iOS Dev Accelerator Week6 Day3

- Localization
- Internationalization
- Accessibility
- Unit Testing

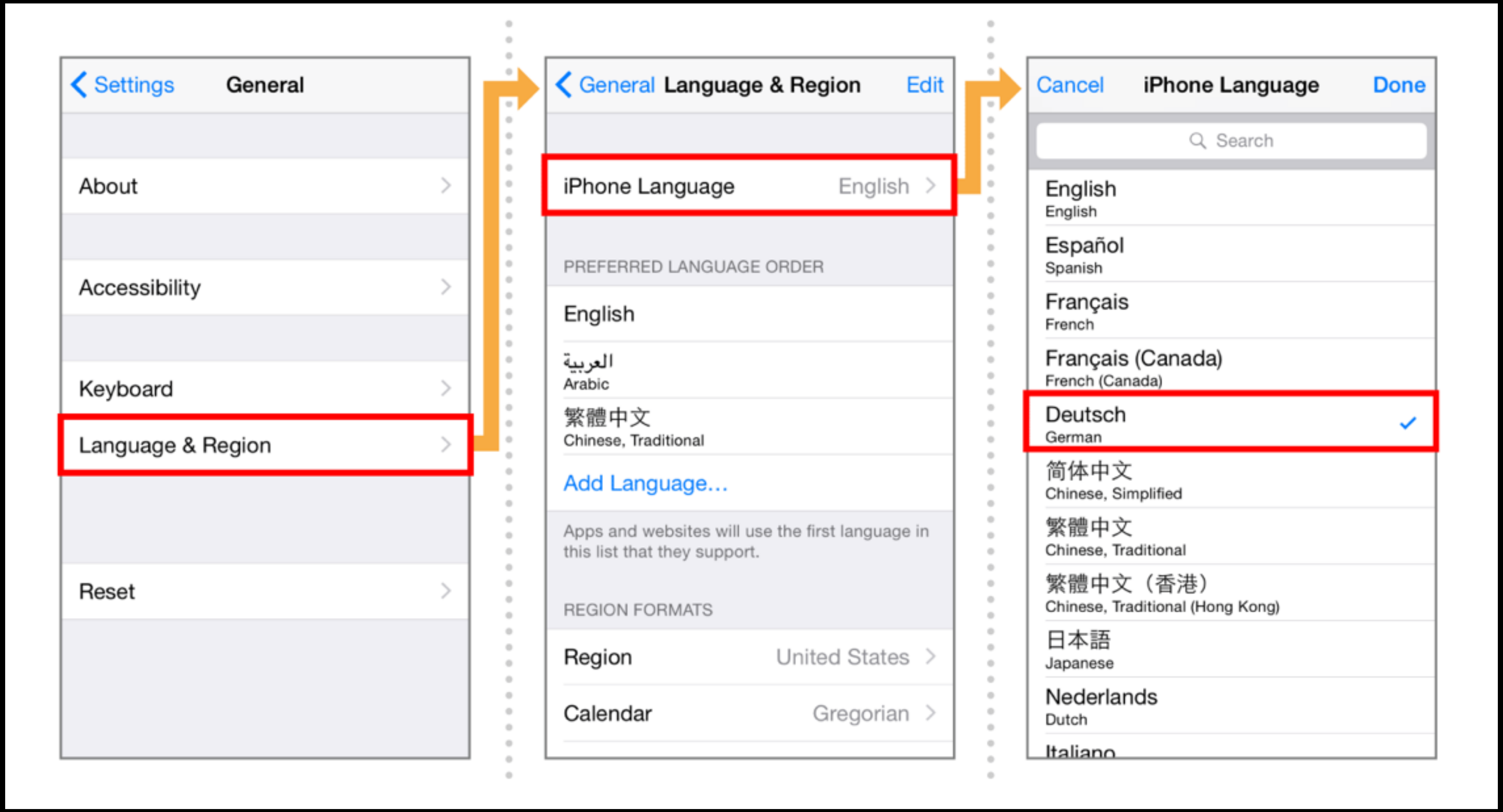
Localization and Internationalization

- **Localization** is the process of translating your app into multiple languages
- **Internationalization** is the process of creating an app that adapts to multiple languages and regions
- Internationalization is what you do; Localization is what a translator does

Switching Locales on iOS

Use the **Settings** app to change Locales on Simulator or iOS

Be careful! Difficult to return to previous Language



Base Internationalization

- Technique to separate user-facing strings from Storyboard and XIBs
- Relieves localizers from having to modify Storyboards and XIBs
- Xcode will generate language specific files for each .storyboard and .xib file
- Enabled by default in Xcode 5 and later

Expected length of Text

According to IBM's Globalization Guidelines, expect translations from English to many European languages to have 2 or 3 times to number of characters

# of chacters	Additional space
< 10	100 - 200%
11 - 20	80 - 100%
21- 30	60 - 80%
31 - 50	40 - 60%
51 - 70	31 - 40%
70+	30%

http://www-01.ibm.com/software/globalization/guidelines/a3.html

Example Translations

Language	Example Text	Length	Expansion
English	Set the power switch to 0.	26	
11 - 20	Placez l'interrupteur de tension à 0.	37	42% more
21- 30	Ponga el interruptor de alimentación de corriente en 0.	55	112% more

http://www-01.ibm.com/software/globalization/guidelines/a3.html

AutoLayout Techniques

- Remove fixed width constraints
- Use intrinsic content size
- Use leading and trailing attributes
- Pin views to adjacent views
- Test, test, and test layout changes
- Don’t set min or max window size (OS X)

Internationalizing Code

- All user facing programmatically generated Strings need to be localized
- Titles, Labels, Error messages
- Use NSLocalizedString macro
- Don’t overload keys or compose phrases from multiple keys

NSLocalizedString

- Use this for all user facing strings. Keys will be replaced by localized strings.
- Where you might have typed out

```
self.title = "Code Fellows"
```

- You'd instead type

```
  self.title = NSLocalizedString("Code Fellows", comment: "Code
Fellows title")
```

Launch Arguments

```
-NSShowNonLocalizedStrings YES

-NSDoubleLocalizedString YES

-AppleLanguages (es)
```

Workflow

- Use `NSLocalizedString` for all user facing strings
- Create Base Internationalized version for XIB and Storyboard files
- Setup AutoLayout with variable length text content in mind
- Use `NSDoubleLocalizedStrings` to test layout, without translations
- Use genstrings to generate translation files; Translate all the strings
- Test app with different languages, looking for layout issues.

Accessibility

- Set of developer tools and APIs that allows iOS app to the provide best mobile experience for customers of all needs.
- Captioning— iOS supports captioned video during playback
- VoiceOver— Screen reader to drive interface using Voice
- Speech— Read selected text aloud in multiple languages
- Guided Access— Limits iOS to running one app

Accessibility; Why?

- Increases your user base
- Allows people to use app without seeing the Screen
- It’s the right thing to do™
    - Apple is a big advocate of Accessibility.
    - First class citizen on iOS

VoiceOver

- VoiceOver is a gesture based screen reader built into iOS
- Uses a cursor or focus ring to denote current UI element
- Use touch and drag events to read whats onscreen.
- Works with all default iOS apps out of box.
- Add support to your app for participate in ecosystem

Accessible View Controllers

- An accessible app supplies information about its User Interface
- VoiceOver will “speak aloud” any on screen text. Button titles, labels, alerts
- You should supply enough contextual information for each UI alert
- In practice, this includes informational user interface elements too.
- Accessible elements might provide more information than is onscreen.

Accessibility Notifications

- Listen for accessibility notifications and trigger your own callbacks
- For example, listen for “didFinish” notification and followup the completion of VoiceOver speech with custom action
- iBook uses this to flip the page after reading last line.
- Register using defaultCenter; Notification userInfo will have two keys

```
UIAccessibilityAnnouncementKeyStringValue

UIAccessibilityAnnouncementKeyWasSuccessful
```

Accessibility Notifications

UIAccessibility**ClosedCaptioningStatusDidChange**Notification

UIAccessibility**GuidedAccessStatusDidChange**Notification

UIAccessibility**InvertColorsStatusDidChange**Notification

UIAccessibility**LayoutChanged**Notification

UIAccessibility**MonoAudioStatusDidChange**Notification

UIAccessibility**PageScrolled**Notification

UIAccessibility**ScreenChanged**Notification

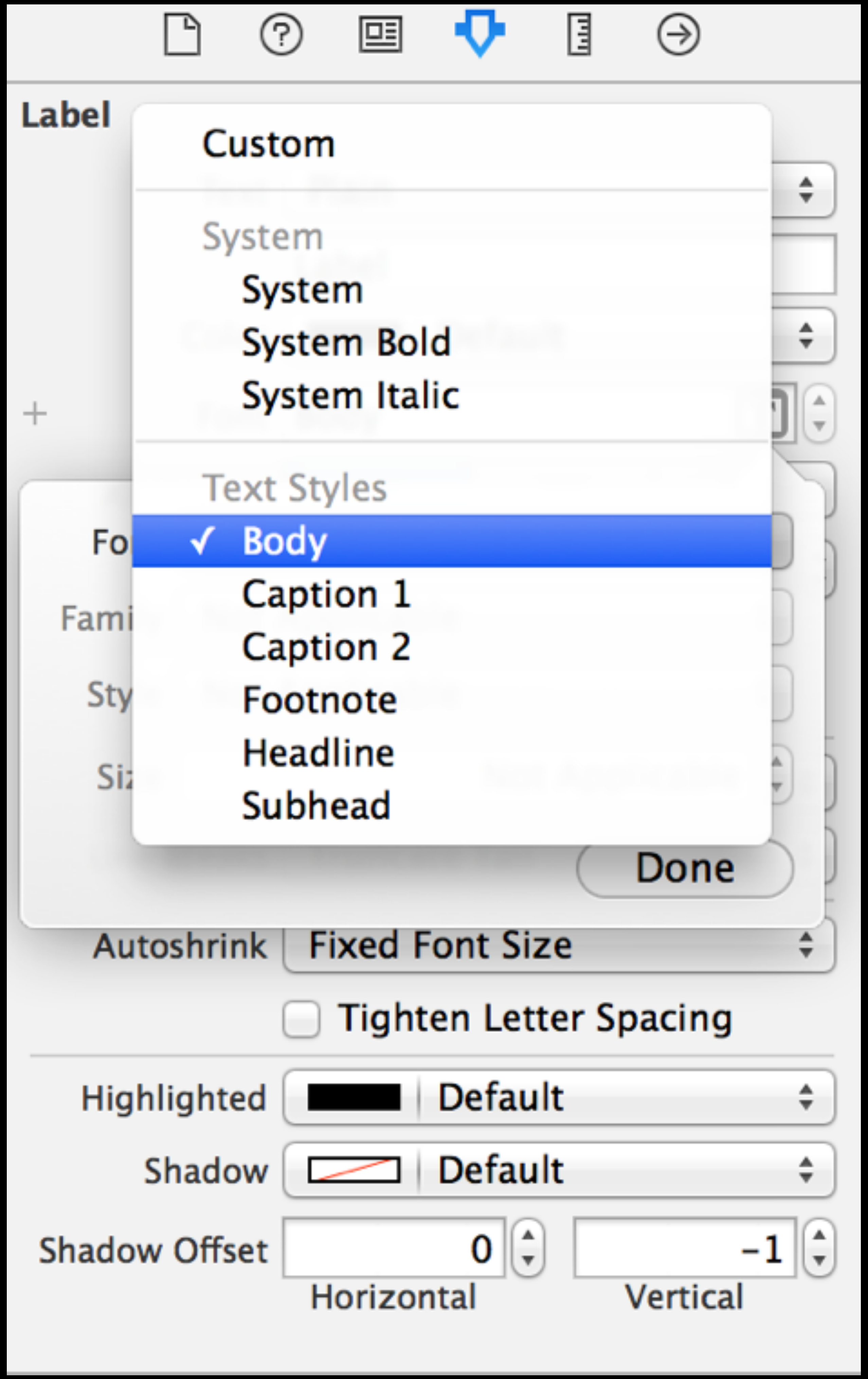
UIAccessibility**VoiceOverStatus**Changed

Dynamic Type

- Allows apps to use fonts that vary in size, according to device’s configuration
- Use the Settings app to increase, decrease font sizes
- In your app, elect to use Dynamic in Storyboard or Code
- Listen for `UIContentSizeCategoryDidChangeNotification`

Dynamic Type

- Using Interface Builder, select a text style best matching the type of text your label will show:
  - Headline, Subhead
  - Body
  - Caption 1, Caption 2
  - Footnote



Dynamic Type

- Alternatively, in Xcode, set a label font to font derived from preferred text style

```
self.nameLabel.font = UIFont.preferredFontForTextStyle(UIFontTextStyleBody)
self.title.font = UIFont.preferredFontForTextStyle(UIFontTextStyleHeadline)
```

- **`UIFont.preferredFontForTextStyle`** will return a UIFont object matching the text style

Changes to Preferred Type

- When the user changes the preferred text style, your app should respond
- This could happen while your app is running.
- Listen for `UIContentSizeCategoryDidChangeNotification` and act accordingly
- e.g, Set text style on labels, reload table view data, etc..

Changes to Preferred Type

- Add yourself as an observer of

  `UIContentSizeCategoryDidChangeNotification`

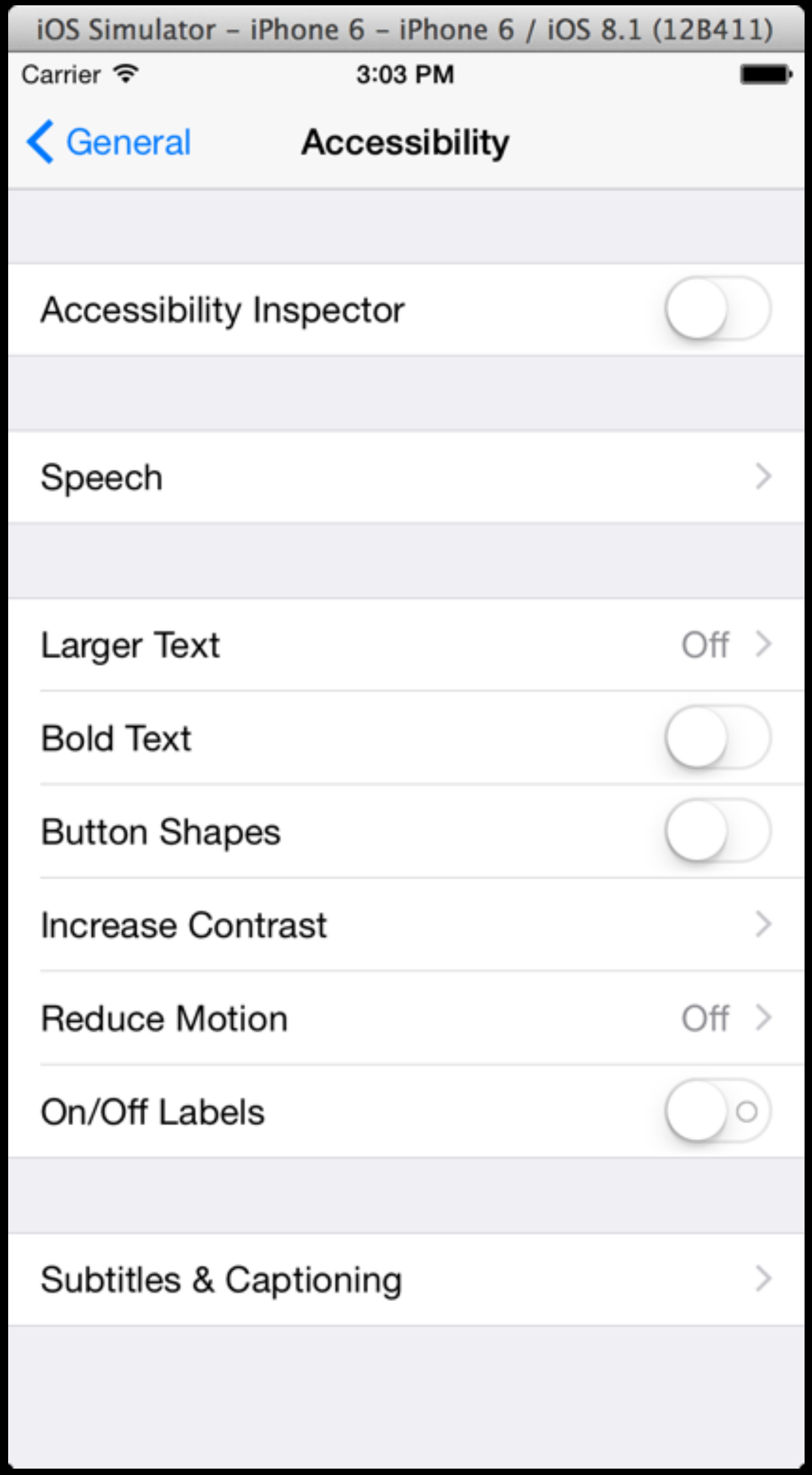
```
- (void)handleSizeChangeNotification:(NSNotification *)notification {
    self.bodyLabel.font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
    self.headlineLabel.font =
    [UIFont preferredFontForTextStyle:UIFontTextStyleHeadline];

    [self.tableView reloadData];
}
```

Accessibility Inspector

- Displays accessibility information about each accessible element in an app
- Runs in simulator and lets you see the accessibility label, value, hint, traits, and frame for each element onscreen.
- Helpful for testing the accessibility of your app during development, but..
- It is no substitute for testing your app with VoiceOver on a physical device

Accessibility Inspector


iOS Simulator – iPhone 6 – iPhone 6 / iOS 8.1 (12B411)

Carrier 3:03 PM

General Accessibility

Accessibility Inspector

Speech

Larger Text Off

Bold Text

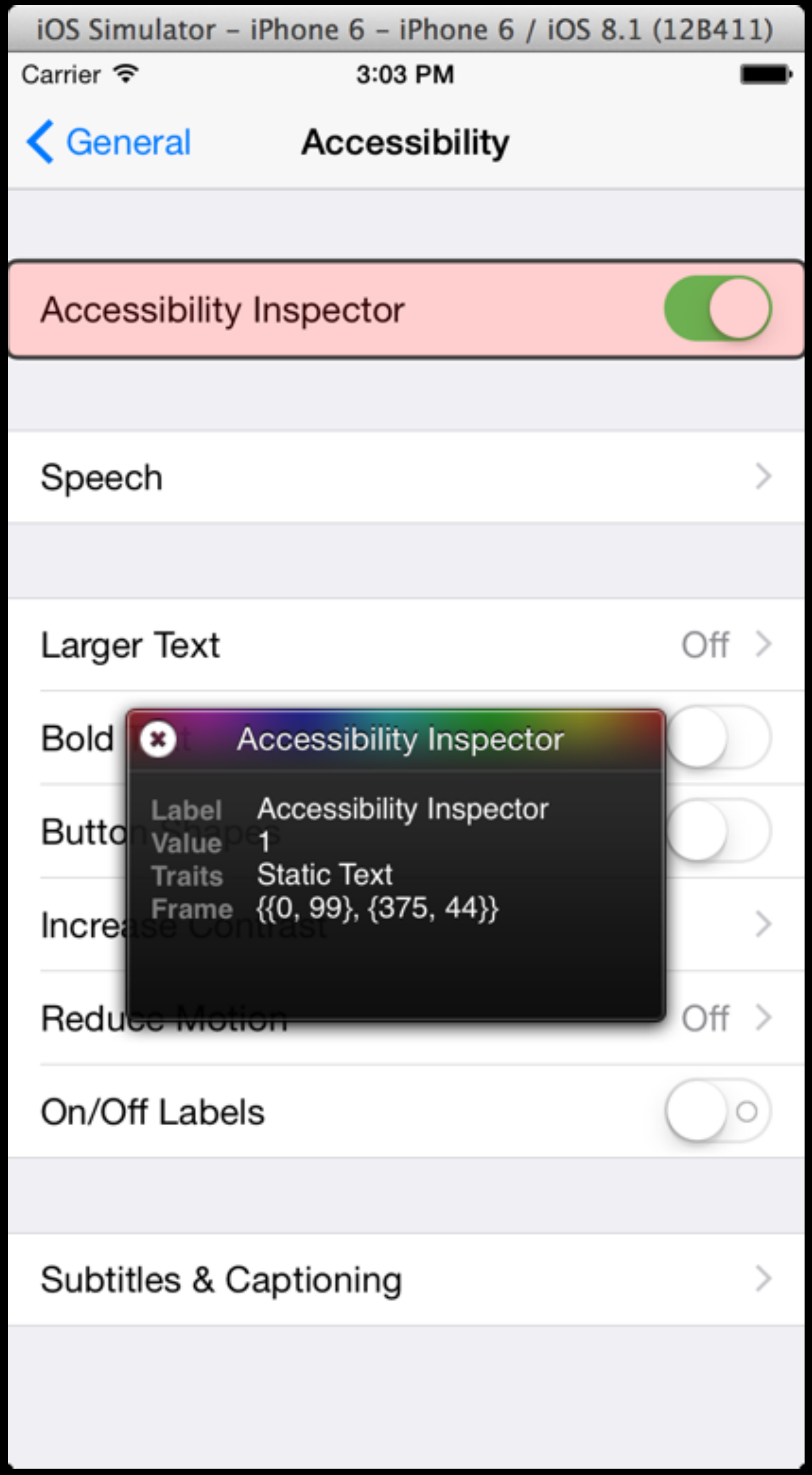
Button Shapes

Increase Contrast

Reduce Motion Off

On/Off Labels

Subtitles & Captioning


iOS Simulator – iPhone 6 – iPhone 6 / iOS 8.1 (12B411)

Carrier 3:03 PM

General Accessibility

Accessibility Inspector

Speech

Larger Text Off

Bold Text

Button Shapes

Increase Contrast

Reduce Motion Off

On/Off Labels

Subtitles & Captioning

Accessibility Inspector

Label	Accessibility Inspector
Value	1
Traits	Static Text
Frame	{{0, 99}, {375, 44}}

Unit Testing

- Mechanism for testing individual units of code to determine if they are functionally fit
- Goal is to isolate each part of the program and show that the individual parts are correct
- Provides a strict, written contract that the piece of code must satisfy
- Unit could be a module, could be individual function; up to you.
- Each test case is independent from others

Unit Testing; Why?

- Find problems early in development cycles
- Facilitates change by verifying code works after refactoring
- Simplifies integration by reducing uncertainty
- Provides “living” documentation
- Design

Terminology

- Method Stubs— A piece of code used to stand in for some other functionality. May simulate behavior or provide substitute.
- Mock Objects— Simulated objects to mimic behavior of real objects, in a controlled manner. Test object. Think crash test dummies.
- Promises— An object acting as a proxy for a results that is initially unknown.
- Refactor— Changing the underlying architecture or implementation.

OCUnit (SenTestingKit)

- Integrated into Xcode 2.1, around 2005. Apple used this developing Core Data
- Unit tests done in separate testing target.
- Each test file defines a SenTestCase subclass
- Assert style macros are used to fail tests if conditional are not met
- Older but popular Unit Testing tool; you'll see it in the wild!

XCTestCase

- Introduced in Xcode 6— testing keeps getting better!
- Xcode supports testing out of the box, using two targets/groups
  - ProductName and ProductNameTests
- Run the test target using Product > Test or ⌘U
- Tests run “outside” simulator, but you’ll see if launch briefly
  - Test results are output on your console.
- Not intended to be initialized directly; shared properties are optionals

`setup() and tearDown()`

- XCTestCase subclasses have two default methods
  - `setup()` is called before test in an XCTestCase is run
  - `tearDown()` is called as each test finishes
- Useful for creating common objects associated with all tests
- Common pattern in other unit testing frameworks too.
- Because XCTestCase is not initialized directly, any shared properties need to be optionals

setup() and tearDown()

```
class CrocodileTests: XCTestCase {
    var formatter: NSDateFormatter?

    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before
        // the invocation of each test method in the class.

        self.formatter = NSDateFormatter()
        self.formatter?.dateStyle = .FullStyle
        self.formatter?.timeStyle = .ShortStyle
    }

    override func tearDown() {
        // Put teardown code here. This method is called after
        // the invocation of each test method in the class.
        super.tearDown()
    }
}
```

Functional Testing

- Each method that begins with “test” is recognized as an actual test to run
- Running a test will evaluate any assertions within, and determine if the test should **Pass** or **Fail**

```swift
func testOnePlusOneEqualsTwo() {
    XCTAssertEqual(1 + 1, 2, "One plus one should equal two")
}
```

Asserting the truth

- Assertions are about what *you* expect to have. They are assertions about the state of a running program

XCTAssertEqualObjects	XCTAssertGreaterThan	XCTAssertNil
XCTAssertNotEqualObjects	XCTAssertGreaterThanOrEqual	XCTAssertNotNil
XCTAssertEqual	XCTAssertLessThan	XCTAssertTrue
XCTAssertNotEqual	XCTAssertLessThanOrEqual	XCTAssertFalse

Asserting the truth

- For Boolean tests, use `XCTAssertTrue` and `XCTAssertFalse`
- When testing for equality, use `XCTAssertEqual` or `XCTAssertNotEqual`
- Use `XCTAssetNil` or `XCTAssetNotNil` for asserting existence of nil
- Use `XCTFail` to fail all the time

Performance Testing

- Xcode 6 provides the ability to performance test a piece of test

```
func testDateFormatterPerformance() {
    let dateFormatter = NSDateFormatter()
    dateFormatter.dateStyle = .LongStyle
    dateFormatter.timeStyle = .ShortStyle

    let date = NSDate()

    self.measureBlock() {
        let string = dateFormatter.stringFromDate(date)
    }
}
```