Lesson 1: Views and Auto Layout

INTRODUCTION

Windows and Views are used to present app content on the screen. Use a view to define a portion of the window that requires content. For example, images and text will use a view. Every app will have at least one window and one view for presenting content.

Auto Layout can be used to lay out the app's user interface. Auto layout is built into Interface Builder in Xcode 5.

LESSON OBJECTIVES

By the end of this lesson, the student will be able to:

1. Explain why view hierarchy is important.
2. Differentiate between a parent (superview) and a child (subview).
3. Determine where a view goes in the view hierarchy.
4. Define first responder.
5. Define the role of the responder chain.
6. Determine if a UI element was able to trigger actions.
7. Define the following terms: auto layout, constraint.
8. Explain adding an icon to a project.
9. Explain creating constraints between parent and child views.
10. Explain creating constraints between two views in the same level of view hierarchy.

LEARNING SEQUENCE

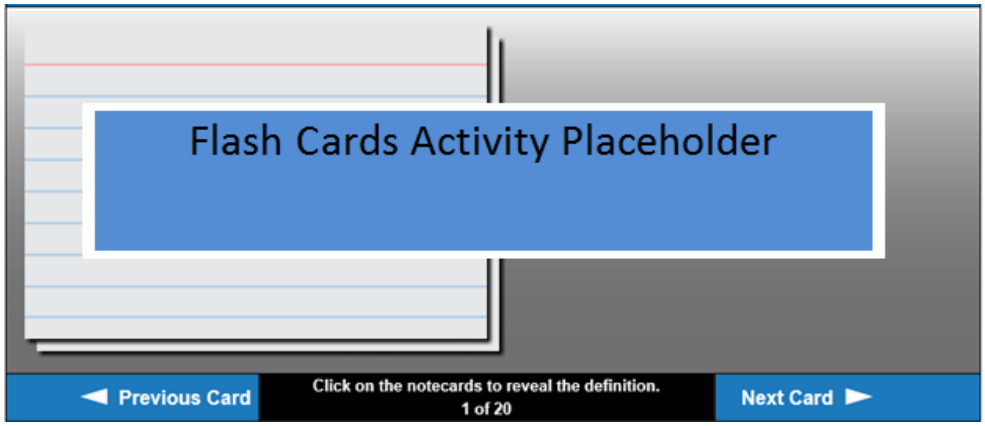
Required Reading	Read the following: • Lesson 1: Views and View Hierarchy
Resources	View the following: • *Starting from a UIWindow Template and Creating a UIView Based Template-iPhone* (10:19) • *Xcode 5 Login ViewController Screen with Auto Layout and Layout Constraints – Part 3/5* (10:33)
Assignments	Complete the following: 1. Quiz 1

KEY TERMS

As you read your lesson, pay close attention to the key terms and phrases listed throughout the lesson. These terms and concepts are important to your understanding of the information provided in the lesson.

Karina Whetstine 2/17/2015 2:57 PM
Comment [1]: Please link to document titled:

L6_Key_Terms



INSTRUCTION

Architecture Fundamentals

The **window** is a container for the views, so there is no actual visible content in a window. UIKit and other frameworks provide predefined views. **Views** can range from buttons and text labels to table views and scroll views. A custom view can also be created, if the predefined views do not provide what is needed. The **UIView** and **UIWindow classes** provide the ability to manage the layout and presentation of views.

View objects, or instances of UIView class, define a rectangular region on the screen and handle the drawing and touch events for that area. Views work with the Core Animation layers to render and animate the view's content. When drawing code is called, the results are cached by Core Animation. By caching the results, it is now possible to reuse that rendered content, eliminating the drawing cycle which would otherwise be needed to update a view. Reusing content is especially important for animation.

A view can act as a container for other views. This creates a parent-child relationship between the two views as shown in Figure 1.

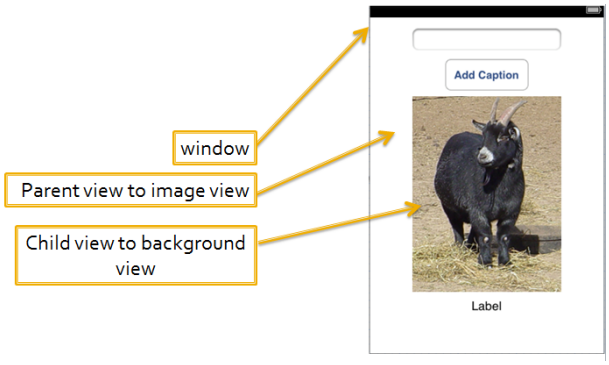


Figure 1: The parent view is the background while the child view is the image all contained in the window

The parent view is known as the **superview**, and the child view is the **subview**. The content of a subview will obscure all or part of its parent view and changing the size of a parent view can have a ripple effect on any subviews. The arrangement of views, the view hierarchy, determines how the app will respond to events such as a touch event.

Windows

Every app requires at least one window, a blank container for one or more views. A window is an instance of the UIWindow class. A window object does the following:

- Contains the app's visible content
- Delivers touch events to the views and other application objects
- Works with view controllers to facilitate orientation changes

Xcode project templates automatically create the app's main window in Interface Builder. Make sure, though, that Full Screen at Launch is enabled in the attributes inspector so that the window is not smaller than the screen of the target device.

Views

Since view objects are the way that the user interacts with the app, views do many things such as:

- Layout and subview management
- Drawing and animation
- Event handling

Use Interface Builder to add views to the interface, arrange views into hierarchies, configure view settings, and connect views.

View the video, *Starting from a UIWindow Template and Creating a UIView Based Template-iPhone* (10:19) to see how app development can be built from the window view up.

Karina Whetstine 2/17/2015 2:57 PM
Comment [2]:

http://www.youtube.com/watch?v=JegNs2xbLms

<iframe width="420" height="315" src="//www.youtube.com/embed/JegNs2xbLms" frameborder="0" allowfullscreen></iframe>

View Types

There are several different view types that a developer has to choose from

Container views are used to group other views or to visually set apart an area, as shown in Figure 2.

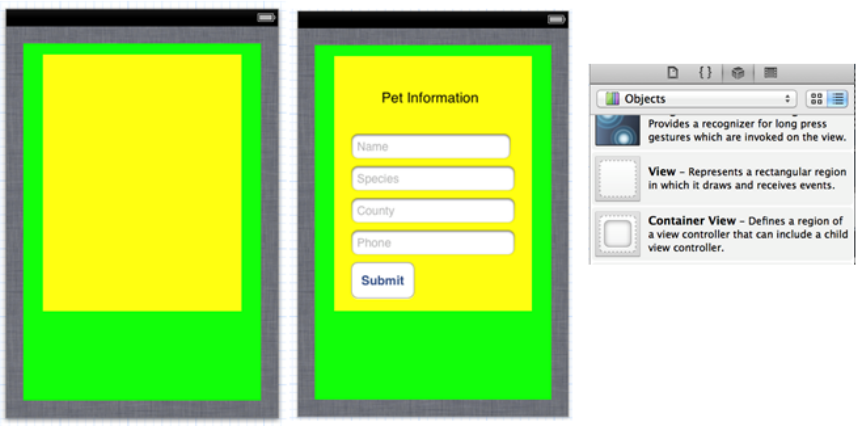


Figure 2: Container View

Controls are views that provide for user interaction because they have the ability to "see" events as shown in Figure 3.

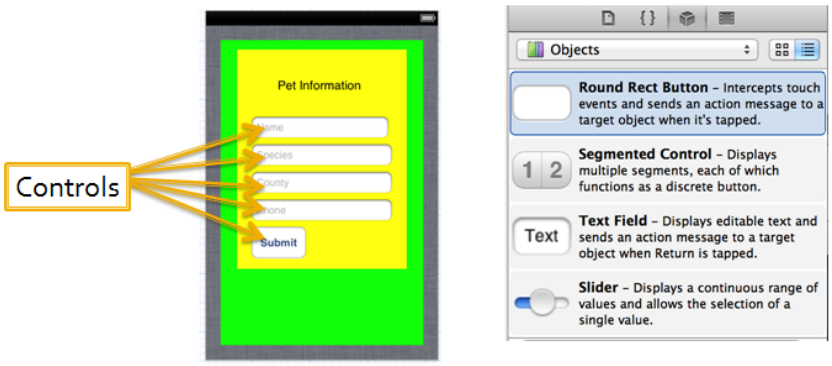


Figure 3: Controls

Display views provide information to users, but do not have user interaction (IBActions) available. The image and label in Figure 4 are both display views.

Figure 4: Display Views

Navigation and tab bars are used with View Controller for navigation. Figure 5 shows both a navigation bar and a tab bar.

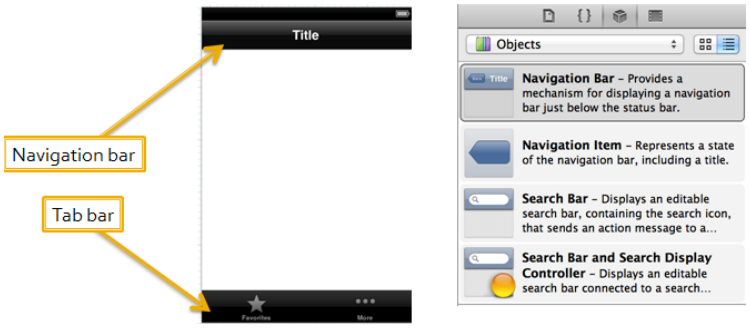


Figure 5: Navigation and Tab bars

Alert Views and **Action Sheets** can be added to provide information or to prompt a user for a choice. An alert has a blue popup box and displays an informative alert message to the user. An Action Sheet has a panel that slides up from the bottom and it displays a set of buttons that offers choices to the user.

Responder Chain

View objects are responder objects, instances of the UIResponder class, and capable of receiving touch events, motion events, remote control events, and action messages. The **first responder** is the view that is currently active. If a particular view does not handle that particular event, it passes the event object along to its superview. If the superview does not handle the event, it passes the event object up to its superview. The event object may continue up the responder chain until it reaches the window. If the window does not know what to do with the event object, it goes to Object(AppDelegate). If AppDelegate does not know what to do, the event is ignored.

Auto Layout

Auto Layout is built into Interface Builder in Xcode. Auto Layout is used to lay out the app's user interface by mathematically defining relationships between elements. It can easily handle changes in screen sizes and orientation. For example, if a developer centers an image horizontally, that image will remain horizontally centered even if the user changes the orientation of the device.

By default, Auto Layout is available when a new application is created, but the option can be de-selected. With Auto Layout, the developer is able to set a position relative to a superview and to other subviews.

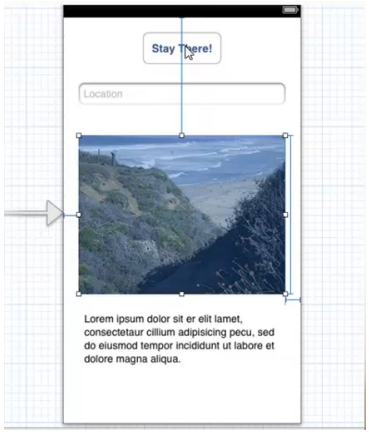


Figure 6 displays an image showing the constraints (blue lines)

Constraints are the building blocks in Auto Layout used to set the distance between views, to center a view, or to set the distance between a view and its parent view. In figure 6, the blue line on the side of the image is the constraint setting the height of the image. There is also a constraint to position the image a set number of points from the top and one to position the image a set number of points from the left. The image will always be the same number of points from the top and left no matter what the screen size is. In the above example, the image is a child of the background view.

Document Outline

Go to the Document Outline to see what constraints are present in the View Controller. If a constraint is selected by clicking it on the canvas or selecting it from the outline view, its definition can be edited in the Attributes Inspector.

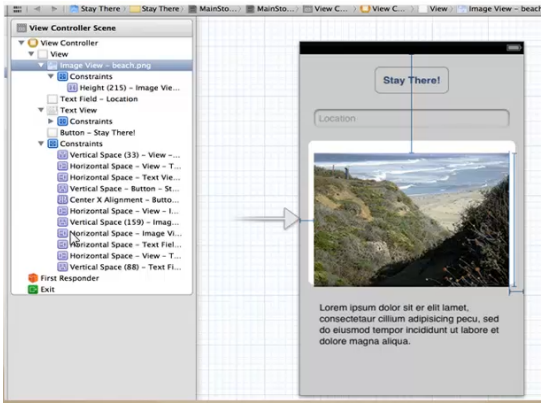


Figure 7: Document Outline

Size Inspector

If the image view is selected, click on the Size Inspector icon in the Toolbar.

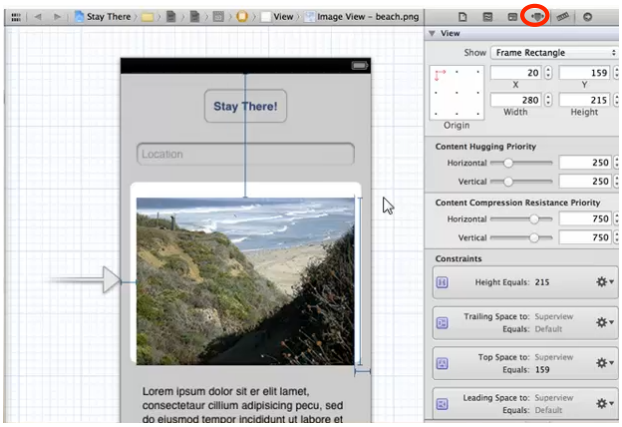


Figure 8: Size Inspector

As shown in Figure 8, the **Size Inspector** displays the width and height of the image in addition to its compression. Use compression resistance and content hugging priorities to set how sensitive the object is to changing size. The constraints appear at the bottom of the view

Individual Constraint Settings

To see an individual constraint setting, select a constraint and go to Attribute Inspector as shown in Figure 9 by clicking the appropriate icon on the Toolbar.

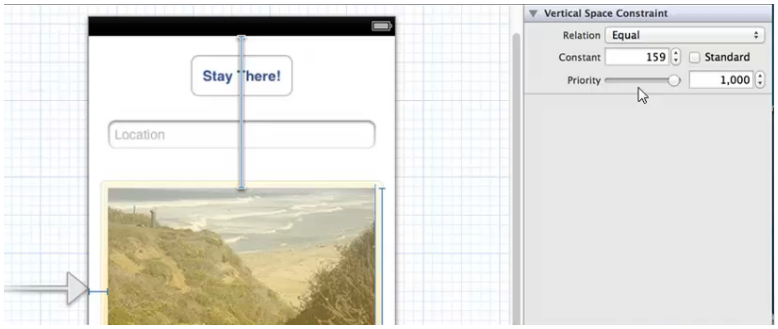


Figure 9: Display the settings for an individual constraint

Constraints can be explicitly set or they can be a variable. An explicit example follows:

yourLabel.width=100

An example of a variable constraint follows:

yourLabel.width<=100

Constraint priority can also be set. Valid values range from 0 to 1,000 with zero having the lowest priority, and 1,000 having the highest priority. In Figure 9, an explicit constraint is being used along with the highest priority. Zero is considered an optional constraint and a priority of 1,000 is considered a required constraint.

Adding and Working With Constraints

A constraint can be added visually in Interface Builder by holding down the Control key and dragging from a view on the canvas or by using the Auto Layout menu on the Interface Builder canvas. A constraint can also be coded in using the Visual Format Language or API code.

Several constraints are added automatically when an object is added to Interface Builder. If the lines are purple, this indicates that the line is required. Blue lines are not required and can easily be deleted. If a purple line needs to be removed, add constraints until the purple line changes to blue and then delete it.

Issues

Issues occur if there are constraints which are working against each other, if there are not enough constraints provided, or when the final layout contains a set of constraints that are unclear. Interface Builder displays Auto Layout issues in the following places:

- The Issues Navigator
- The Interface Builder outline View
- On the canvas

For example, if a constraint is missing, a red arrow appears in the Interface Builder outline indicating the constraint problem with the image view. By clicking on the red arrow and clicking *Add missing constraints*, Xcode can fix the problem.

When a constraint line is setup, the constraint line will be either orange or blue. An orange constraint line indicates that there are insufficient constraints as shown in Figure 10. This can be fixed by adding additional constraints.

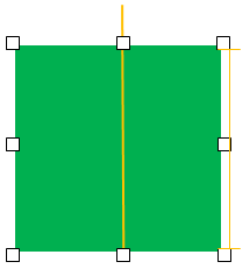
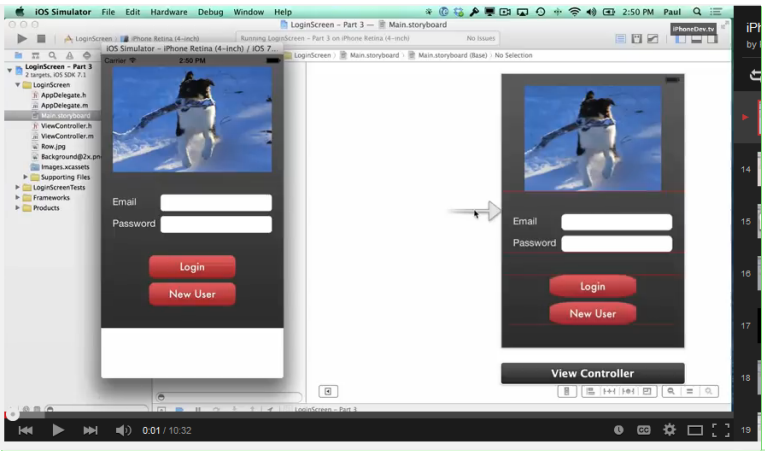


Figure 10: Orange constraint

View the video, *Xcode 5 Login ViewController Screen with Auto Layout and Layout Constraints – Part 3/5* (10:33).



Karina Whetstine 2/17/2015 2:57 PM
Comment [3]:
http://youtu.be/qsvaVgUQF-o?list=PLLYKjb-Uo9cxRjebq32cM2-eROA5soXYf

<iframe width="560" height="315" src="//www.youtube.com/embed/qsvaVgUQF-o?list=PLLYKjb-Uo9cxRjebq32cM2-eROA5soXYf" frameborder="0" allowfullscreen></iframe>

SUMMARY

This lesson introduced the view hierarchy of an iOS app. Every app requires at least one window, a blank container for one or more views. View objects allow a user to interact with the app. Use auto layout to lay out user interface. It can easily handle changes in screen sizes and orientation.

ASSIGNMENTS

1. Quiz 1