Swift 2.0

Xcode 7 includes a major update to Swift. Version 2.0 introduces a number of changes to the language that are not backwards compatible. As a result, the projects from the third edition of The iOS Apprentice will give errors when you try to compile them with Xcode 7.0.

Xcode includes a tool for automatically converting to Swift 2.0. The first step to upgrading your project is to open it in Xcode 7 and use the Edit → Convert → To Latest Swift Syntax menu.

Unfortunately, this won’t convert everything. This document lists what changes you need to make to the source code in order to build the projects without errors.

For a complete overview of what’s new in Swift 2.0, visit the following link: http://www.raywenderlich.com/108522/whats-new-in-swift-2

Fewer as! and as? type casts

Interoperability between Swift and Objective-C is much improved. In the past, UIKit would often return AnyObject, which then had to be cast to the correct type using as! or as?. In many cases, these casts are no longer necessary.

For example,

```swift
let paths = NSSearchPathForDirectoriesInDomains(
```

is now simply:

```swift
let paths = NSSearchPathForDirectoriesInDomains(
```

The as! [String] cast is no longer necessary because the return type of NSSearchPathForDirectoriesInDomains() has been changed to [String] instead of the meaningless AnyObject.
Another example,

```swift
func locationManager(manager: CLLocationManager!, didUpdateLocations
    locations: [AnyObject]!) {
    let newLocation = locations.last as! CLLocation
}
```

is now written as:

```swift
func locationManager(manager: CLLocationManager, didUpdateLocations
    locations: [CLLocation]) {
    let newLocation = locations.last!
}
```

The `locations` array already knows it contains `CLLocation` objects instead of `AnyObject`. Also note that the `!` exclamation marks are gone from the method’s parameters.

## Creating table view cells

The previous edition of the book dequeued prototype cells in this manner:

```swift
let cell = tableView.dequeueReusableCellWithIdentifier("ChecklistItem")
    as! UITableViewCell
```

This is now done as:

```swift
let cell = tableView.dequeueReusableCellWithIdentifier("ChecklistItem",
    forIndexPath: indexPath)
```

The `dequeueReusableCellWithIdentifier(forIndexPath)` method is guaranteed to return a valid `UITableViewCell` object (not an optional), so no type cast is necessary. This only works with prototype cells.

### `init(coder)`

The method name of `init(coder)` has changed to:

```swift
required init?(coder aDecoder: NSCoder) {
    . . .
}
```

The question mark indicates that this method can return `nil` if decoding fails. That will never happen in these tutorials, but the `NSCoding` protocol requires that you write `init?(coder)` this way.
Error handling

In most of the iOS SDK the use of NSError objects has been replaced by a new error handling mechanism, do-try-catch.

Instead of writing,

```swift
var error: NSError?
if let store = coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: storeURL, options: nil, error: &error) {
    ...
} else {
    println("Error adding persistent store at \(storeURL): \(error!)")
}
```

you now write:

```swift
do {
    try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: storeURL, options: nil)
    ...
} catch {
    fatalError("Error adding persistent store at \(storeURL): \(error)")
}
```

Note: NSError is still used for errors in the CLLocationManager delegate, both for obtaining GPS coordinates and for reverse geocoding. It’s also still used to report problems in NSURLSession's completion handler.

Other changes

println() is now named print().

The sort() method is now called sortInPlace().

find() is now called indexOf() and is no longer a function but a method. So instead of writing,

```swift
find(checklist.items, item)
```

you now do:

```swift
checklist.items.indexOf(item)
```
join() is now joinWithSeparator():

```swift
searchResult.genre = (genres as! [String]).joinWithSeparator("", ")
```

enumerate() is no longer a free function but a method, so you have to write:

```swift
for (index, searchResult) in searchResults.enumerate() {
```

The text property of UITextField is an optional of type String?. That means you need to unwrap it before you can use it. Because the contents of text will never be nil anyway, we simply write `textField.text!` to force unwrap it.

To create a UIButton from code, you used to have to write:

```swift
let button = UIButton.buttonWithType(.Custom) as! UIButton
```

Now you can simply write:

```swift
let button = UIButton(type: .Custom)
```

The stringByAppendingPathComponent() method no longer works on String objects, you have to cast to NSString first:

```swift
return (applicationDocumentsDirectory as NSString)
    .stringByAppendingPathComponent(filename)
```

**Tutorial 1: Getting Started**

The code for loading the HTML page into the About screen is changed to:

```swift
if let htmlFile = NSBundle.mainBundle().pathForResource("BullsEye",
ofType: "html") {
    if let htmlData = NSData(contentsOfFile: htmlFile) {
        let baseURL = NSURL(fileURLWithPath: 
            NSBundle.mainBundle().bundlePath)
        webView.loadData(htmlData, MIMEType: "text/html", textEncodingName: 
            "UTF-8", baseURL: baseURL)
    }
}
```

As of Xcode 7, asset catalogs no longer support Retina 4 images (-568h). The tutorial now uses the same background image for all devices. Auto Layout
constraints are used to center the background image so that it also works on the smaller iPhone 4S.

**Tutorial 2: Checklists**

In `AllListsViewController.swift` you create table view cells by hand. That used to read:

```swift
let cellIdentifier = "Cell"
var cell: UITableViewCell! =
    tableView.dequeueReusableCellWithIdentifier(cellIdentifier) as? UITableViewCell
if cell == nil {
    cell = UITableViewCell(style: .Subtitle, reuseIdentifier: cellIdentifier)
}
```

Now this is done in a separate helper method, which is a bit cleaner:

```swift
func cellForTableView(tableView: UITableView) -> UITableViewCell {
    let cellIdentifier = "Cell"
    if let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier) {
        return cell
    } else {
        return UITableViewCell(style: .Subtitle, reuseIdentifier: cellIdentifier)
    }
}
```

The date picker that lets you schedule a local notification is now created differently. Xcode 7 lets you add auxiliary views to a scene, which makes it a lot simpler to add the date picker cell in the Add/Edit Item screen. Instead of creating the `UIDatePicker` object by hand, you design the date picker cell in the storyboard and you use an outlet to refer to it.

**Tutorial 3: MyLocations**

No additional remarks.
Tutorial 4: StoreSearch

Optionals

The iOS APIs often returned implicitly unwrapped optional objects. In most cases these have been replaced by true optionals. That means you have to unwrap them first.

For example, in the completion handler of NSURLSession’s downloadTaskWithURL(), the url, response, and error parameters are now all optionals:

```swift
let downloadTask = session.downloadTaskWithURL(url, completionHandler: {
    url, response, error in
    if error == nil, let url = url, data = NSData(contentsOfURL: url),
        image = UIImage(data: data) {
        ...
    }
})
```

The if-statement first makes sure that error has no value, and then unwraps url. If that succeeds, it tries to load the contents of the file from the URL into an NSData object. And if that succeeds as well, it tries to turn that data object into a UIImage. So this one statement unwraps three optionals in a single go.

Option sets

To combine different options from an option set, Swift 1.2 used the following syntax:

```swift
autoresizingMask = .FlexibleWidth | .FlexibleHeight
```

In Swift 2.0, you no longer use the “logical or” operator to combine options, but you use array notation:

```swift
autoresizingMask = [.FlexibleWidth, .FlexibleHeight]
```

App Transport Security

As of iOS 9, unprotected http:// connections are no longer allowed, only https://. If you use an http:// URL anyway, NSURLSession will try to connect using HTTPS. If that fails, the connection is not allowed.

You can override this behavior in the Info.plist. That is necessary for this app otherwise it cannot download the artwork preview images.

The end.