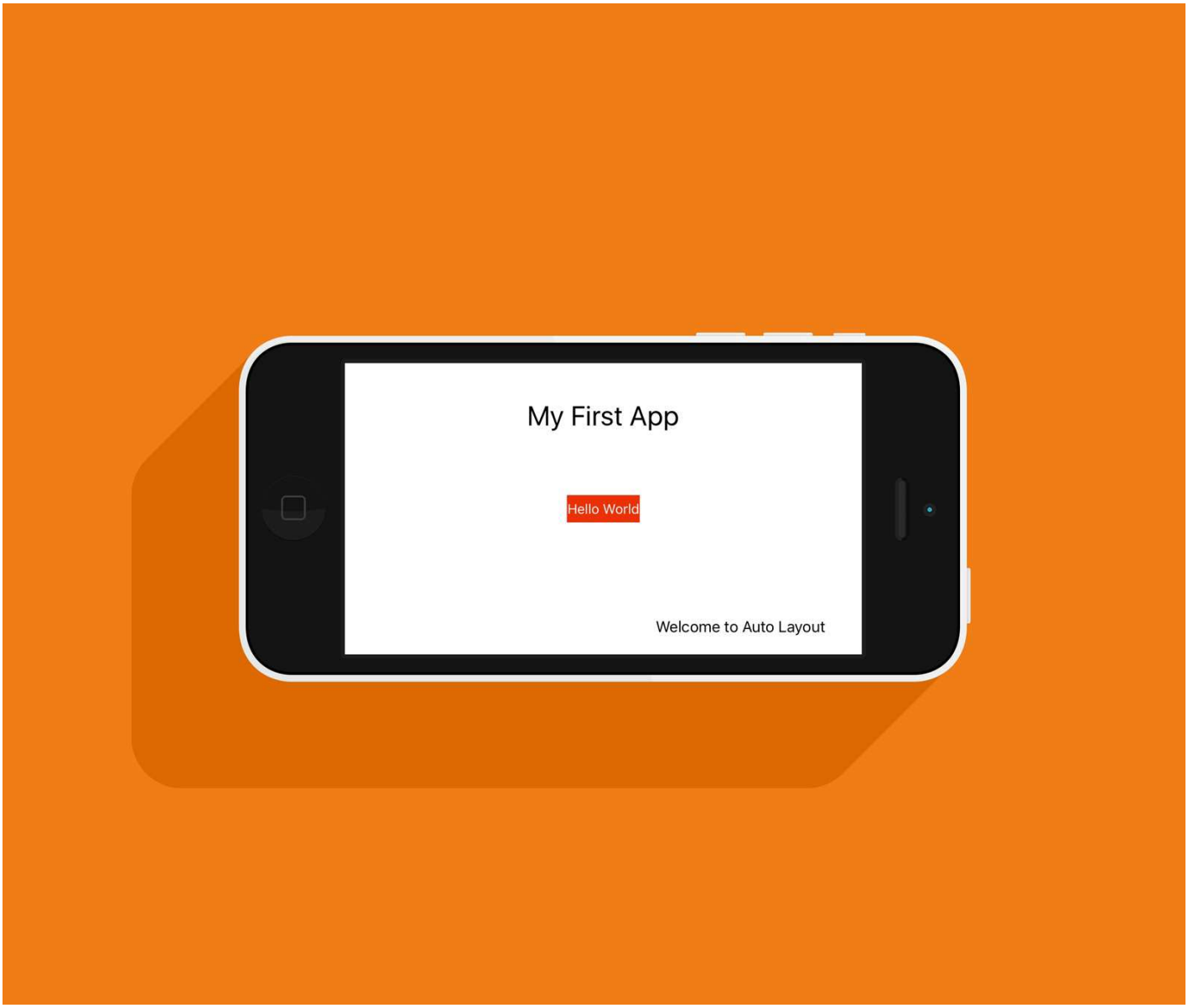
Chapter 5
Introduction to Auto Layout



> Life is short. Build stuff that matters. – Siqi Chen

Wasn't it fun to create the Hello World app? Before we move onto building a real app, we'll take a look at Auto Layout in this chapter.

Auto layout is a constraint-based layout system. It allows developers to create an adaptive UI that responds appropriately to changes in screen size and device orientation. Some beginners find it hard to learn and avoid using it. But believe me, you won't be able to live without it when developing an app today.

With the release of iPhone 6 and 6 Plus, Apple's iPhones are available in different screen sizes including 3.5-inch, 4-inch, 4.7-inch and 5.5-inch displays. Without using auto layout, it would be very hard for you to create an app that supports all screen resolutions. And, starting from Xcode 6, it is inevitable to use auto layout to design the user interface. This is why I want to teach you auto layout at the very beginning of this book, rather than jumping right into coding a real app. In this chapter and the one that follows, I want to help you build a solid foundation on designing an adaptive user interface.

Quick note: Auto layout is not as difficult as some developers thought. Once you understand the basics, you will be able to use auto layout to create complex user interfaces for all types of iOS devices.

Why Auto Layout?

Let me give you an example, and you'll have a better idea why auto layout is needed. Open the HelloWorld project you built in chapter 3. Instead of running the app on iPhone 5s simulator, run it using the iPhone 6 simulator (or iPhone 4s). You'll end up with the results illustrated in figure 5-1. It turns out that the button isn't centered when running on iPhone 6.

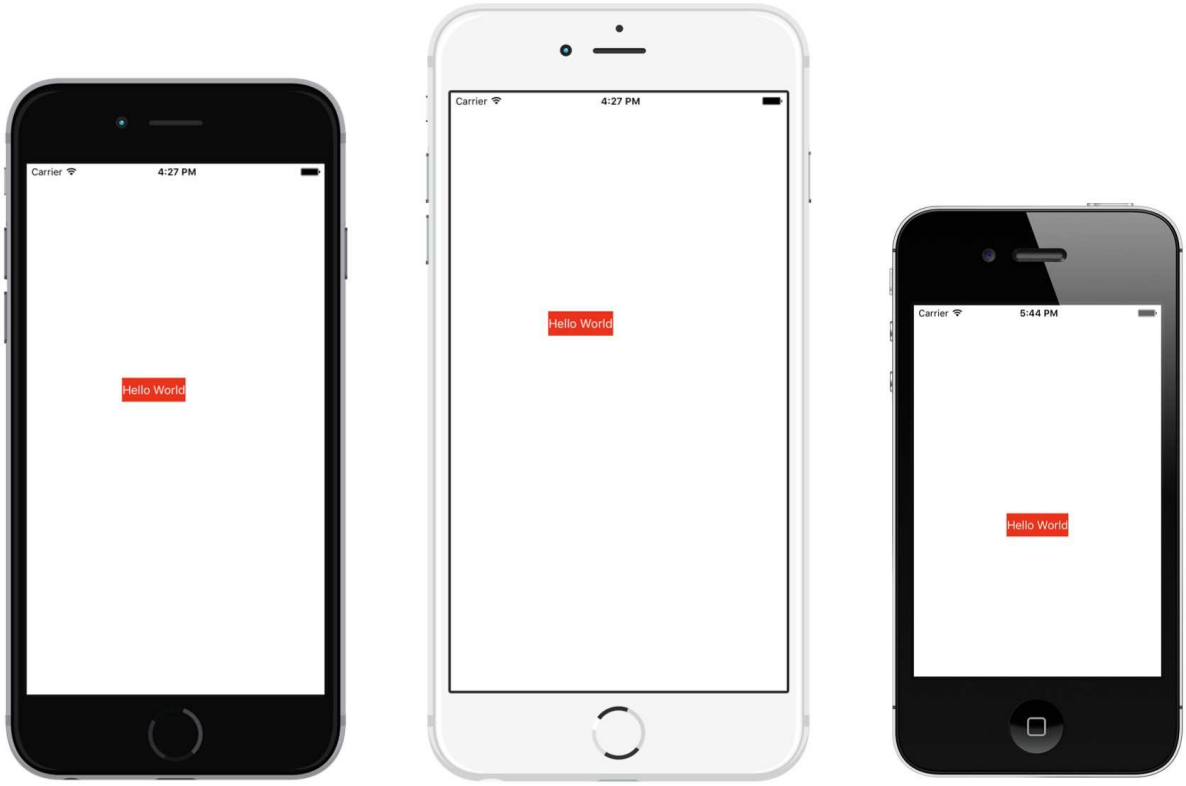


Figure 5-1. The app UI looks different when running on iPhone 4s, iPhone 6 (4.7-inch) and iPhone 6 Plus (5.5-inch)

Let's try one more thing.

Click the Stop button and run the app using the iPhone 5s simulator. After the simulator launches, go up to the menu and select Hardware > Rotate Left (or Rotate Right) from the menu. This rotates the device to landscape mode. Alternatively, you can press command+left arrow/right arrow to rotate the device sideway. Again, the Hello World button is not centered.

Why? What's wrong with it?

As you know, iPhone 5s, iPhone 6 and iPhone 6 Plus have different screen dimensions. For iPhone 5/5s, the screen in portrait mode consists of 320 points (or 640 pixels) horizontally and 568 points (or 1136 pixels) vertically. For iPhone 4s, the screen consists of 320 points (or 640 pixels) and 480 points (or 960 pixels). For iPhone 6, the screen consists of 375 points (or 750 pixels) horizontally and 667 points (or 1334 pixels) vertically.

Why Points instead of Pixels?

Back in 2007, Apple introduced the original iPhone with a 3.5-inch display with a resolution of 320x480. That is 320 pixels horizontally and 480 pixels vertically. Apple retained this screen resolution with the succeeding iPhone 3G and iPhone 3GS. Obviously, if you were building an app at that time, one point corresponds to one pixel. Later, Apple introduced iPhone 4 with retina display. The screen resolution was doubled to 640x960 pixels. So one point corresponds to two pixels for retina display.

The point system makes our developers' lives easier. No matter how the screen resolution is changed (say, the resolution is doubled again to 1280x1920 pixels), we still deal with with points and the base resolution (i.e. 320x480 for iPhone 4/4s or 320x568 for iPhone 5/5s). The translation between points and pixels is handled by iOS.

Without using auto layout, the position of the button we lay out in the storyboard is fixed. In other words, we "hard-code" the frame origin of the button. In our example, the "Hello World" button's frame origin is set to (120, 269). Therefore, whether you're using a 3.5-inch or 4-inch or 4.7-inch simulator, iOS draws the button in the specified position. Figure 5-2 illustrates the frame origin on different devices. This explains why the "Hello World" button can only be centered on iPhone 5/5s, and it is shifted away from the screen center on other iOS devices, as well as, in landscape orientation.

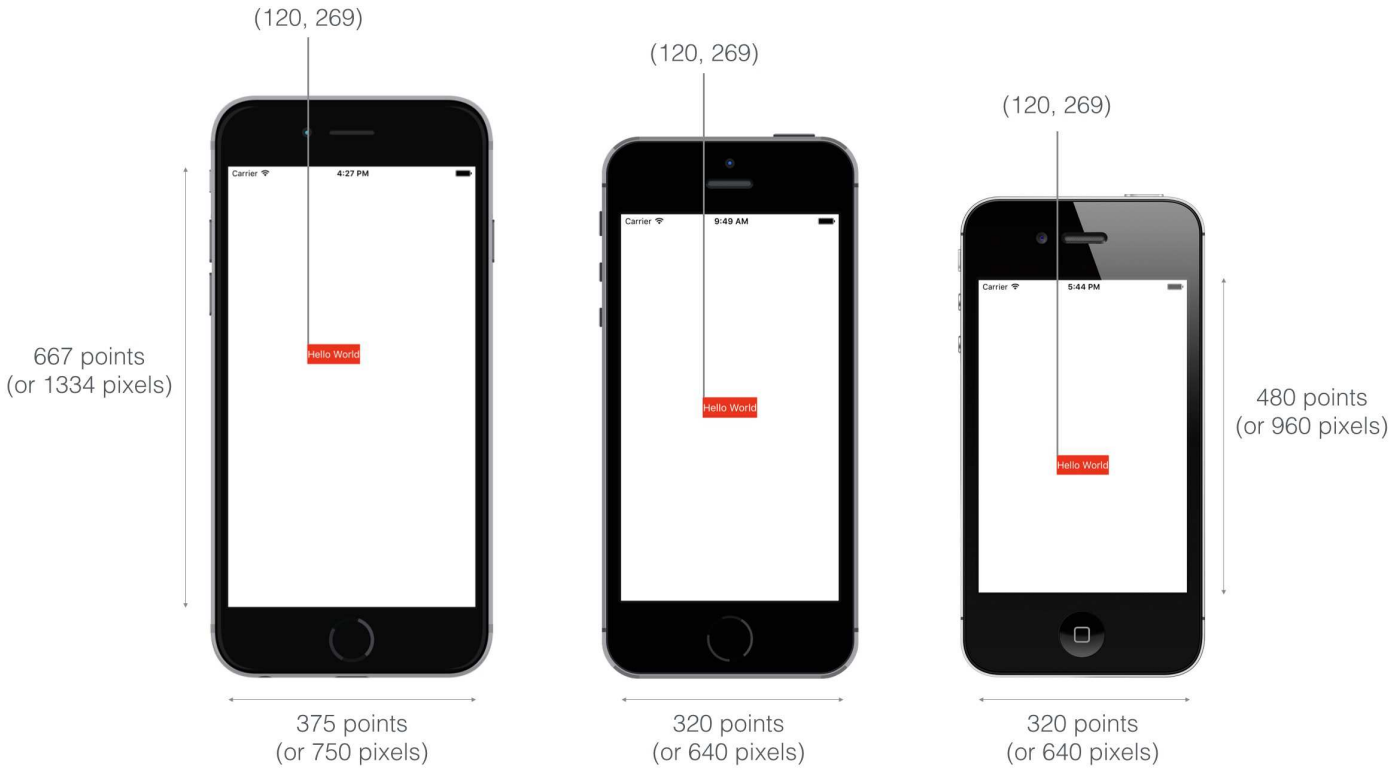


Figure 5-2. Screen dimensions for iPhone 6, iPhone 5/5s and 4s

Obviously, we want the app to look good on all iPhone models, and in both portrait & landscape orientation. This is why we have to learn auto layout. It's the answer to the layout issues, that we have just talked about.

Auto Layout is All About Constraints

As mentioned before, auto layout is a constraint-based layout system. It allows developers to create an adaptive UI that responds appropriately to changes in screen size and device orientation. Okay, it sounds good. But what does the term "constraint-based layout" mean? Let me put it in a more descriptive way. Consider the "Hello World" button again, how do you describe its position if you want to place the button at the center of the view? You would probably describe it like this:

The button should be centered both horizontally and vertically, regardless of the screen resolution and orientation.

Here you actually define two constraints:

- center horizontally
- center vertically

These constraints express rules for the layout of the button in the interface.

Auto layout is all about constraints. While we describe the constraints in words, the constraints in auto layout are expressed in mathematical form. For example, if you're defining the position of a button, you might want to say "the left edge should be 30 points from the left edge of its containing view." This translates to `button.left = (container.left + 30)`.

Fortunately, we do not need to deal with the formulas. All you need to know is how to express the constraints descriptively and use Interface Builder to create them.

Okay, that's quite enough for the auto layout theory. Now let's see how to define layout constraints in Interface Builder to center the "Hello World" button.

Reactivating Size Classes

First, open `Main.storyboard` of your HelloWorld project (or download it from https://www.dropbox.com/s/uiy4b31da10k58c/HelloWorld.zip). Before we add the layout constraints to the user interface, let's re-enable *Size Classes*. This is the option we have deactivated when we built the first app. Under the File inspector, check the "Use Size Classes" option to re-enable it. When prompted, click "Enable Size Classes" to confirm the change.

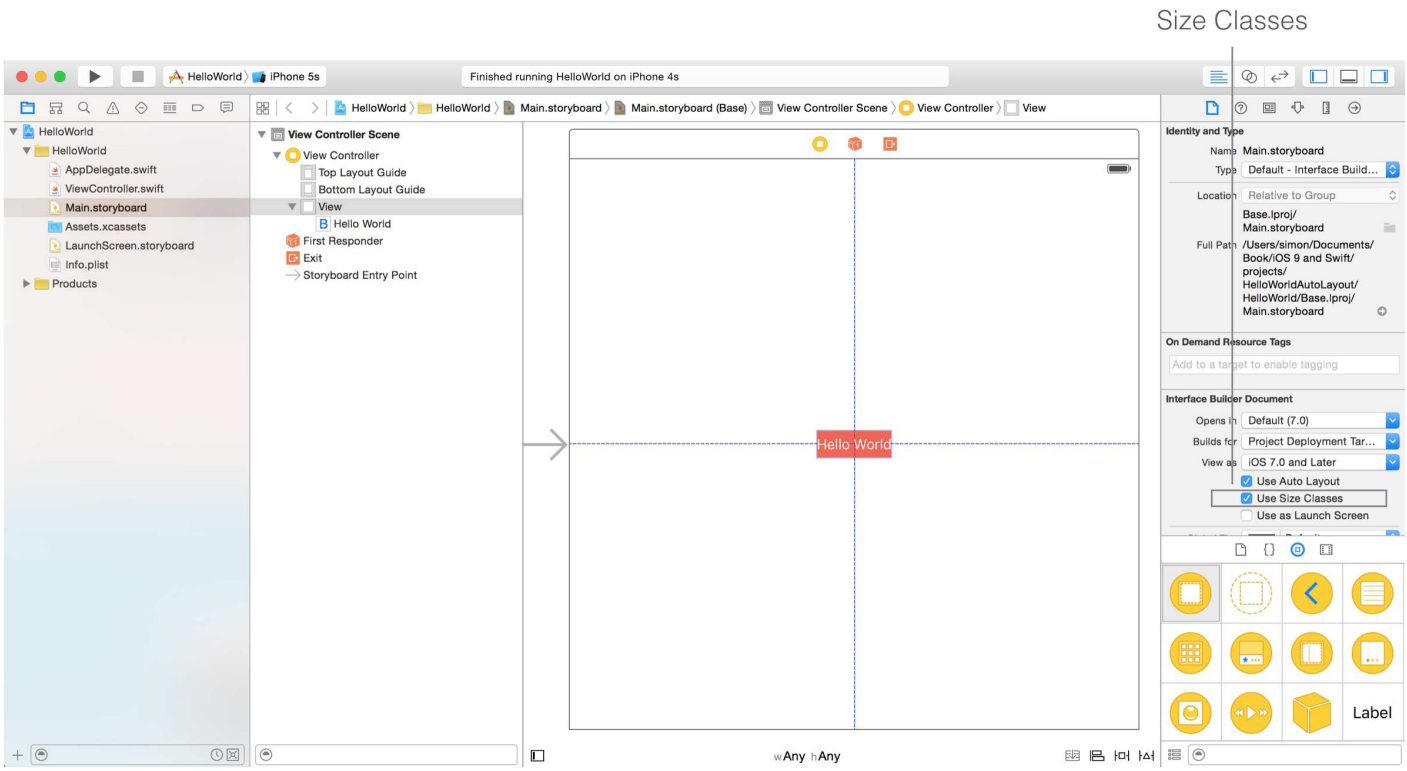


Figure 5-3. Freeform canvas

The view controller is restored to a *freeform* canvas. The button is now shifted to the left. You can drag it to the center of the view as that shown in the figure.

Why do we need to make this change? While you can still define layout constraints without using size classes, I want you to start designing the UI using the standard canvas. It will take some time to get used to the freeform canvas, but you're required to use it for designing adaptive UI that fits all screen sizes. So let's start using it.

```
Quick note: You may wonder what the term "size classes" means. Just forget
about it right now and focus on learning auto layout. I will discuss about it
in the later chapter.
```

Using Auto Layout to Center the Button

Xcode provides two ways to define auto layout constraints:

1. Auto layout bar

2. Control-drag

We'll demonstrate both approaches in this chapter. Let's begin with the auto layout bar. At the bottom-right corner of the Interface Builder editor, you should find a few buttons. These buttons are from the layout bar. You can use them to define various types of layout constraints.

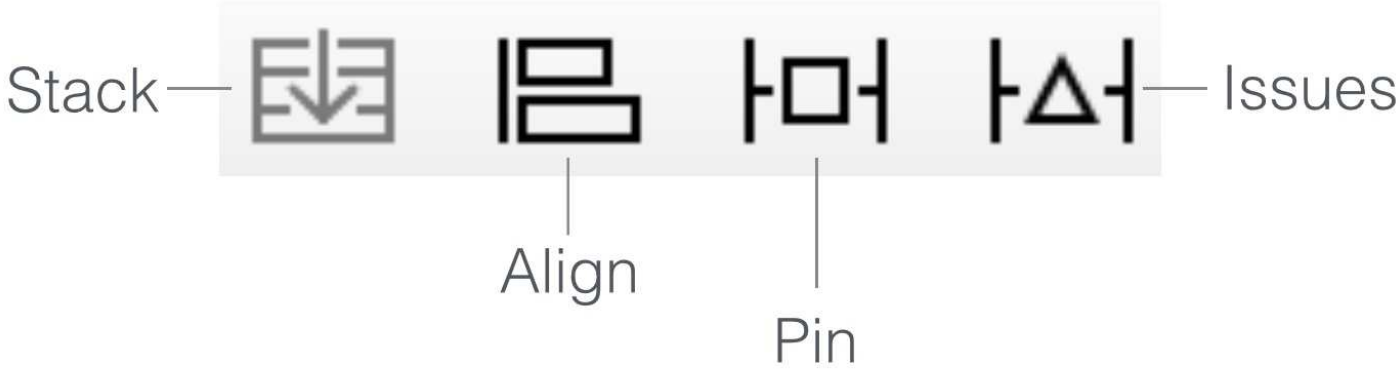


Figure 5-4. Auto layout menu

Each button has its own function:

- Align – Create alignment constraints, such as aligning the left edges of two views.
- Pin – Create spacing constraints, such as defining the width of a UI control.
- Issues – Resolve layout issues.
- Stack – Embed views into a stack view. Stack view is a new feature in Xcode 7. We will further discuss about it in the next chapter.

As discussed earlier, to center the "Hello World" button, you have to define two constraints: *center horizontally* and *center vertically*. Both constraints are with respect to the view.

To create the constraints, we will use the Align function. First, select the button in Interface Builder and then click the *Align* icon in the layout bar. In the pop-over menu, check both "Vertical center in container" and "Horizontal center in container" options. Then click the "Add 2 Constraints" button.

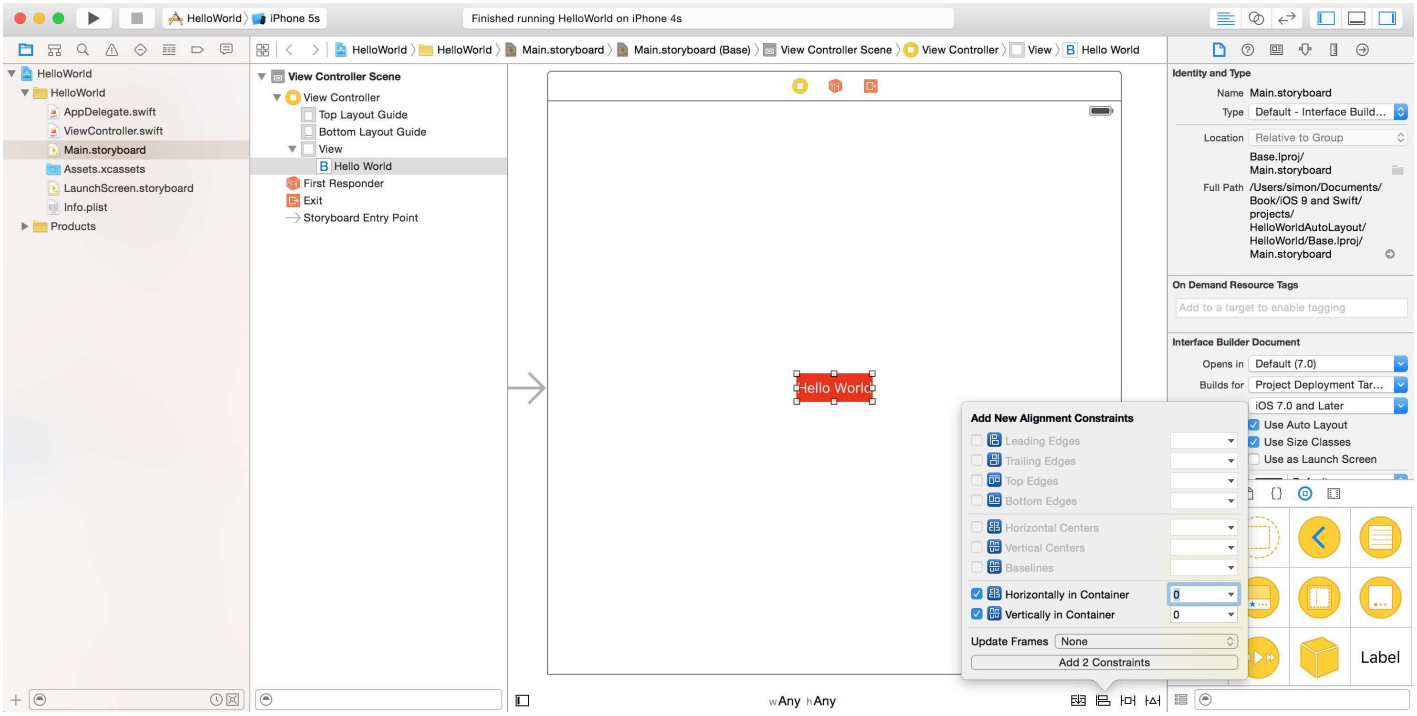


Figure 5-5. Adding constraints using Align button

You should now see a set of constraint lines. If you expand the *Constraints* option in the document outline view, you should find two new constraints for the button. These constraints ensure the button is always positioned at the center of the view. Alternatively, you can view these constraints in the Size inspector.

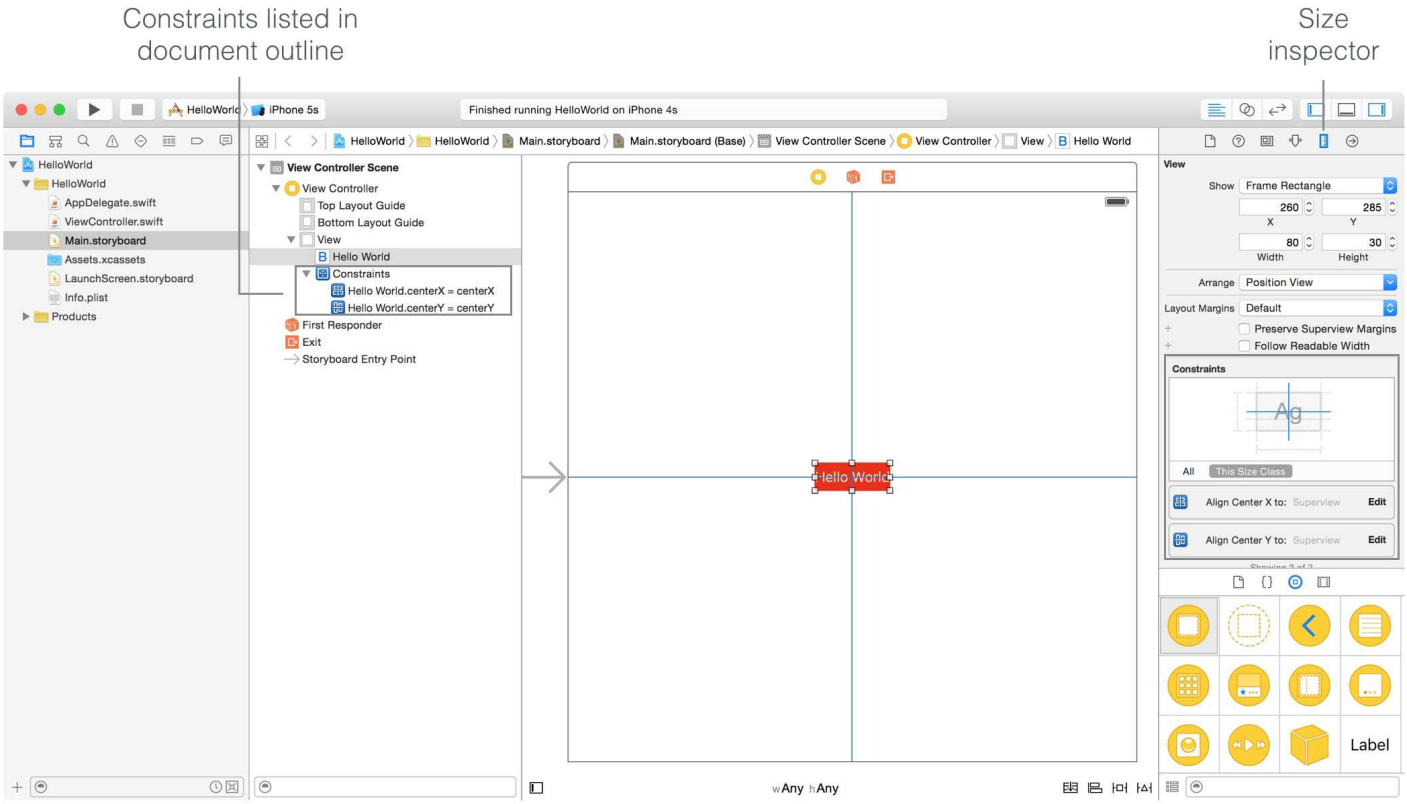


Figure 5-6. View the constraints in Document Outline and Size Inspector

Quick note: When your view layout is being configured correctly and there is no ambiguity, the constraint lines are in blue.

Okay, you're ready to test the app. You can click the Run button to launch the app on iPhone 6/6 Plus (or iPhone 4s), the button should be centered perfectly.

Resolving Layout Constraint Issues

The layout constraints that we have just set are perfect. But that is not always the case. Xcode is intelligent enough to detect any constraint issues. Try to drag the Hello World button to the lower-left part of the screen. Xcode immediately detects some layout issues and the corresponding constraint lines turns orange that indicates a misplaced item.

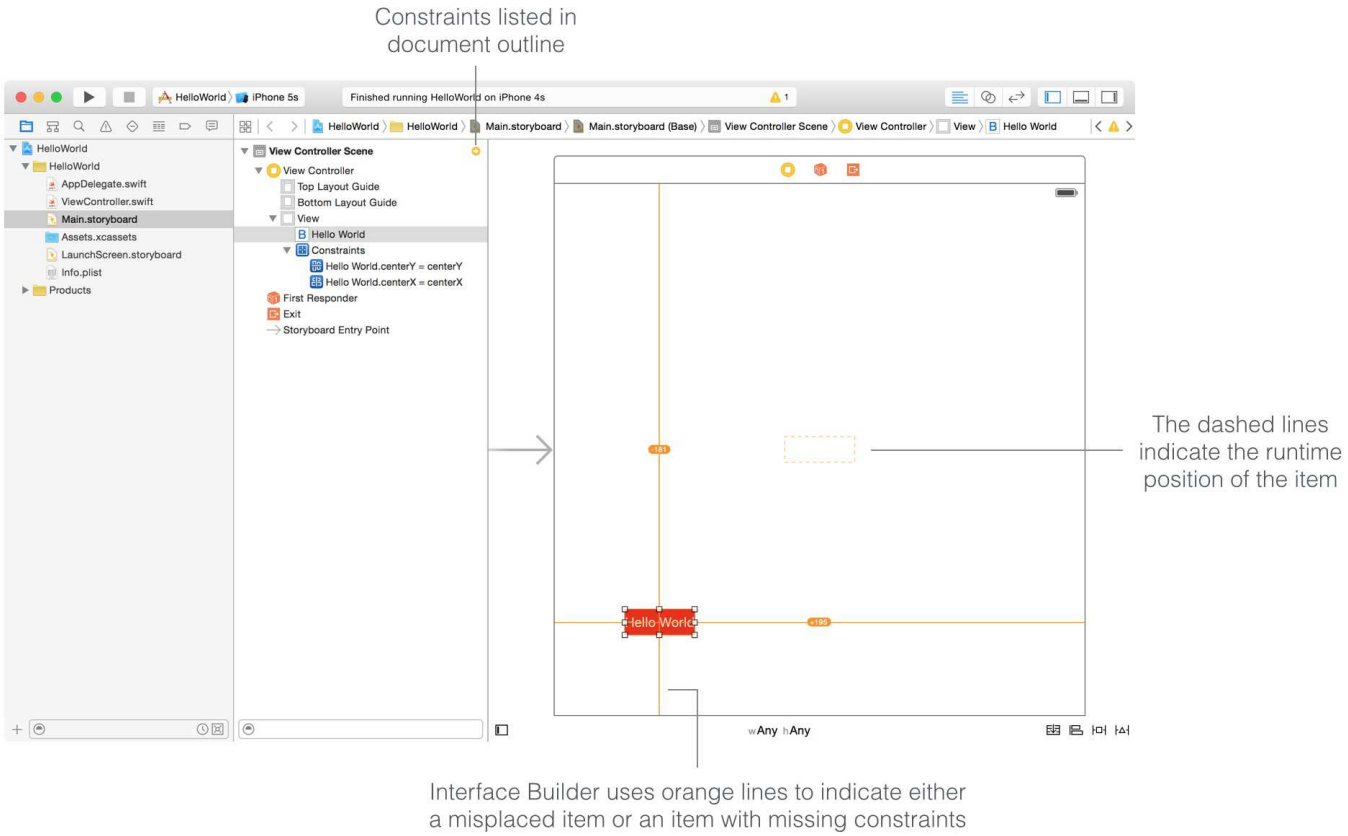


Figure 5-7. Interface Builder uses orange/red lines to indicate Auto Layout issues

Auto layout issues occur when you create ambiguous or conflicting constraints. Here we said the button should be vertically and horizontally centered in the container (i.e. the view). However, the button is now placed at the lower-left corner of the view. Interface Builder found this confusing, therefore it uses orange lines to indicate the layout issues.

When there is any layout issue, the Document Outline view displays a disclosure arrow (red/orange). Click the disclosure arrow to see a list of the issues. Interface Builder is smart enough to resolve the layout issues for us. Click the indicator icon next to the issue and a pop-over shows you a number of solutions. In this case, select the "Update Frame" option and click "Fix Misplacement" button. The button will then be moved to the center of the view.

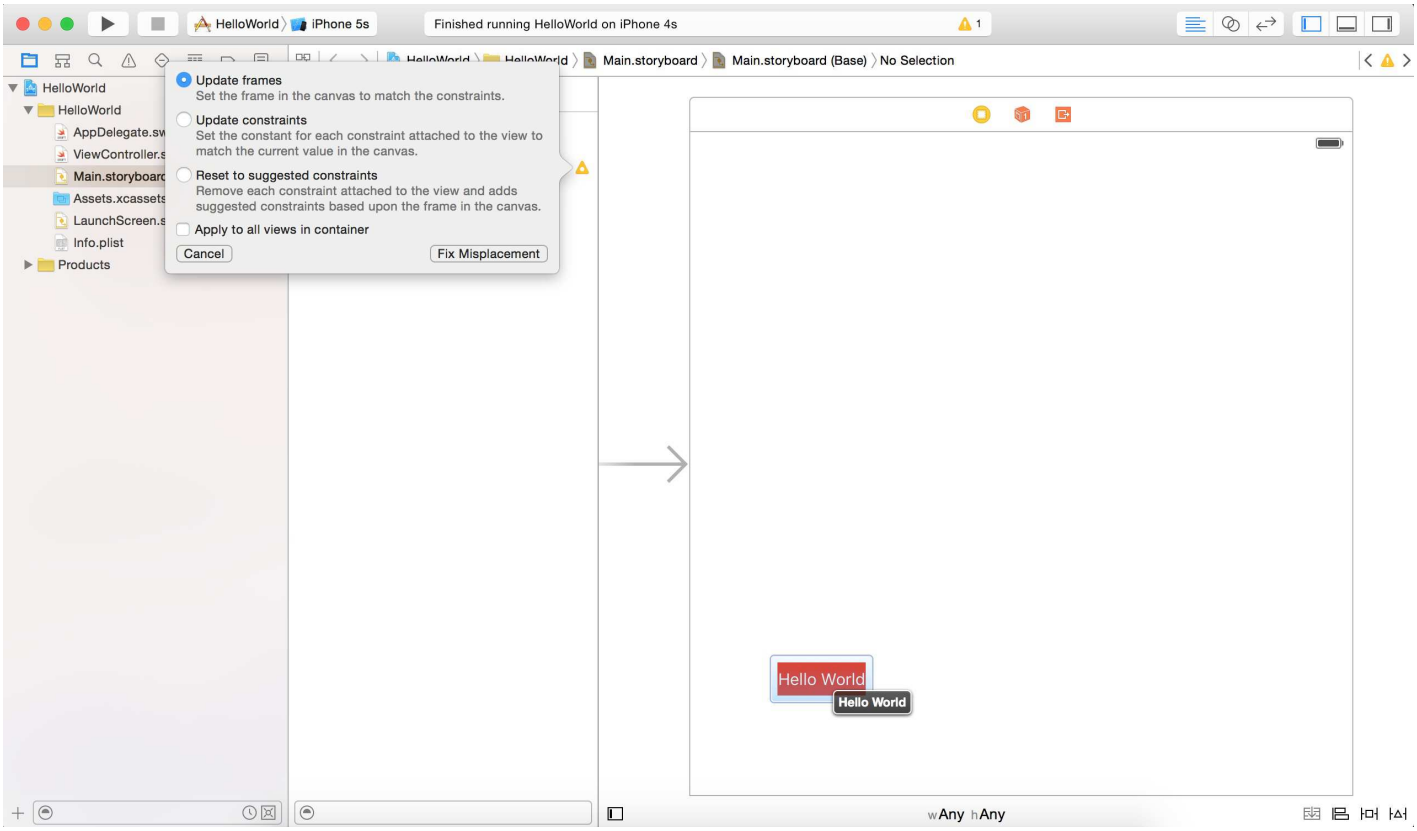


Figure 5-8. Resolving misplacement issue

This layout issue was triggered manually. I just wanted to demonstrate how to find the issues and fix them. As you go through the exercises in the later chapters, you will probably face a similar layout issue. You should know how to resolve layout issues easily and quickly.

Previewing Storyboards

So far we have only used the simulators to test the UI changes. While the built-in simulator is great, it's not the only way to experiment with your app's UI. Xcode 7 provides a Preview feature for developers to preview the user interface on different devices, right in the Interface Builder editor.

```
Quick tip: If you're using Xcode on a laptop, go up to the Xcode menu and
choose View > Navigators > Hide Navigator (or simply press command-0 key.) This
will hide the project navigator to give you more screen space for editing your
storyboard.
```

In Interface Builder, open the Assistant pop-up menu > Preview (1). Now press and hold `option` key, then click Main.storyboard (Preview). You can refer to figure 5-9 for the steps.

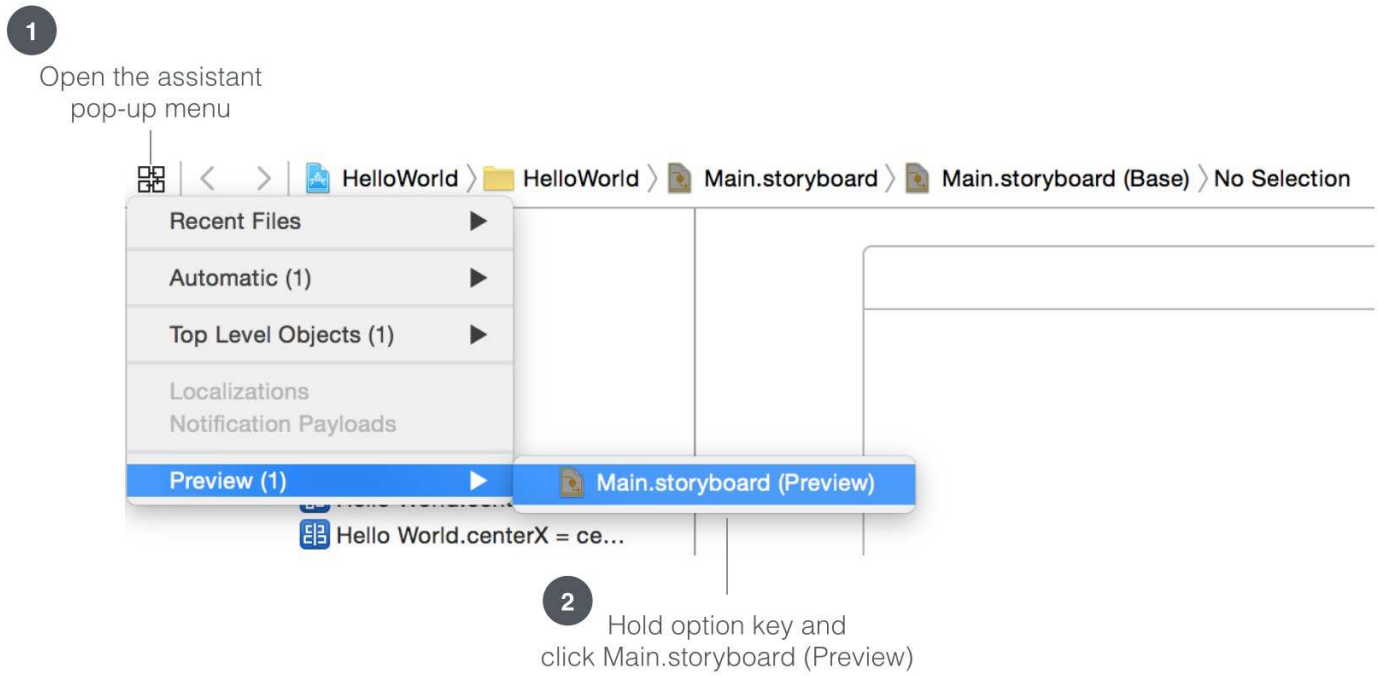


Figure 5-9. Assistant Pop-up Menu

Xcode will display a preview of your app's UI in the assistant editor. By default, it shows you the preview on a iPhone 4-inch device. You can click the + button at the lower-left corner of the assistant editor to add other iOS devices (e.g. iPhone 4.7-inch) for preview. If you want to see how the screen looks like in landscape orientation, simply click the rotate button. The Preview feature is extremely useful for designing your app's user interface. You can make changes to the storyboard (say, adding another button to the view) and simultaneously see how those changes look in the preview. This will save you from loading the app in the simulator time and time again, especially if you just want to test a simple UI change.

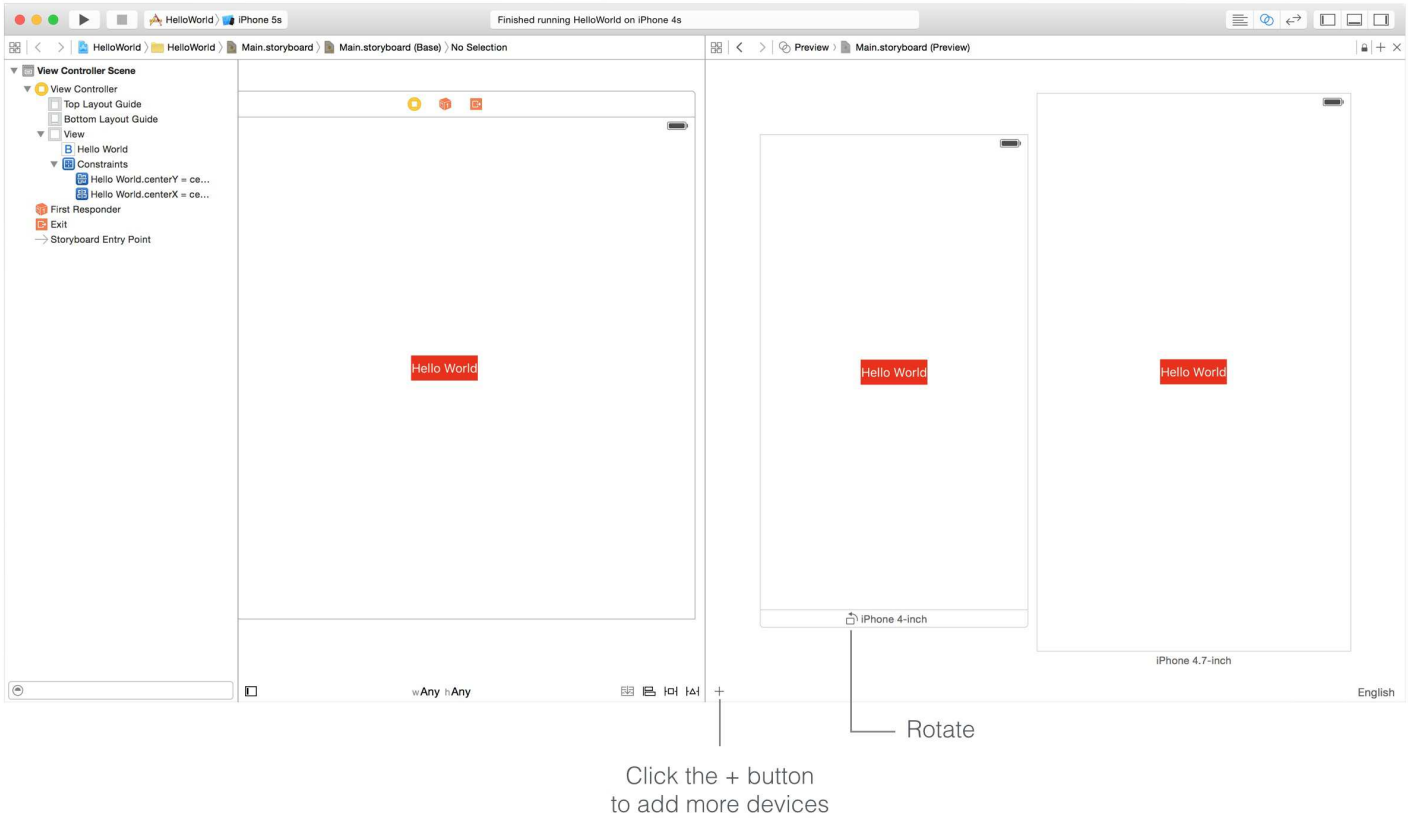


Figure 5-10. Previewing storyboard in the Assistant Editor

Quick tip: When you add more devices in the preview assistant, Xcode may not be able fit the preview of all devices sizes into the screen at the same time. If you're using a trackpad, you can scroll through the preview by swiping left or right with two fingers. What if you're still using a mouse with a scroll wheel? Simply hold the shift key to scroll horizontally.

Adding a Label

Now that you have some idea about auto layout and the preview feature, let's add a label to the lower-right part of the view and see how to define the layout constraints for the label. When designing your UI, you should always remember that you have to make it fit for all screen sizes.

In the Interface Builder editor, drag a label from the Object library and place it near the lower-right corner of the view. Double-click the label and change it to "Welcome to Auto Layout" or whatever title you want.

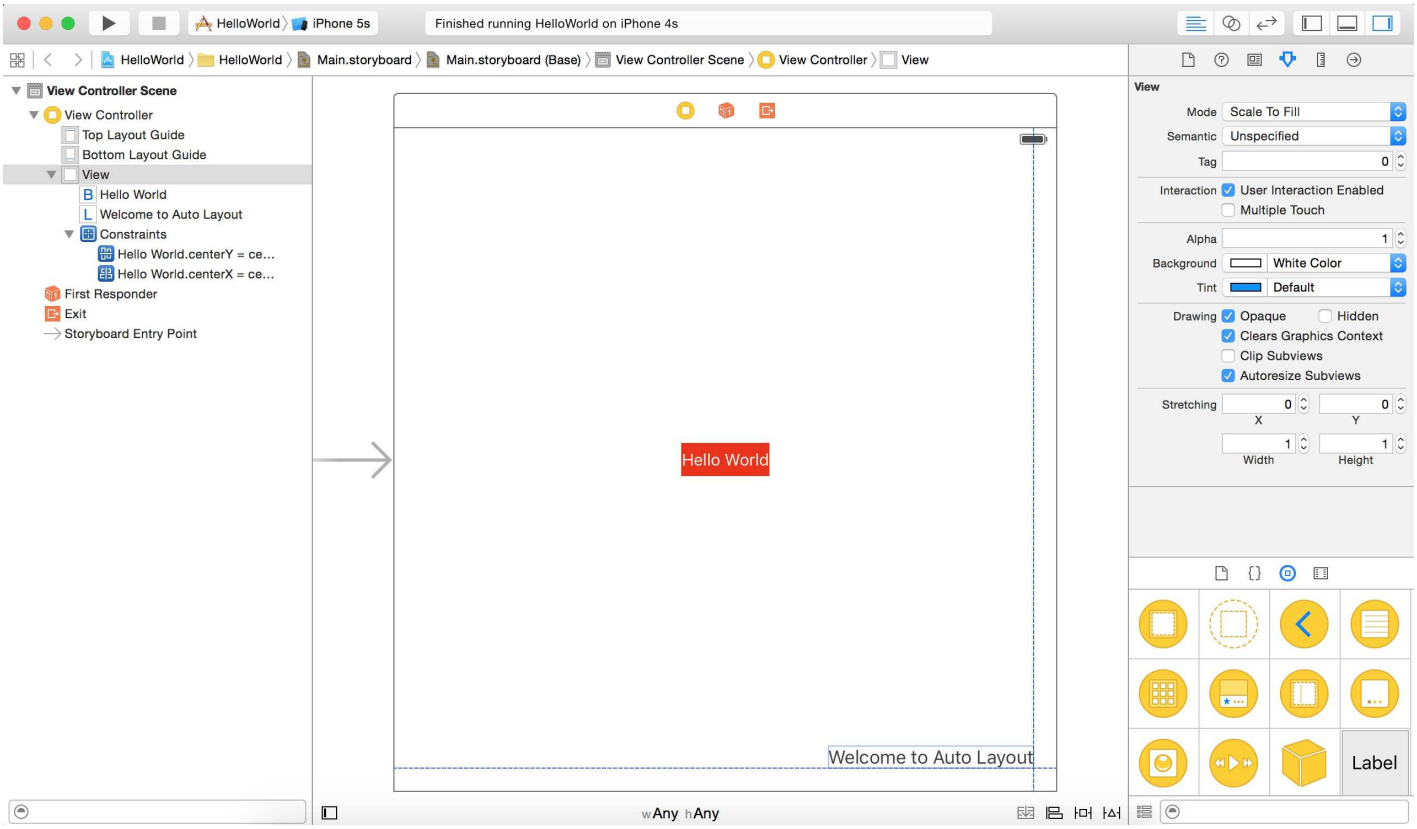


Figure 5-11. Adding a label to the view

If you opened the preview assistant again, you should see the UI change immediately (see figure 5-12). Without defining any layout constraints for the label, you are not able to display the label on all iPhone devices.

Figure 5-12. The label can't be displayed properly

How can you resolve this issue? Obviously, we need to setup a couple of constraints to make it work properly. The question is: *what constraints should we add?*

Let's try to describe the requirement of the label in words. You probably describe it like this:

The label should be placed at the lower-right corner of the view.

That's okay, but not precise enough. A more precise way to describe the location of the label is like this:

The label is located 0 points away from the right margin of the view and 20 points away from the bottom of the view.

This is much better. When you describe the position of an item precisely, you can easily come up with the layout constraints. Here, the constraints of the label are:

1. The label is 0 points away from the right margin of the view.

2. The label is 20 points away from the bottom of the view. In auto layout, we refer this kind of constraints as spacing constraints. To create these spacing constraints, you can use the Pin button of the layout button. But this time we'll use the *Control-drag* approach to apply auto layout. In Interface Builder, you can control-drag from an item to itself or to another item along the axis for which you want to add the constraint. To add the first spacing constraint, hold the `control` key and drag from the label to the right until the view becomes highlighted in blue. Now release the button, you'll see a pop-over menu showing a list of constraint options. Select "Trailing space to container margin" to add a spacing constraint from the label to the view's right margin.

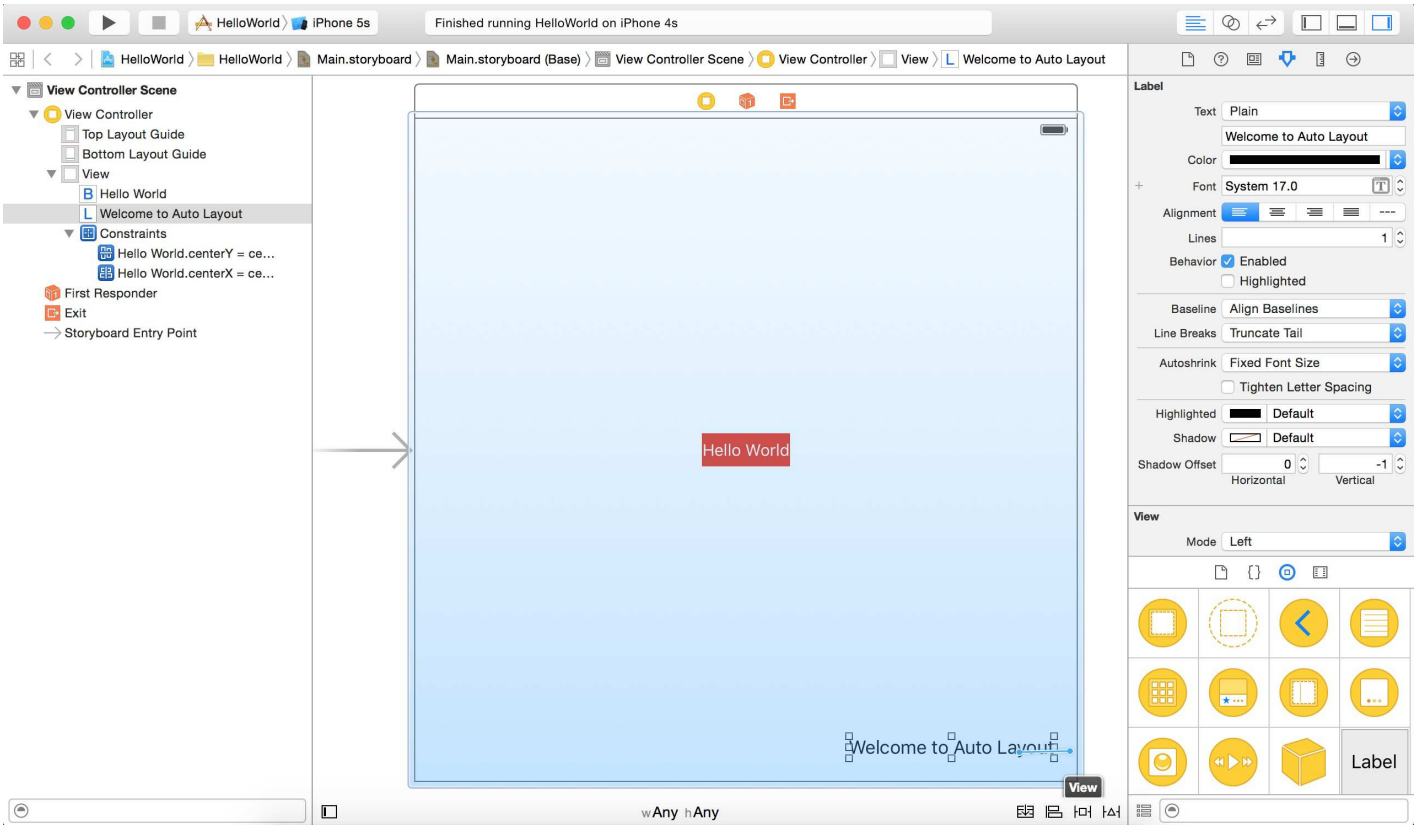


Figure 5-13. Using Control-drag to add the first constraint

In the document outline view, you should see the new constraint. Interface Builder now displays constraint lines in red indicating that there are some missing constraints. That's normal as we haven't defined the second constraint.

Now control-drag from the label to the bottom of the view. Release the button and select

"Vertical Spacing to Bottom Layout Guide" in the shortcut menu. This creates a spacing constraint from the label to the bottom layout guide of the view.

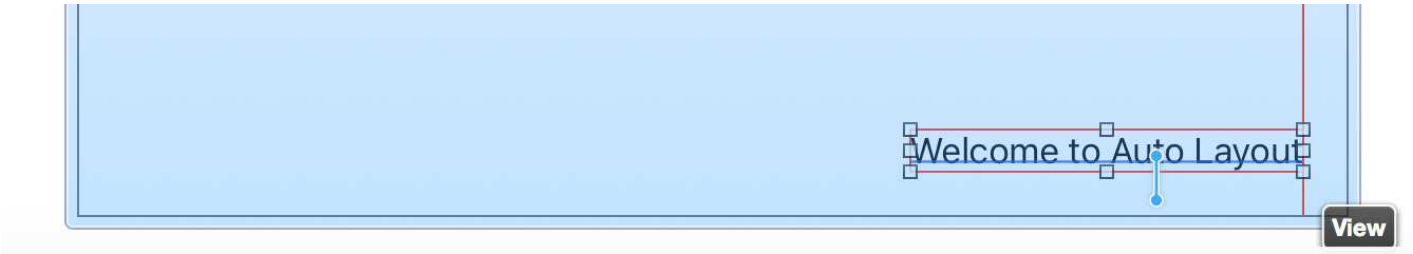


Figure 5-14. Using Control-drag to add the second constraint

Once you added the two constraints, all constraint lines should be in solid blue. When you preview the UI or run the app in simulator, the label should display properly on all screen sizes, and even in landscape mode.

Figure 5-15. The UI now supports all screen sizes

Top and Bottom Layout Guide

You may wonder what the Bottom Layout Guide means. Generally, the Bottom Layout Guide refers to the bottom of the view, like the one shown in the example. Sometimes, the Bottom Layout Guide varies. For example, if there is a tab bar, the Bottom Layout Guide will refer to the top of tab bar.

For the Top Layout Guide, it sits at 20 points (which is the height of the status bar) from the top of the view. Again the guide varies. If the view controller is embedded in a navigation controller, the guide sits below the navigation bar.

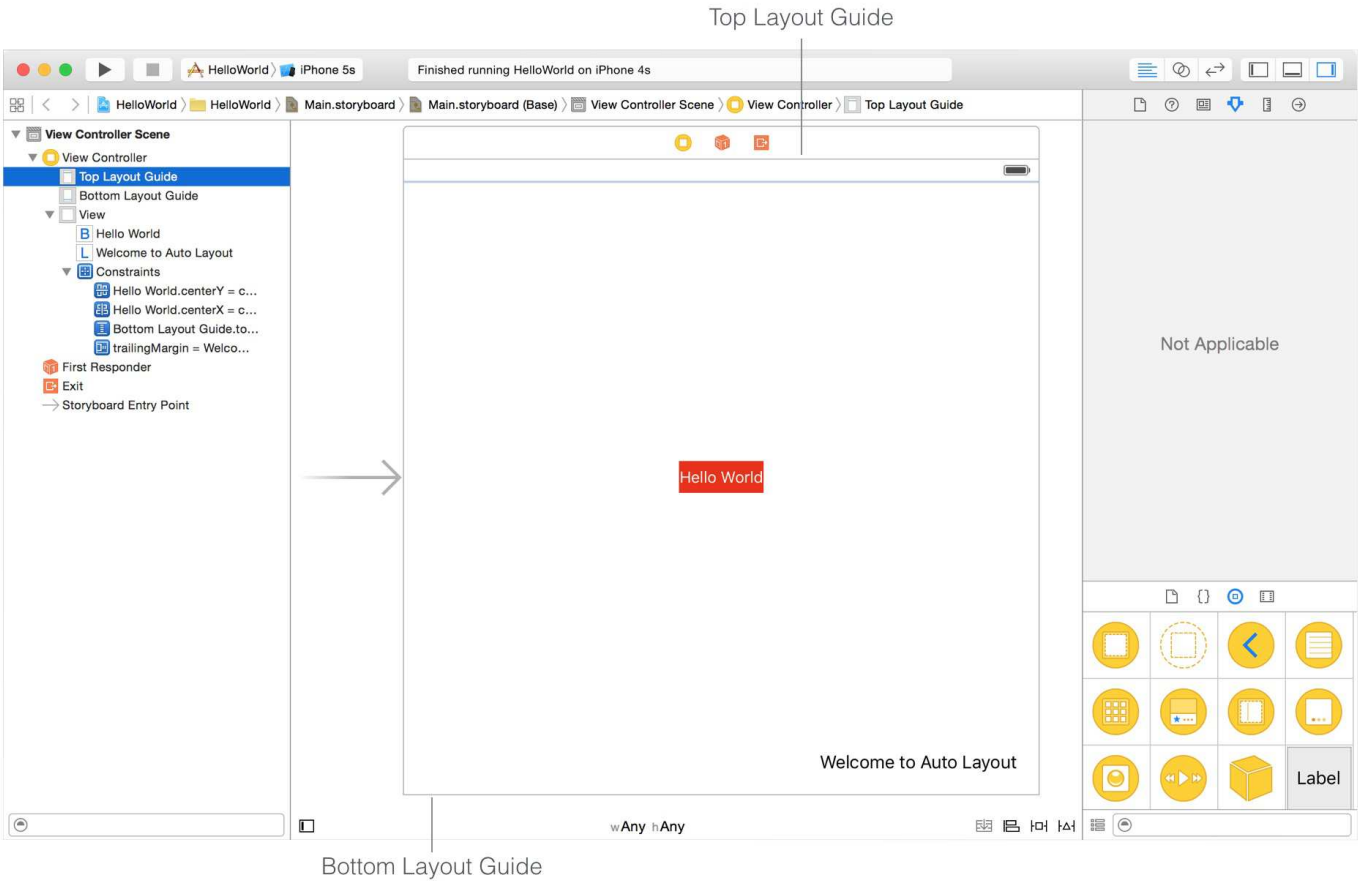


Figure 5-16. Top layout guide

If you're confused, the best way is to check out the guide in Interface Builder. In the document outline, select the "Top Layout Guide" or "Bottom Layout Guide" in the Document Outline. Interface Builder will show you the guide line in light blue.

Editing Constraints

The label is now located 0 points away from the right margin of the view. What if you want to add some space between the label and the right margin of the view? Interface Builder provides a convenient way to edit the constant of a constraint.

You can simply choose the constraint in the document outline view. Here, you should select "trailingMargin" constraint. In the Attributes inspector, you can find the properties of this constraint including relation, constant, and priority. The constant is now set to 0. You can change it to 20 to add some extra space.

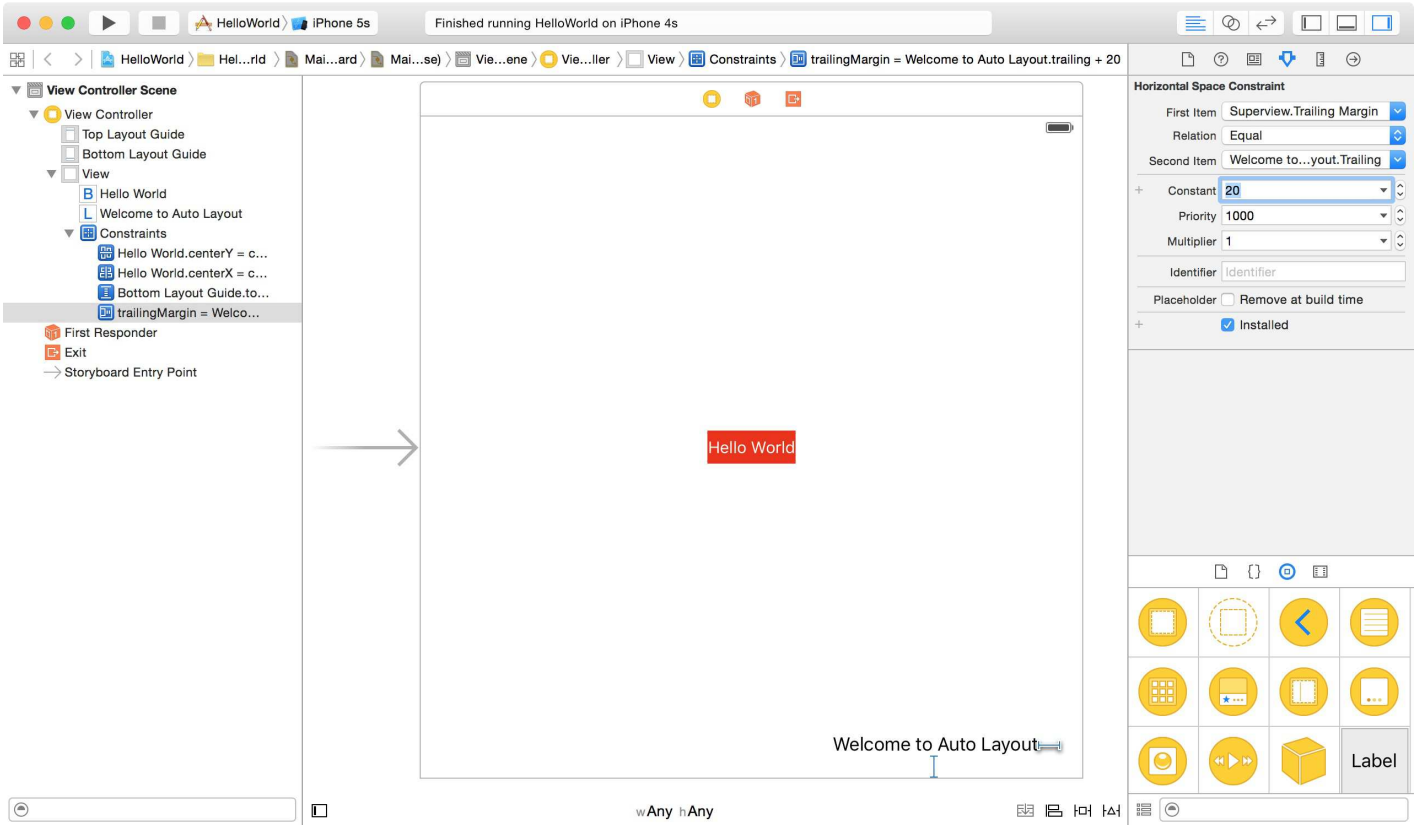


Figure 5-17. Editing a constraint

Your Exercise

By now, I hope you should have some basic ideas about how to lay out your app UI and make it fit for all screen sizes. This is the first exercise for you. It is pretty simple. All I need you to do is

add another label named **My First App** to the view with the following constraints:

- The new label should be 40 points away from the top layout guide.
- The new label should be centered horizontally.

Optionally, you can increase the font size of the label to 30 points. To change the font size, you just open the Attributes inspector and edit the *Font* option. Your end result should look like this:

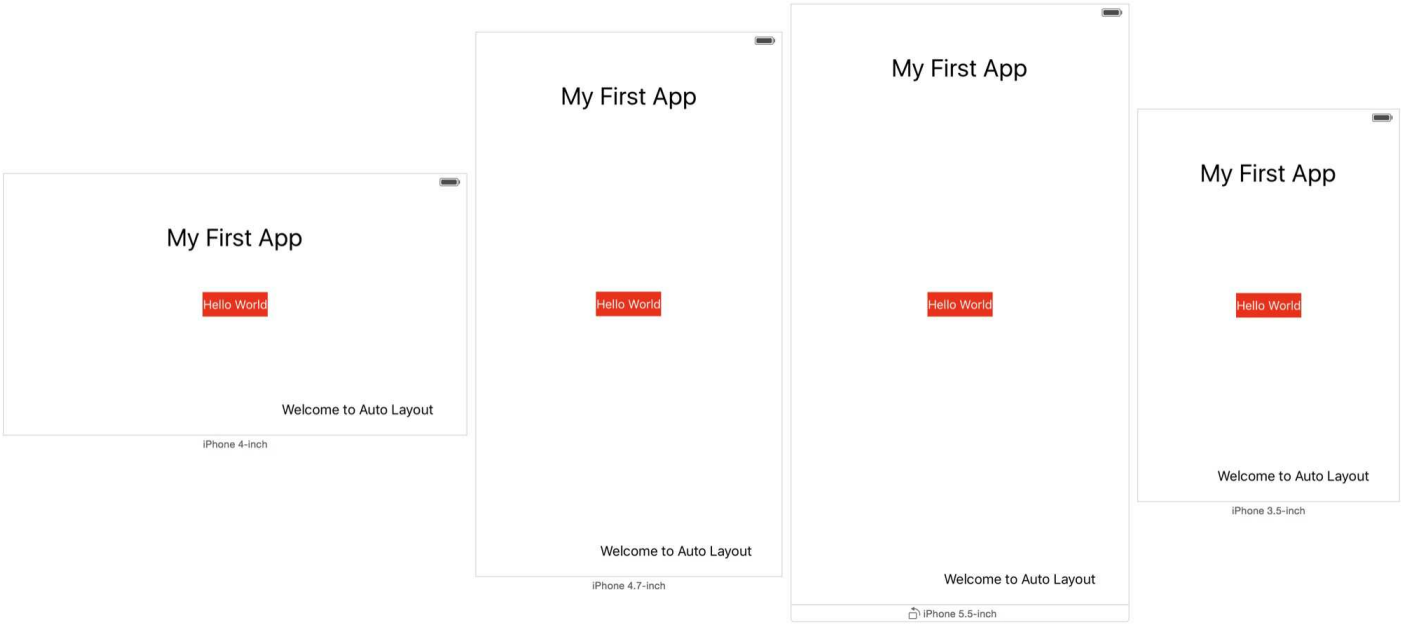


Figure 5-18. The expected app layout

Summary

In this chapter, we went through the basics of Auto Layout. Yes, it's just the basics because I don't want to scare you away from learning auto layout. As we dig deeper and create a real app, we'll continue to explore some other features of auto layout.

Most of the beginners (and even some experienced iOS programmers) avoid using auto layout because it looks confusing. If you throughly understand what I have covered in this chapter, you're on your way to becoming a competent iOS developer. The original iPhone was launched

in 2007. Over these years, there have been tons of changes and advancements in the area of iOS development. Unlike the good old days when your apps just needed to run on a 3.5-inch, non-retina device, your apps now have to cater to various screen resolution and sizes. This is why I devoted a whole chapter to Auto Layout. So take some time to digest the materials.