementation file

1. Find the `application:didFinishLaunchingWithOptions:` method in `XYZAppDelegate.m`.

   It is the first method in the file.

2. Delete the first three lines of code from that method so it looks just like this:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    return YES;
}
```

Xcode automatically saves the changes. Xcode continuously tracks and saves all of your work. (You can undo your changes by choosing Edit > Undo Typing.)

Create a Storyboard

At this point, you're ready to create a storyboard for your app. A **storyboard** is a visual representation of the app's user interface, showing screens of content and the transitions between them. You use storyboards to lay out the flow—or story—that drives your app.

To see how a storyboard fits into an app, in this tutorial you create one manually and add it to your app. Unlike the Empty Application template you started with, other Xcode templates include preconfigured storyboards providing views, view controllers, and associated source code files that set up the basic architecture for an app of that type. After you've configured a storyboard manually, you'll see how the pieces fit together. Then you'll be able to start with a project template that comes with a preconfigured storyboard, which will save you some overhead.

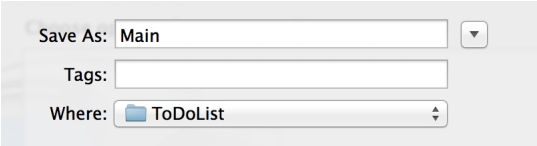
To create a new storyboard

1. Choose File > New > File (or press Command-N).

A dialog appears that prompts you to choose a template for your new file.

2. On the left, select User Interface under iOS.
3. Click Storyboard, and click Next.
4. For the Devices option, select iPhone.
5. Click Next.

   A dialog appears that prompts you to choose a location and name for your new storyboard.
6. In the Save As field, name the file `Main`.
7. Make sure the file is saved in the same directory as your project.



8. For the Group option, select ToDoList.
9. For Targets, select the checkbox next to ToDoList.

   This option tells Xcode to include the new storyboard when it builds your app.
10. Click Create.

A new storyboard file is created and added to your project. You'll work in this file to lay out the content of your app.

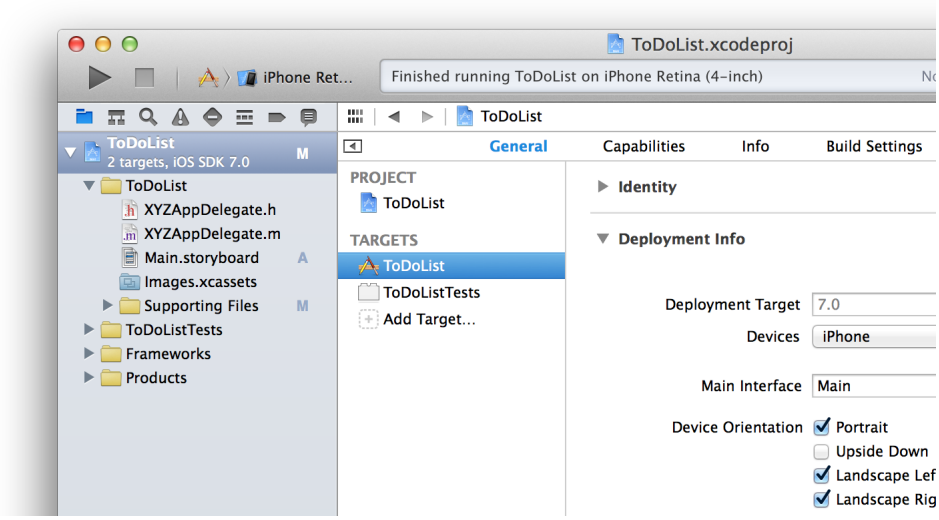
Now you need to tell Xcode that you want to use this storyboard as the interface to your app. When it starts up, the application object checks whether the app has a main interface configured. If it does, the application object loads the defined storyboard when the app launches.

To set the storyboard as the app's main interface

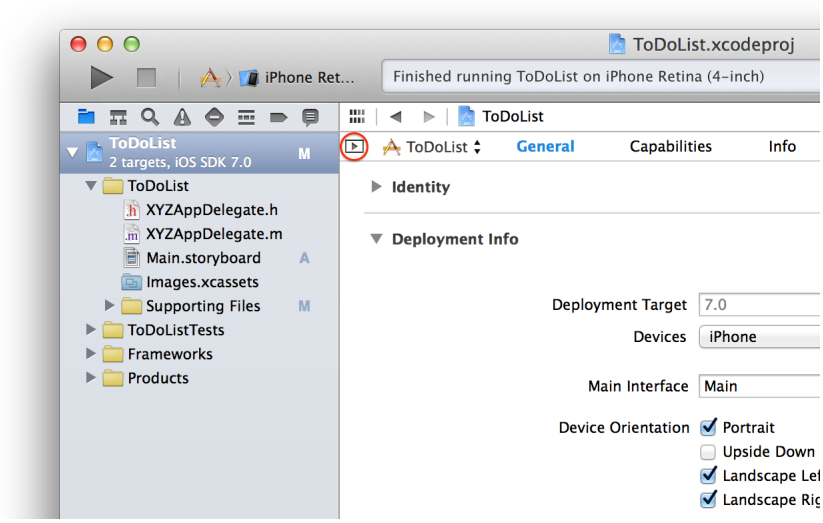
1. In the project navigator, select your project.

   In the editor area of the workspace window, Xcode displays the project editor, which allows you to view and edit details about how your app is built.

2. Under Targets, select ToDoList.



If the Project and Targets lists don't appear in the project editor, click the disclosure triangle in the top-left corner of the editor pane to reveal the lists.



3. Select the General tab.
4. Under Deployment Info, find the Main Interface option.
5. Select your storyboard, `Main.storyboard`.

Add a Scene to Your Storyboard

Now that you have a storyboard, it's time to start adding app content. Xcode provides a library of objects that you can add to a storyboard file. Some of these are user interface elements that belong in a view, such as buttons and text fields. Others define the behavior of your app but don't themselves appear onscreen, such as view controllers and gesture recognizers.

To start, you'll add a view controller to your storyboard. A view controller manages a corresponding view and its subviews. You'll learn more about the roles of views and view controllers in the next chapter, "App Development Process" (page 30).

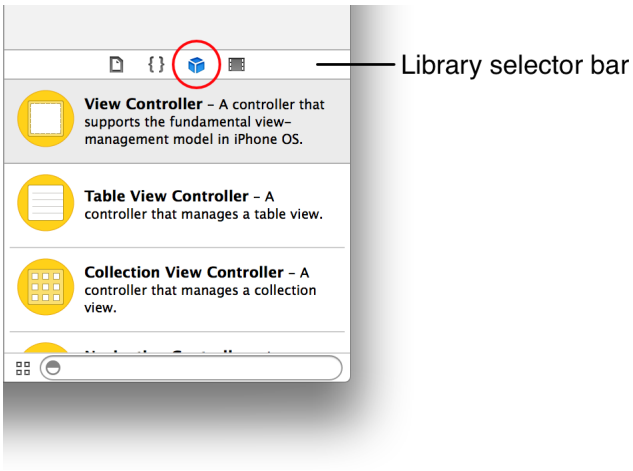
To add a view controller to your storyboard

1. In the project navigator, select `Main.storyboard`.

   Xcode opens the storyboard in **Interface Builder**—its visual interface editor—in the editor area. Because the storyboard is empty, what you see is a blank **canvas**. You use the canvas to add and arrange user interface elements.

2. Open the Object library.

   The **Object library** appears at the bottom of the utility area. If you don't see the Object library, you can click its button, which is the third button from the left in the library selector bar. (If you don't see the utility area, you can display it by choosing View > Utilities > Show Utilities.)



   A list appears showing each object's name, description, and visual representation.

3. Drag a View Controller object from the list to the canvas.

   If you can't find the object titled View Controller in the Object library, filter the list of objects by typing in the text field below the list. Type `View Controller`, and you see only view controller objects in the filtered list.

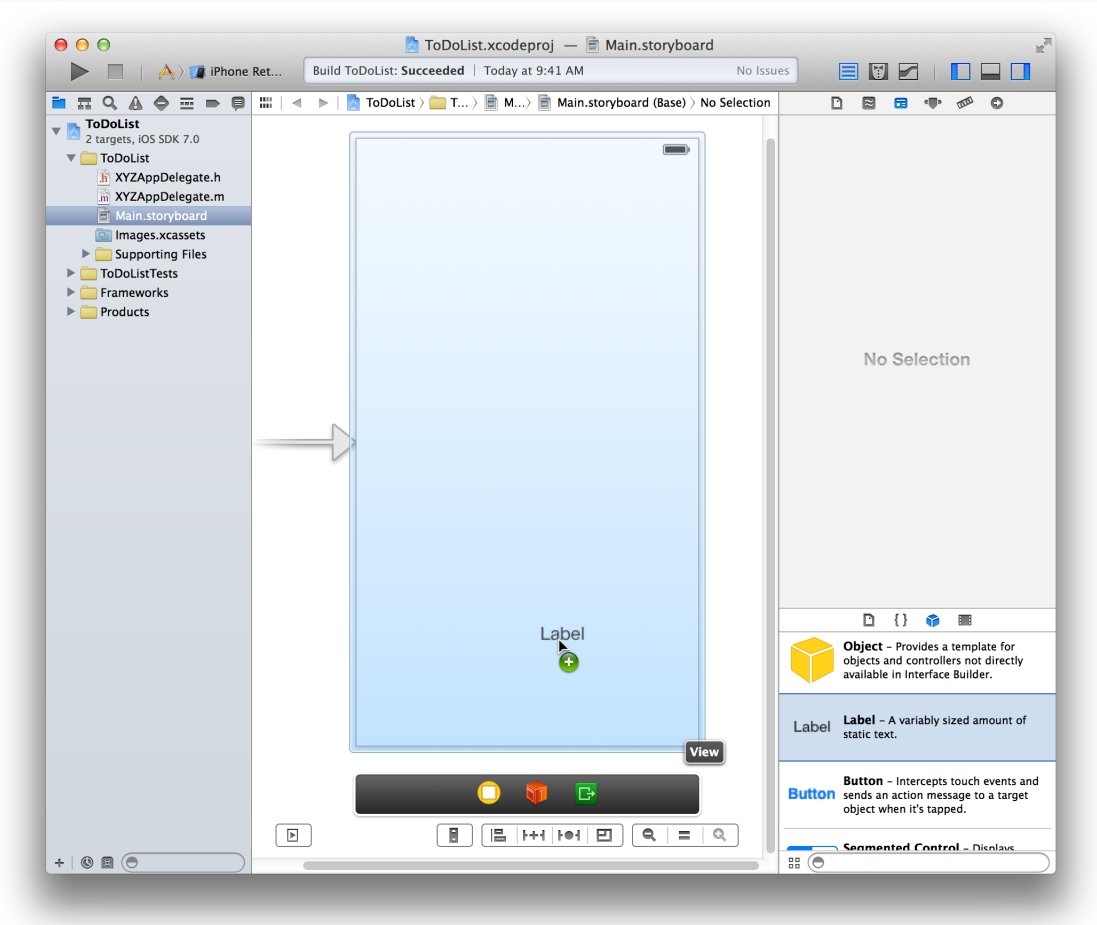
At this point, the storyboard in your app contains one **scene**. The arrow that points to the left side of the scene on the canvas is the **initial scene indicator**, which means that this scene is loaded first when the app starts. Right now, the scene that you see on the canvas contains a single view that's managed by a view controller. If you run your app in iOS Simulator, this view is what you see on the device screen. It's useful to run your app in iOS Simulator to verify that everything is configured correctly. Before doing that, add something to the scene that you'll be able to see when you run the app.

To add a label to your scene

1. In the Object library, find the Label object.

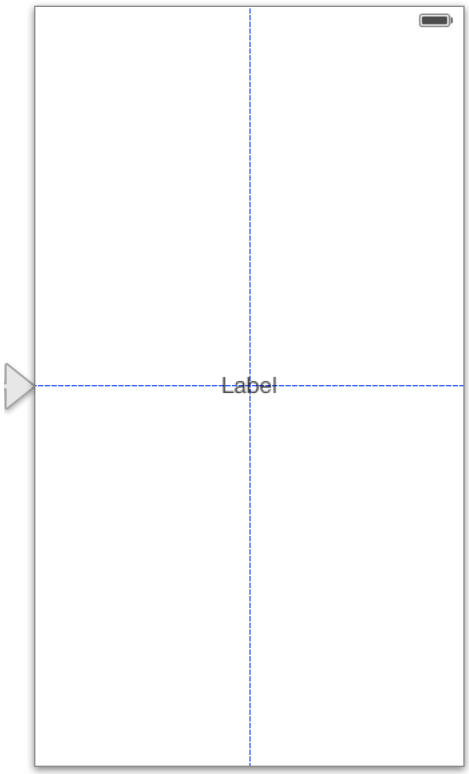
   If you entered text in the filter text field, you may need to clear its contents before you can see the Label object. You can also type `Label` in the filter field to find the Label object quickly.

2. Drag a Label object from the list to your scene.



3. Drag the label to the center of the scene until horizontal and vertical guides appear.

Stop dragging the label when you see something like this:



The guides mean that the label is now centered horizontally and vertically. (The guides are visible only when you're dragging or resizing objects next to them, so they will disappear when you let go of the label.)

4. Double-click the text of the label to select it for editing.
5. Type `Hello, World!` and press Return.

   If necessary, recenter the label.

Test Your Changes

Running your app in iOS Simulator is a great way to periodically check that everything is working the way you expect. At this point your app should launch and load the scene you created in your main storyboard. Click the Run button in Xcode. You should see something like this:

If you don't see the label you added, make sure the storyboard you created is configured as the main interface for your app and make sure you've removed the code that creates the empty white window in the app delegate. If necessary, go back and repeat the steps for those sections.

This is also a good time to experiment with what you can add to an interface. Explore Interface Builder by changing:

- The text of the label
- The font size of the label
- The color of the text

Build the Basic Interface

Now that you can put content in a scene, it's time to build the basic interface for the scene that lets you add a new item to the to-do list.

To add an item to the to-do list, you need a single piece of information: the item name. You get this information from a text field. A text field is the interface element that lets a user input a single line of text using a keyboard. But first, you need to remove the label you added earlier.

To remove the label from your scene

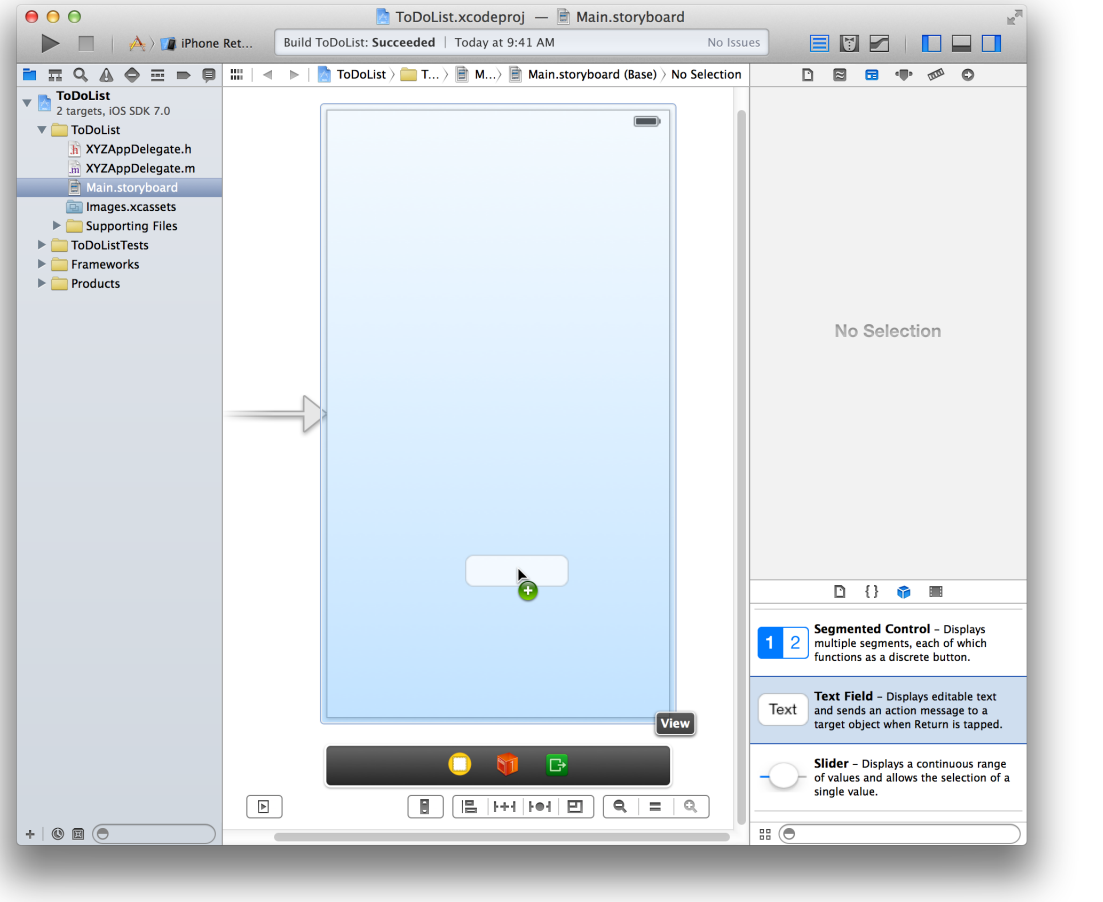
1. Click the label to select it.
2. Press the Delete key.

   The label is removed from the scene. If this wasn't what you wanted, you can choose Edit > Undo Delete Label. (Every editor has an Edit > Undo command to undo the last action.)

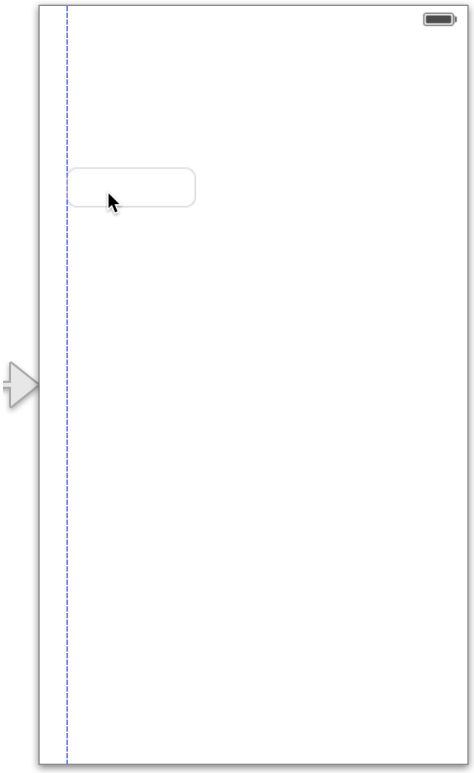
Now that you again have a blank canvas, create the scene for adding a to-do item.

To add a text field to your scene

1. If necessary, open the Object library.
2. Drag a Text Field object from the list to your scene.

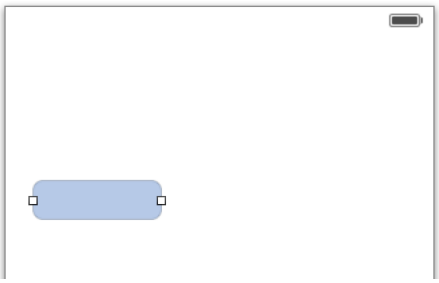


3. Drag the text field so that it's positioned about two-thirds from the bottom of the screen.



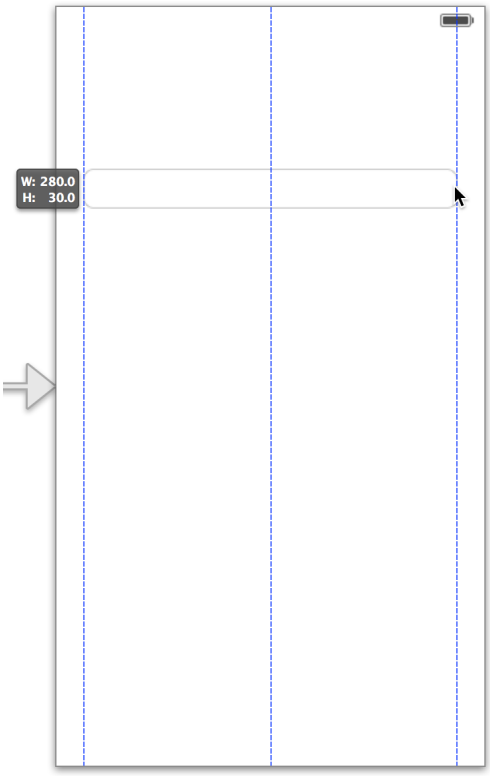
4. If necessary, click the text field to reveal the resize handles.

   You resize a UI element by dragging its **resize handles**, which are small white squares that appear on the element's borders. You reveal an element's resize handles by selecting it. In this case, the text field should already be selected because you just stopped dragging it. If your text field looks like the one below, you're ready to resize it; if it doesn't, select it on the canvas.



5. Resize the left and right edges of the text field until you see vertical guides appear.

Stop resizing the text field when you see something like this:

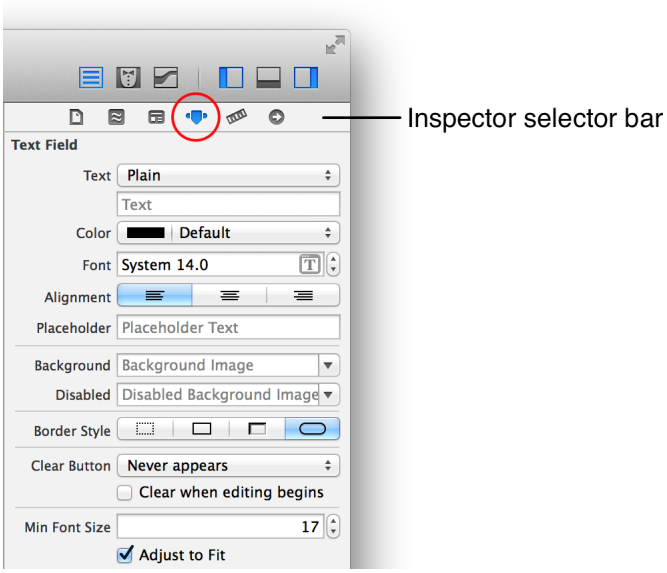


Although you have the text field in your scene, there's no instruction to the user about what to enter in the field. Use the text field's placeholder text to prompt the user to enter the name of a new to-do item.

To configure the text field's placeholder text

1. With the text field selected, open the Attributes inspector in the utility area.

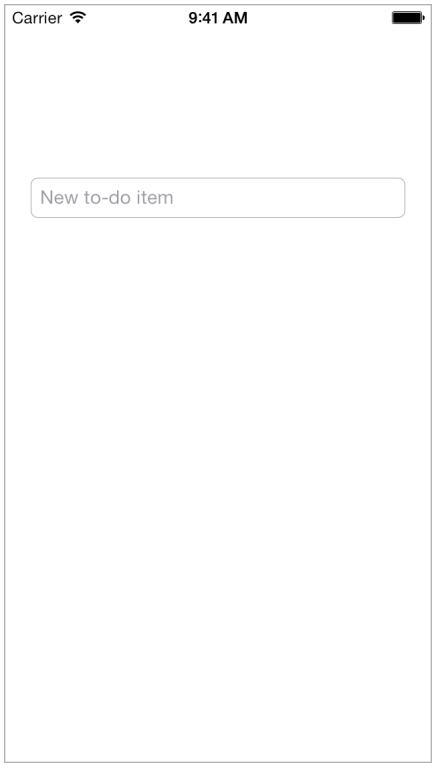
The **Attributes inspector** appears when you select the fourth button from the left in the inspector selector bar. It lets you edit the properties of an object in your storyboard.



2. In the Attributes inspector, find the field labeled Placeholder and type `New to-do item`.

   To display the new placeholder text in the text field, press Return.

Checkpoint: Run your app in iOS Simulator to make sure that the scene you created looks the way you expect it to. You should be able to click inside the text field and enter a string using the keyboard.



Recap

You're now well on your way to being able to create a basic interface using storyboards. In the remaining tutorials, you'll learn more about adding interaction to your interface and writing code to create custom behavior. The chapters between the tutorials guide you through the concepts that you'll put into practice while working on your app.