|  |
| --- |
| COMP 3097 IOS APPLICATION DEVELOPMENT LAB MON MARC 23 2020 |

Start of Class Quiz to get everyone mentally focused and in the zone: **Do This Now.**

<https://t2m.io/Comp3097-46983-Lab-March-23>

References:

<https://developer.apple.com/documentation/uikit/uialertcontroller>

## Learning Outcomes:

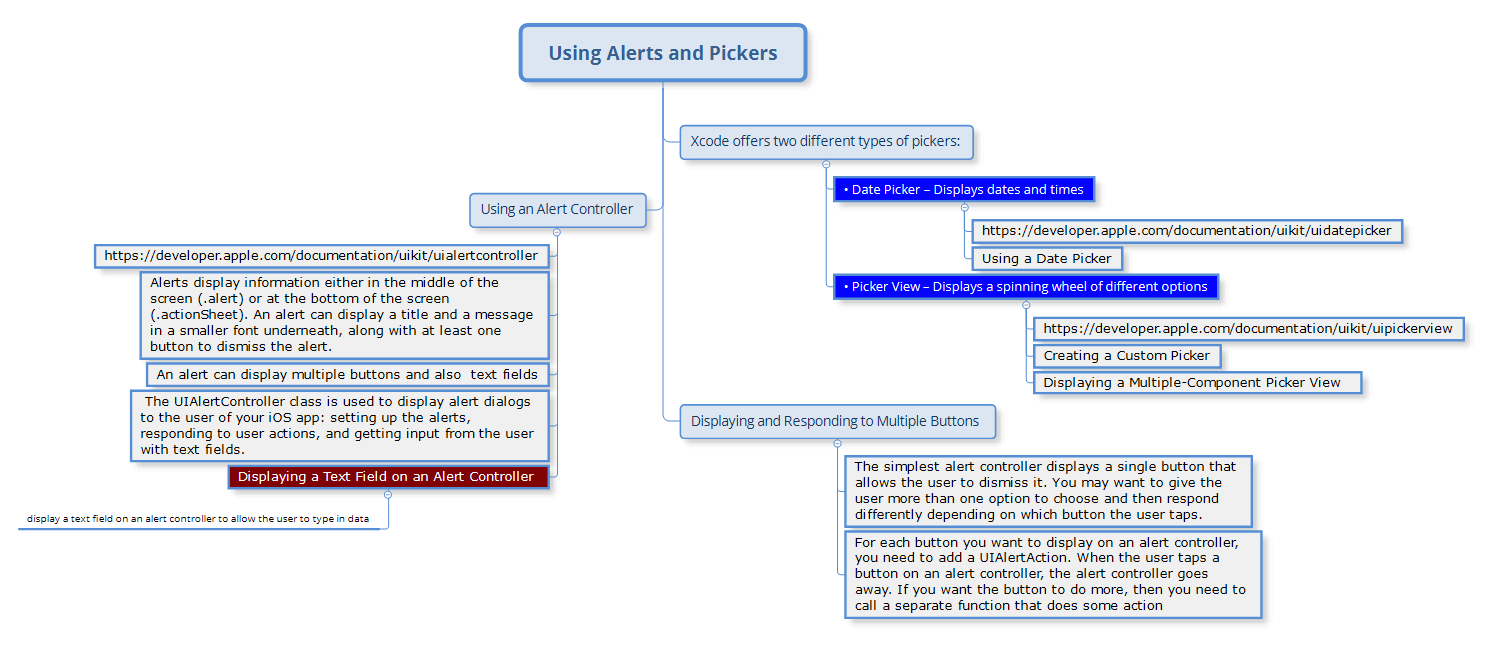
## Using Alerts and Pickers

### Displaying and accepting data from the user.



Visual Topics Roadmap:

LINK: <http://www.xmind.net/m/eMPCYu>



The simplest way to display data is through a label, but sometimes you need to display data and give the user a way to respond. In that case, you need to use an **alert controller.**

When creating your alert controller, you can choose between two options:

* **A simple alert**, with the .alert constant
* **An action sheet**, with the .actionSheet constant

## Using an Alert Controller

Every user interface needs to display data back to the user. In some cases, this data can be displayed just in a label, but sometimes you need to make sure the user sees certain information. In those situations, you should use an alert controller.

An alert controller appears over an app’s user interface and can be customized by changing the following properties:

• Title – Text that appears at the top of the alert controller, often in bold

and a large font size

• Message – Text that appears underneath the title in a smaller font size

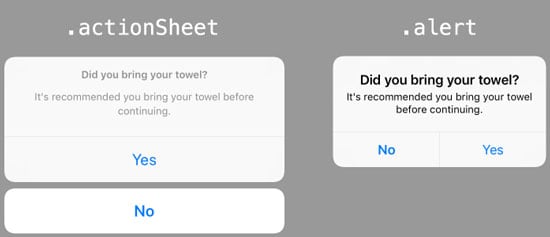
• Preferred style – Defines the appearance of the alert controller as an

action sheet (as shown in Figure 12-3) that appear at the bottom of

the screen or as an alert (see Figure 12-1) that appear in the middle of

the screen

Here’s what that exact same alert looks like, with both options:



A title typically consists of a single word or short phrase that explains the purpose of the alert controller such as displaying “Warning” or “Log In”.

To dismiss an alert controller, an alert controller always needs at least one button.

However, an alert controller can display two or more buttons to give user a choice. Besides displaying buttons, alert controllers can also text fields to allow users to enter data as well.

You can create an alert controller solely by writing Swift code. First, you must create the alert controller that defines a title, message, and style. Second, you must define an alert action for each button or text field you want to display on the alert controller.

Third, you must actually present that alert controller on the user interface.

To see how to create a simple alert controller that does nothing but display a title, a

message, and a button to dismiss it, follow these steps:

1. Create a new iOS project using the Single View App template

name this new project AlertControllerApp. This creates a single view for the user interface.

2. Click the Main.storyboard file in the Navigator pane. Xcode displays the single view.

3. Click the Library icon to open the Object Library window.

4. Drag and drop a button anywhere on the view.

5. Choose Editor ➤ Resolve Auto Layout Issues ➤ Reset to Suggested

Constraints. Xcode adds constraints to the button.

6. Choose View ➤ Assistant Editor ➤ Show Assistant Editor. Xcode displays

the Main.storyboard and ViewController.swift file side by side.

7. Move the mouse pointer over the button, hold down the Control key, and Ctrl-drag from the image view to the ViewController.swift file above the last curly bracket at the bottom of the file. Release the Control key and the left mouse button. A popup window appears.

8. Release the Control key and the left mouse button. A popup

window appears.

9. Click in the Name text field, type buttonTapped, click the Type popup menu and choose UIButton, and click the Connect button. Xcode creates an IBAction method.

10. Edit this buttonTapped IBAction method as follows:

@IBAction func buttonTapped(\_ sender: UIButton) {

let alert = UIAlertController(title: "Warning", message:

"Zombies are loose!", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default,

handler: { action -> Void in

//Just dismiss the action sheet

})

alert.addAction(okAction)

self.present(alert, animated: true, completion: nil)

}

11. Click the Run button or choose Product ➤ Run. The Simulator

screen appears.

12. Click the button. Notice that the alert controller appears in the

middle of the screen because the preferredStyle were .alert. (If

the preferredStyle property were .actionSheet, then the alert

controller would appear at the bottom of the screen.)

13. Tap the OK button on the alert controller to make it go away.

14. Choose Simulator ➤ Quit Simulator to return to Xcode.

## Action sheets have three benefits over alerts:

* They slide up from the bottom of the iPhone screen, so you can reach the action sheet buttons with your thumb. Easy!
* You can fit more text on the button, compared to the alert style.
* Usually, you need *less* text than an alert, because the action sheet buttons are self-explanatory.

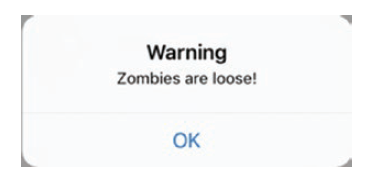
When should you use an alert, and when an action sheet?

Use an action sheet for choosing between multiple similar options, like *“Choose from Photo Library”* and *“Take Picture with Camera”*.

When you need explicit confirmation on a positive and negative action, like *“Yes”* and *“No”*, or *“OK”* and *“Cancel”*, it’s best to use an alert style dialog.

It’s fairly typical for action sheets to *not* display a title or message to the user. Instead, you let the options speak for themselves. When you present a user with an action sheet, always give them the option to get out of the action sheet with a *Cancel* button.

An **alert controller** can display a sheet that pops up on the screen, giving the user a chance to respond. Alert controllers appear over the user interface to force the user to respond by tapping a button to make a choice or dismiss the alert controller as shown



## Besides displaying data, user interfaces also need to let users input data.

If that data is text, then you can use a text field to let the user type anything in that text field such as

a name or telephone number. A text field is perfect for accepting any type of data, but

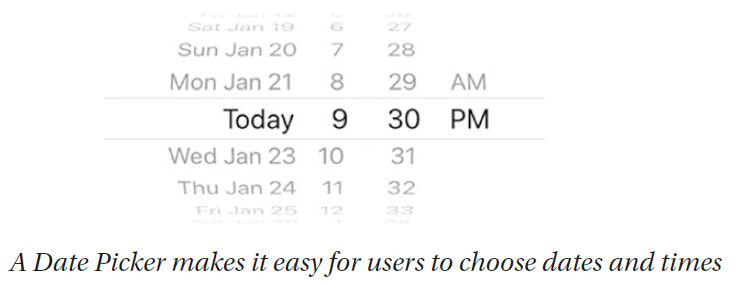
when you need to offer the user a list of valid choices, use a picker instead.

### Xcode offers two different types of pickers:

• Date Picker – Displays dates and times as shown in Figure 12-2

• Picker View – Displays a spinning wheel of different options

*An alert controller typically displays a message and one or more buttons*



A picker contains a limited number of options on one or more wheels that the user can spin around to select a particular option. The available number of options can be

as small as two (such as the AM and PM option for choosing a time), but typically the available options are greater than two. Pickers ensure that the user can never choose a non-valid option such as typing a number for a name or typing a city incorrectly.

How to use the UIAlertController class to display alert dialogs to the user of your iOS app.

* Setting up the alerts
* Responding to user actions
* Getting input from the user with text fields.

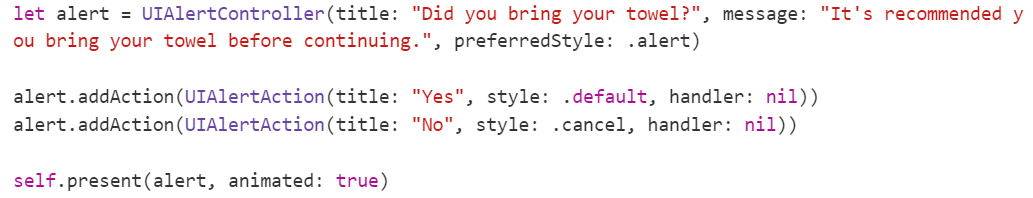
The UIAlertController is the default way to ask users to confirm an action.

UIAlertController also supports the action sheet – a dialog style that allows your app’s users to quickly take action.

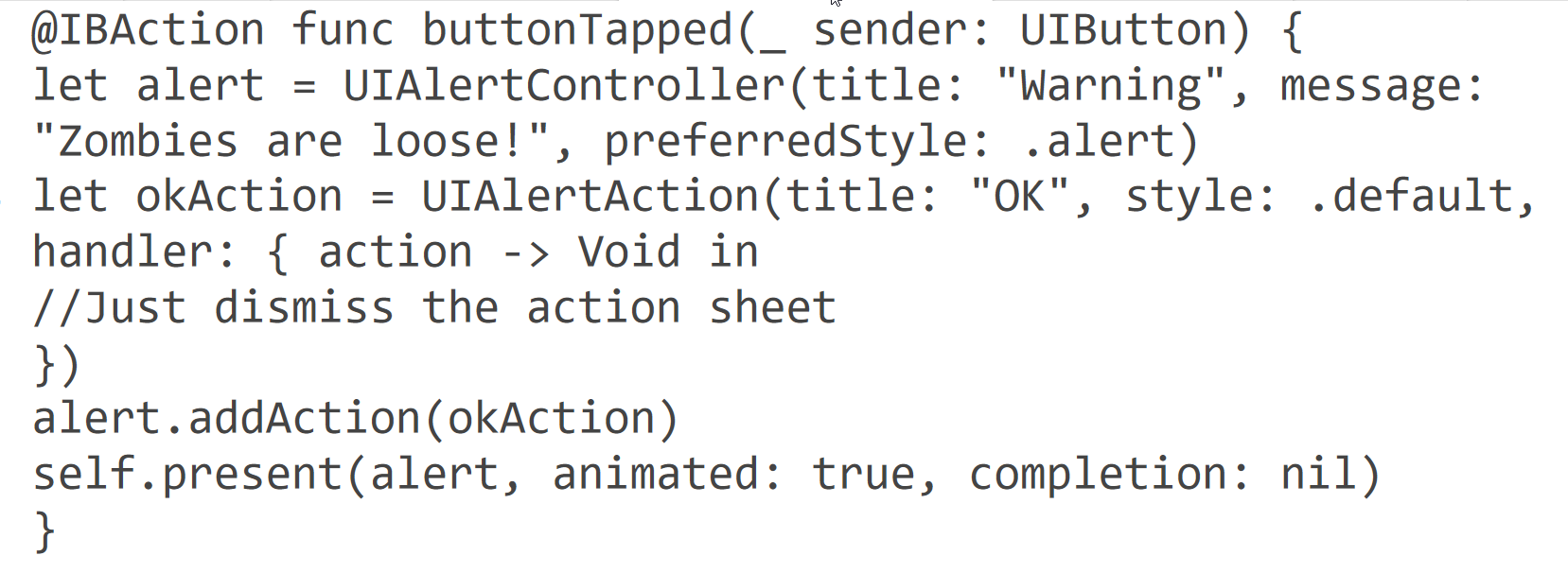
Most alerts can’t be dismissed without taking an action, and that’s why they’re so effective. When an alert is presented, it fills the entire iPhone screen. You can’t move away without making a choice. Such an alert is known as a modal dialog. On iOS, they’re simply called alerts.

Always have a Cancel option that dismisses the alert and reverts the app to its previous state.

Example:



## A Simple Example: Put this code into the Action method of a BUTTON:



# Displaying and Responding to Multiple Buttons

The simplest alert controller displays a single button that allows the user to dismiss it. However, you may want to give the user more than one option to choose and then respond differently depending on which button the user taps.

### For each button you want to display on an alert controller, you need to add a UIAlertAction.

**When the user taps a button on an alert controller, the alert controller goes away. If you want the button to do more, then you need to call a separate function that does some action.**

To see how to create an alert controller that displays two buttons and responds differently to each button, follow these steps:

Create a new iOS project using the Single View App template and name this new project AlertControllerButtonsApp.

This creates a single view for the user interface.

2. Click the Main.storyboard file in the Navigator pane. Xcode displays the single view.

3. Click the Library icon to open the Object Library window.

4. Drag and drop a button and a label anywhere on the view.

5. Choose Editor ➤ Resolve Auto Layout Issues ➤ Reset to Suggested Constraints on the bottom half of the submenu. Xcode adds constraints to the button and label.

6. Choose View ➤ Assistant Editor ➤ Show Assistant Editor. Xcode

displays the Main.storyboard and ViewController.swift file side by side.

7. Move the mouse pointer over the label, hold down the Control key, and Ctrl-drag from the image view to the ViewController.swift file under the “class ViewController” line.

8. Release the Control key and the left mouse button. A popup window appears.

9. Click in the Name text field, type labelResult, and click the Connect button. Xcode creates an IBOutlet variable as follows: @IBOutlet var labelResult: UILabel!

10. Move the mouse pointer over the button, hold down the Control key, and Ctrl-drag from the image view to the ViewController.swift file above the last curly bracket at the bottom of the file.

11. Release the Control key and the left mouse button. A popup window appears.

12. Click in the Name text field, type buttonTapped, click in the Type popup menu and choose UIButton, and click the Connect button.

Xcode creates an IBAction method.

13. Choose View ➤ Standard Editor ➤ Show Standard Editor, or click the Standard Editor icon in the upper right corner of the Xcode window.

14. Click the ViewController.swift file in the Navigator pane.

15. Modify the viewDidLoad method as follows:

override func viewDidLoad() {

super.viewDidLoad()

labelResult.numberOfLines = 0

//Do any additional setup after loading the view, typically from a nib.

}

Setting the numberOfLines property of the label to 0 allows it to expand in width no matter the size of text stored in it. If this numberOfLines property is left at its default value of 1, then the label will not resize automatically and risks cutting text off if the text is longer than the width of the label.

16. **Edit this buttonTapped IBAction method as follows:**

@IBAction func buttonTapped(\_ sender: UIButton) {

let alert = UIAlertController(title: "Warning", message:

"Zombies are loose!", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default,

handler: { action -> Void in

self.labelResult.text = "OK"

})

let cancelAction = UIAlertAction(title: "Cancel", style:

.cancel, handler: { action -> Void in

self.labelResult.text = "Cancel"

})

let destroyAction = UIAlertAction(title: "Destroy", style:

.destructive, handler: { action -> Void in

self.labelResult.text = "Destroy"

})

alert.addAction(okAction)

alert.addAction(cancelAction)

alert.addAction(destroyAction)

self.present(alert, animated: true, completion: nil)

}

The code to define each UIAlertAction simply changes the text displayed in the label. If you need each button to perform more complicated tasks, you can put the name of one or more functions

in the handler.

Then you’ll need to create those functions such as:

let destroyAction = UIAlertAction(title: "Destroy", style:.destructive, handler: { action -> Void in

self.labelResult.text = "Destroy"

self.callFunctionOne()

self.callFunctionTwo()

})

func callFunctionOne(){

*// Code here*

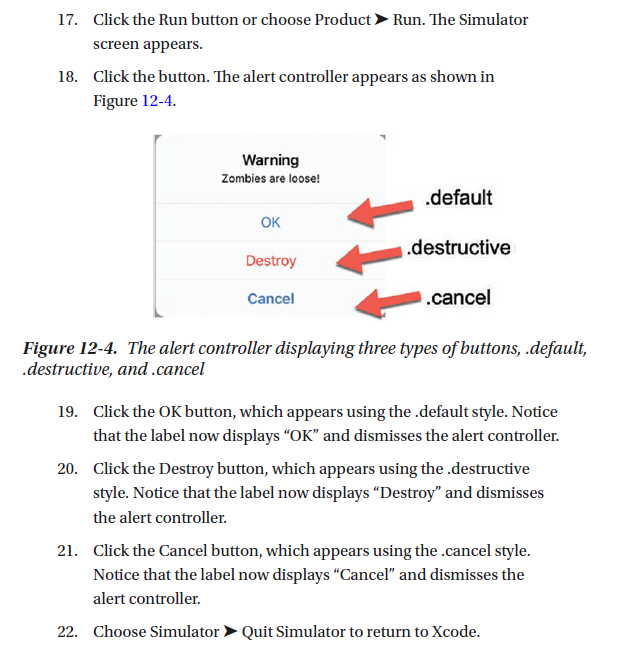
}

func callFunctionTwo(){

*// Code here*

}

Enter the entire Swift file from <https://github.com/computationalknowledge/SwiftAlertHandlers/blob/master/Snippet2>



**Displaying a Text Field on an Alert Controller**

**Buttons let an alert controller accept choices from the user. Based on those choices, your code can then respond. However, sometimes you might want to display a text field on an alert controller to allow the user to type in data. Then you’ll need to store this data.**

Note You can only add text fields on an alert controller that appears in the .alert

preferredStyle. You cannot display text fields on an alert controller that appears as

the .actionSheet preferredStyle.

When you add a text field to an alert controller, you can modify that text field like any text field such as defining a background color or font. In addition, you must also create a constant to store the contents of that text field so the rest of your app can access whatever data the user typed into the text field on the alert controller.

To see how to display a text field on an alert controller and access its contents, follow

these steps:

**#LabToDo**

1. Create a new iOS project using the Single View App template and name this new project

AlertControllerTextFieldApp. This creates a single view for the user interface.

2. Click the Main.storyboard file in the Navigator pane. Xcode displays the single view.

3. Click the Library icon to open the Object Library window.

4. Drag and drop a button and a label anywhere on the view.

5. Choose Editor ➤ Resolve Auto Layout Issues ➤ Reset to Suggested

Constraints on the bottom half of the submenu. Xcode adds

constraints to the button and label.

6. Choose View ➤ Assistant Editor ➤ Show Assistant Editor. Xcode displays the Main.storyboard and ViewController.swift file side by side.

7. Move the mouse pointer over the label, hold down the Control

key, and Ctrl-drag from the image view to the ViewController.swift file under the “class ViewController” line.

8. Release the Control key and the left mouse button. A popup window appears.

9. Click in the Name text field, type labelResult, and click the

Connect button. Xcode creates an IBOutlet variable as follows:

**@IBOutlet var labelResult: UILabel!**

10. Move the mouse pointer over the button, hold down the Control

key, and Ctrl-drag from the image view to the ViewController.swift

file above the last curly bracket at the bottom of the file.

11. Release the Control key and the left mouse button. A popup

window appears.

12. Click in the Name text field, type buttonTapped, click in the Type

popup menu and choose UIButton, and click the Connect button.

Xcode creates an IBAction method.

13. Choose View ➤ Standard Editor ➤ Show Standard Editor, or click the Standard Editor icon in the upper right corner of the Xcode window.

14. Click the ViewController.swift file in the Navigator pane.

15. Modify the viewDidLoad method as follows:

**override func viewDidLoad() {**

**super.viewDidLoad()**

**labelResult.numberOfLines = 0**

**// Do any additional setup after loading the view,**

**typically from a nib.**

**}**

Setting the numberOfLines property of the label to 0 allows it

to expand in width no matter the size of text stored in it. If this

numberOfLines property is left at its default value of 1, then the

label will not resize automatically and risks cutting text off if the

text is longer than the width of the label.

16. Add the following to the buttonTapped IBAction method as

follows:

let alert = UIAlertController(title: "Log In", message:

"Enter Password", preferredStyle: .alert)

alert.addTextField(configurationHandler: {(textField) in

textField.placeholder = "Password here"

textField.isSecureTextEntry = true

})

The first line creates an alert controller that displays the title

“Log In” and underneath the message “Enter Password”. Because

we want to display a text field on the alert controller, the alert

controller’s preferredStyle must be .alert.

The second line adds a text field to the alert controller and

configures the text field to display “Password here” as placeholder

text and also sets its isSecureTextEntry property to true which

hides the characters as the user types them in the text field. Any

code listed here simply customized the text field.

17. Add the following to the buttonTapped IBAction method as follows:

let okAction = UIAlertAction(title: "OK", style: .default,

handler: { action -> Void in

let savedText = alert.textFields![0] as UITextField

self.labelResult.text = savedText.text

})

alert.addAction(okAction)

self.present(alert, animated: true, completion: nil)

The first line defines a button with the title “OK” and a .default style. Inside the handler section of the code, this line defines a constant called savedText, which represents the first text field (note the index value of 0) on the alert controller.

If you add more than one text field to an alert controller, you’ll need to define additional

constants to represent those other text fields.

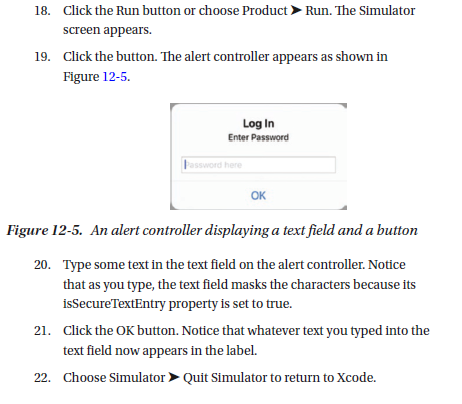
Finally, this line stores the text from the text field (savedText) and displays it in the

labelResult IBOutlet that’s linked to the label on the user interface.

The second line adds the button to the alert controller and the third line presents or displays the alert controller.

**#TODO: Lab Exercise:** Enter and run the entire Swift Controller file:

<https://github.com/computationalknowledge/SwiftAlertHandlers/blob/master/alertController>



## Using a Date Picker

When an app needs the user to input a date and/or a time, it’s best to use the Date Picker

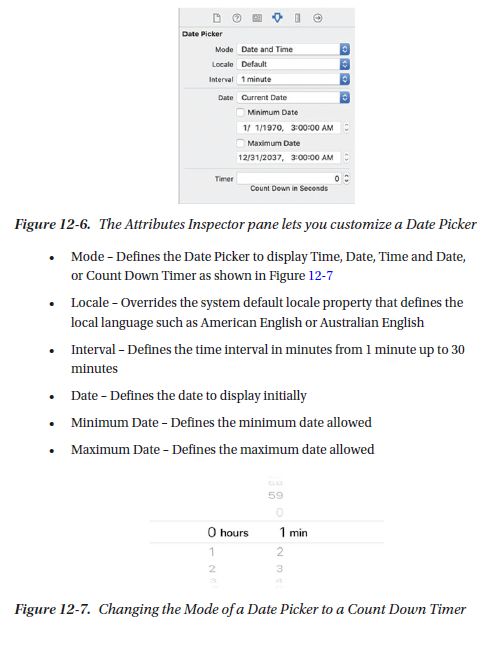
to provide a list of valid options. By spinning different wheels on a Date Picker, users can

pick days, dates, and times without typing a thing.

After dragging and dropping a Date Picker from the Object Library window to a view,

you can customize a Date Picker by modifying the following options in the Attributes

Inspector as shown in Figure 12-6:



No matter what mode you choose for a Date Picker, its value is stored as an NSDate data type. To convert this NSDate data into text, you need a two-step process. First, you need to create a constant that represents a DateFormatter such as

let dateFormatter: DateFormatter = DateFormatter()

Second, you have to define the .dateStyle and .timeStyle properties to display the

date and time. To define the date and time style, you need to set the .dateStyle and

.timeStyle properties of the DateFormatter constant to one of the following options:

• .none – Does not display the date or time

• .short – Displays dates in this format “11/23/19” and times in this

format “3:48 PM”

• .medium – Displays dates in this format “Nov 23, 2019” and times in

this format “3:48:21 PM”

• .long – Displays dates in this format “November 23, 2019” and times

in this format “3:48:21 PM EST”

• .full – Displays dates in this format “Saturday, November 23, 2019

AD” and times in this format “3:48 PM Eastern Standard Time”

dateFormatter.dateStyle = .short

dateFormatter.timeStyle = .short

Finally, you have to retrieve the date/time currently displayed in the Date Picker.

Since the date/time value stored in the Date Picker is in the NSDate data type, you have to convert its .date property to a string by using code such as this:

let selectedDate: String = dateFormatter.string(from: myDatePicker.date)

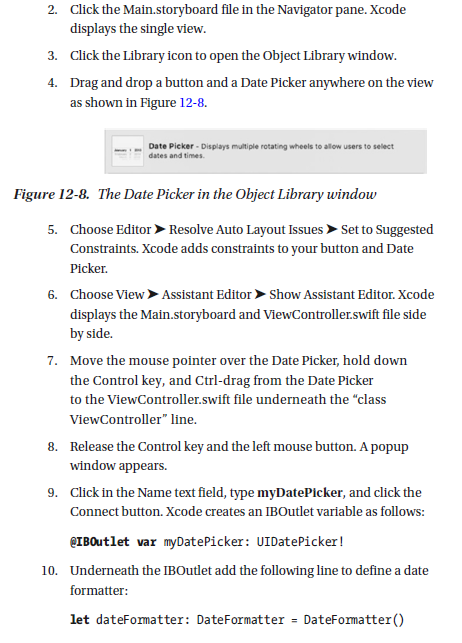
In the following example, we’ll see how to retrieve the value displayed in a Date

Picker in two different ways. First, we’ll retrieve the Date Picker value any time the user changes the Date Picker.

Second, we’ll retrieve the Date Picker value and display it on an alert controller.

To see how to **define the title on a button**, follow these steps:

1. Create a new iOS project using the Single View App template and name this new project DatePickerApp. This creates a single view for the user interface.



11. Edit the viewDidLoad method as follows:

override func viewDidLoad() {

super.viewDidLoad()

dateFormatter.dateStyle = .short

dateFormatter.timeStyle = .short

*// Do any additional setup after loading the view,*

*typically from a nib.*

}

**These lines of code define the .dateStyle and .timeStyle properties**

**to .short**. You can experiment by changing these values to .medium, .long, and .full to see how it defines the date and time differently.

12. Underneath the viewDidLoad method, add the following function which will accept a string and display it in an alert controller:

func ShowAlert(dateTime : String) {

let alert = UIAlertController(title: "Selected Date and

Time", message: "\(dateTime)", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default,

handler: { action -> Void in

//Just dismiss the action sheet

})

alert.addAction(okAction)

self.present(alert, animated: true, completion: nil)

}

13. Move the mouse pointer over the button, hold down the Control key, and Ctrl-drag from the button to the ViewController.swift file above the last curly bracket at the bottom of the file.

14. Release the Control key and the left mouse button. A popup window appears.

15. Click in the Name text field, type getCurrentDateTime, click in the Type popup menu and choose UIButton, and click the Connect button. Xcode creates an IBAction method.

16. **Edit this getCurrentDateTime IBAction method as follows:**

@IBAction func getCurrentDateTime(\_ sender: UIButton) {

let selectedDate: String = dateFormatter.string(from:

myDatePicker.date)

ShowAlert(dateTime: selectedDate)

}

**This IBAction method retrieves the value of the myDatePicker IBOutlet (that represents the Date Picker) and converts this date and time into a string that gets stored in the selectedDate constant.** Then it passes this **selectedDate constant to the Show Alert function to display it on the alert controller**.

17. Move the mouse pointer over the Date Picker, hold down the Control key, and Ctrl-drag from the Date Picker to the ViewController.swift file above the last curly bracket at the bottom

of the file.

18. Release the Control key and the left mouse button. A popup window appears.

19. Click in the Name text field, type dateChanged, click in the Type

popup menu and choose UIDatePicker, and click the Connect

button. Xcode creates an IBAction method.

**20. Edit this dateChanged IBAction method as follows:**

**@IBAction func dateChanged(\_ sender: UIDatePicker) {**

**let selectedDate: String = dateFormatter.string(from:**

**sender.date)**

**ShowAlert(dateTime: selectedDate)**

**}**

This dateChanged IBAction method runs every time the user changes the date or time in the Date Picker. It retrieves the .date property from the Date Picker and converts it into a string, which it stores in the selectedDate constant. Then it passes this selectedDate constant into the Show Alert function that displays it on an alert controller.

**#TODO: Lab Exercise: Enter the VIewController.swift file and experiment with the code:**

<https://github.com/computationalknowledge/SwiftAlertHandlers/blob/master/UsingADatePicker>

21. Click the Run button or choose Product ➤ Run. The Simulator screen appears.

22. Click the button. An alert controller appears, displaying the current date and time displayed in the Date Picker.

23. Click the button to dismiss the alert controller.

24. Change the date or time in the Date Picker. Notice another alert

controller appears, displaying the modified date/time displayed in the Date Picker.

25. Click the button to dismiss the alert controller.

26. Choose Simulator ➤ Quit Simulator to return to Xcode.

Go back to step 11 and change the .dateStyle and .timeStyle properties from .short to .medium, .long, or .full and see how this affects the way dates and times appear. If you change the .dateStyle or .timeStyle to .none, then the date or time displayed in the Date Picker will be ignored.