

iOS Dev Accelerator

Week 6 Day 2

- UIDatePicker & UIPickerView
- Date Handling
- Advanced Fetching
- Internationalization & Localization

UIDatePicker

UIDatePicker

- “The UIDatePicker class implements an object that uses multiple rotating wheels to allow users to select dates and times”
- Can also be used as a countdown timer.
- Has a date property that simply returns the current selected date in the date picker
- You can also hook up an IBAction that fires anytime the date has changed
- Has an intrinsic width of 320, you can squeeze it down to 240 and still see everything though
- Has options in the storyboard attribute inspector

Demo

UIPickerView

- UIPickerView is similar to a DatePicker, but allows you to fully customize what the picker displays instead of just Dates & Times.
- Basic Workflow for a UIPickerView:
 - Drag out your picker view from storyboard and make an outlet, or create the picker view in code.
 - Set your view controller to be the pickers datasource
 - Implement:
 - `numberOfComponentsInPickerView:`
 - `pickerView:numberOfRowsInComponent:`
 - `pickerView:titleForRow:ForComponent:`

NSDate & NSDateFormatter

Dates & Time

- There are 3 primary classes you will use when representing time in your apps:
 - NSDate - a representation of a single point in time
 - NSCalendar - a representation of a particular calendar type (Buddhist, Chinese, Hebrew, etc)
 - NSDateComponents - a representation of particular parts of a date, like an hour, minute, day, year, etc.

Dates

- NSDate represents a single point in time.
- Date objects are immutable once created
- The standard unit of time for date objects is floating point value typed as NSTimeInterval expressed in seconds.
- NSDate computes times as seconds relative to an absolute reference time: the first instant of January 1, 2001, GMT.
- Dates before that time are stored as negative values. Dates after stored as positive.

Creating Date Objects

- To get a date that represents the exact current time:
 1. just init a NSDate instance OR
 2. use the class method .date on NSDate
- To create a specific date:
 1. Use one of NSDate's initWithTimeInterval init methods
 2. Use an NSCalendar and date components

Demo

Basic Date Calculations

- There are many ways to compare dates:
 - isEqualToDate:
 - compare:
 - laterDate:
 - earlierDate:
- and you can also use the method `timeIntervalSinceDate:` on `NSDate` to get the time interval from 2 dates

Date Calculations

- Use the `dateByAddingComponents:toDate:options:` on a calendar to add components (hours, minutes, days, years) to an existing date. They can be negative values to take time away as well.

Calendars

- You use calendar objects to convert between times and date components (years, days, minutes, etc)
- NSCalendar provides an implementation for many different types of calendars.
- Calendar types are specified by constants in NSLocale.
- the method `currentCalendar` returns the user's preferred locale.

Date Components

- You can represent a component of a date using the NSDateComponent class.
- Use methods `setDay:`, `setMonth:`, and `setYear:` to set those individual components.

New date convenience methods

- `[calendar component:NSCalendarUnitHour fromDate: [NSDate date]]` - returns a single component of your specification from a date
- `[calendar componentsInTimeZone: fromDate:]` returns a date shifted to the time zone of your specification.
- `[calendar components: fromDateComponents :toDateComponents :options]` returns the difference between two NSDateComponents instances.

New date comparison methods

- `isDateInToday`: returns true if date is today
- `isDateInTomorrow`:
- `isDateInYesterday`:
- `isDateInWeekend`:
- `isDate: inSameDayAsDate`:
- `isDate: equalToDate: toUnitGranularity`:

Demo

NSDateFormatter

- NSDateFormatter is a subclass of NSFormatter, which has a wide range of specialized subclasses:
 - NSNumberFormatter
 - NSDateFormatter
 - NSByteCountFormatter
 - NSDateFormatter
 - NSDateComponentsFormatter
 - NSDateIntervalFormatter
 - NSEnergyFormatter
 - NSMassFormatter
 - NSLengthFormatter
 - MKDistanceFormatter

NSDateFormatter

- You use an NSDateFormatter to get text representations of both dates and times
- You can set the NSDateFormatter's style to set the desired style of date:

1. *NSDateFormatterNoStyle*: No style.
2. *NSDateFormatterShortStyle*: 12/18/13
3. *NSDateFormatterMediumStyle*: Dec 18, 2013
4. *NSDateFormatterLongStyle*: December 18, 2013
5. *NSDateFormatterFullStyle*: Wednesday, December 18, 2013

*sourced from <http://gtiapps.com/?p=1086>

Custom style

- If none of the pre-baked styles fit your needs, you can create a custom date format using the `setDateFormat:` method.
- This takes in a format string that follows a specific pattern:

- *yyyy*: Year using four digits, e.g. 2013
- *yy*: Year using two digits, e.g. '13
- *MM*: Month using two digits, e.g. 05
- *MMMM*: Month, full name, e.g. March
- *dd*: Day with two digits, e.g. 14
- *EEEE*: Day of the week, full name, e.g. Friday
- *EEE*: Day of the week, short, e.g. Mon

For example, the next format string:

`EEEE, dd MMM yyyy`

represents the following formatted date (when this quick tutorial was written):

Thursday, 07 March 2013

Demo

Advanced Fetching*

*Sourced from NSHipster

More Fetching

- Yesterday we learned what core data is, how to use it, how to insert objects, and how to do a very simple fetch.
- Our fetch just retrieved all the objects for a certain Entity.
- Thats great, but what if we wanted to get more specific with our query?
- Sometimes we don't necessarily always want ALL the objects from a certain entity, that could literally be millions of objects!
- We can configure our fetches to be much more specific. Lets take a look.

When to Fetch?

- Sometimes you don't need to do a fetch, you already have the data you need.
- Relationships are a much more efficient way to traverse your object graph vs fetch requests.
- You typically only need to fetch if
 - a) you haven't done a fetch yet
 - b) you need to search your object graph for specific items that match specific criteria

Creating Fetch Requests

- There's actually 4 different ways for fetch requests to be created:
 1. Using its plain initializer
 2. Using its initializer that takes in an entity name that you want to fetch against
 3. Accessing a fetch request that we pre-made in the MOM file (cool)
 4. Same as 3, but there is a method you can use to pass in extra variables for the fetch to be used in the predicate

Creating Fetch Requests in the MOM file

- Creating a pre-defined fetch request in the MOM is safe & easy!
- Its usually best to do this if you find yourself coding the same fetch over and over again.
- Pretty straight forward process

Demo

NSPredicate

- Query language describing how data should be fetched and filtered
- Describes logic conditions to use when searching collections
- Used in Core Data, but can also be used to filter regular foundation classes like arrays and sets.
- Familiar with SQL? Think of the WHERE clause

NSPredicate String syntax

- When creating a predicate, you use a predicate string to communicate to the database what exactly you are looking for.
- The predicate string parser is whitespace insensitive, case insensitive, and supports nested parenthetical expressions.
- The parser does not do any semantic type checking, so typos will cause run time errors.

```
NSString *lastNameSearchString = @"Wilson";
```

```
[NSPredicate predicateWithFormat:@"lastName like %@", lastNameSearchString];
```

This evaluates to fetching all person entities with the attribute lastName like “Wilson”

NSPredicate

- Use predicateWithFormat for building simple predicates
- You can use %@ to substitute for an object value (usually a string)
- You can use %K to substitute for an attribute or attributes (via keypath)

```
[NSPredicate predicateWithFormat:@"name = 'Clarus'"];  
[NSPredicate predicateWithFormat:@"species = %@", @"Dogcow"];  
[NSPredicate predicateWithFormat:@"%K like %@", attributeName, attributeValue];
```

(In the example above, if you had done “%@ like %@”, the query would be incorrect because %@ substitutes always add quotes to the string value. You never want quotes around the attribute you are querying against.)

- Predicates will traverse key paths in a query, which is super awesome

```
[NSPredicate predicateWithFormat:@"department.name like %@", department];  
[NSPredicate predicateWithFormat:@"ANY employees.salary > %f", salary];
```

NSPredicate Comparisons

- `=`, `==`: the expression on the left is equal to the right-hand expression
- `>=`, `=>`: the left hand expression is greater than or equal to the right hand expression (and vice versa for `<=`, `=<`, and also `<`, `>`)
- `!=`, `<>`: the left hand expression is not equal to the right hand expression
- `BETWEEN`: the left hand expression is between, or equal to either of, the values specified in the right hand side. The right hand side is a two value array giving upper and lower bounds. Arrays denoted with `{}`

NSPredicate String Comparisons

- BEGINSWITH: the left-hand expression begins with the right-hand expression
- CONTAINS: the left-hand expression is contained in the right-hand expression
- ENDSWITH: the left-hand expression ends with the right-hand expression
- LIKE: the left-hand expression equals the right hand expression: ? and * are allowed as wildcard characters, where ? matches 1 character and * matches 0 or more characters
- MATCHES: the left-hand expression equals the right-hand expression using regex-style comparisons.

NSPredicate Relational Operators

- ANY, SOME: Specifies any of the elements in the following expression
- ALL: Specifies all of the elements in the following expression
- NONE: Specifies none of the elements in the following expression
- IN: Equivalent to an SQL IN operation, the left-hand side just appear in the collection specified by the right-hand side.

NSPredicate Basic Compound Predicates

- AND, &&: Logical AND
- OR, ||: Logical OR
- NOT, !: Logical NOT

```
[NSPredicate predicateWithFormat: @"room.hotel.name == %@ AND  
dateFrom <= %@ AND dateTo >= %@", @"Solid Stay",  
self.startPicker.date, self.endPicker.date];
```

NSPredicate and self

- You will see the word self used in NSPredicates
- self refers to each entity we are querying against

```
[NSPredicate predicateWithFormat:@"self == %@", someObject];
```

Demo

NSPredicate variable substitution

- Variable substitution is supported too, using \$ sign for variable names

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"id = $user"];  
NSDictionary *substitutions = @{@"user": currentUser}
```

```
NSPredicate *filter = [predicate  
predicateWithSubstitutionVariables:substitutions];
```

Searching Collections

- Collection classes can be searched and filtered using `NSPredicate`*
 - `NSFetchRequest` exposes a predicate property, for filtering entities
-
- `(NSArray *)filteredArrayUsingPredicate:(NSPredicate *)predicate;`
 - `(NSSet *)filteredSetUsingPredicate:(NSPredicate *)predicate;`

*Most collection classes; `NSDictionary` and `NSIndexSet` don't directly accept an `NSPredicate`

NSFetchRequest resultType

- NSFetchRequest has a property named resultType.
- By default it is set to NSManagedObjectResultType
- Here is all the types:
 - NSManagedObjectResultType: returns the managed objects
 - NSCountResultType: returns the count of all the objects that match the fetch
 - NSDictionaryResultType: This returns the results back in a dictionary
 - NSManagedObjectIDResultType: Returns the ObjectIDs instead of full-fledged managed objects

Fetching the count

- NSCountResultType lets you get back the count of objects that a fetch will retrieve.
- It returns a an array containing an NSNumber, which is the count.
- Its a little weird, but it works!

Demo

Internationalization & Localization

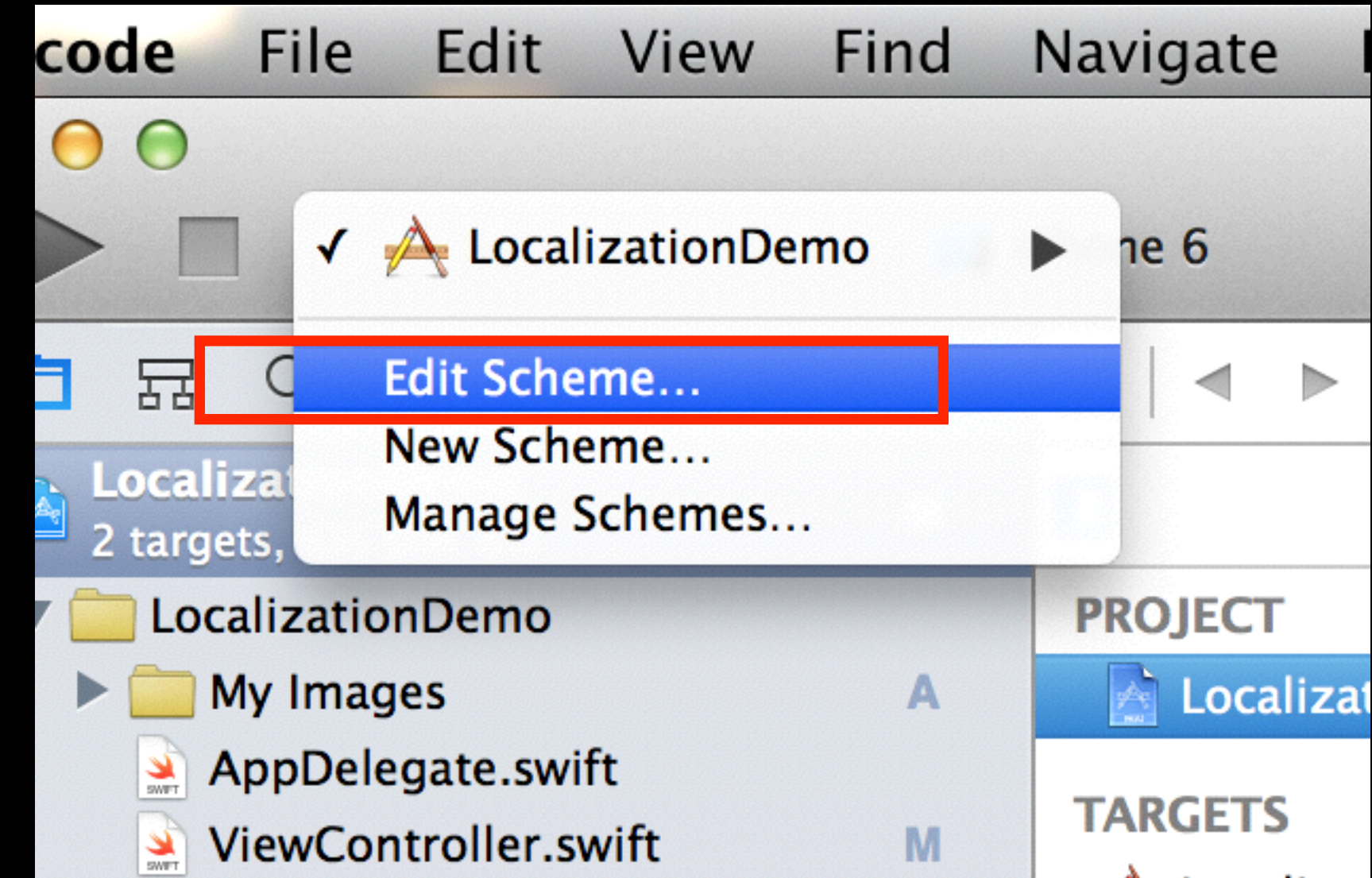
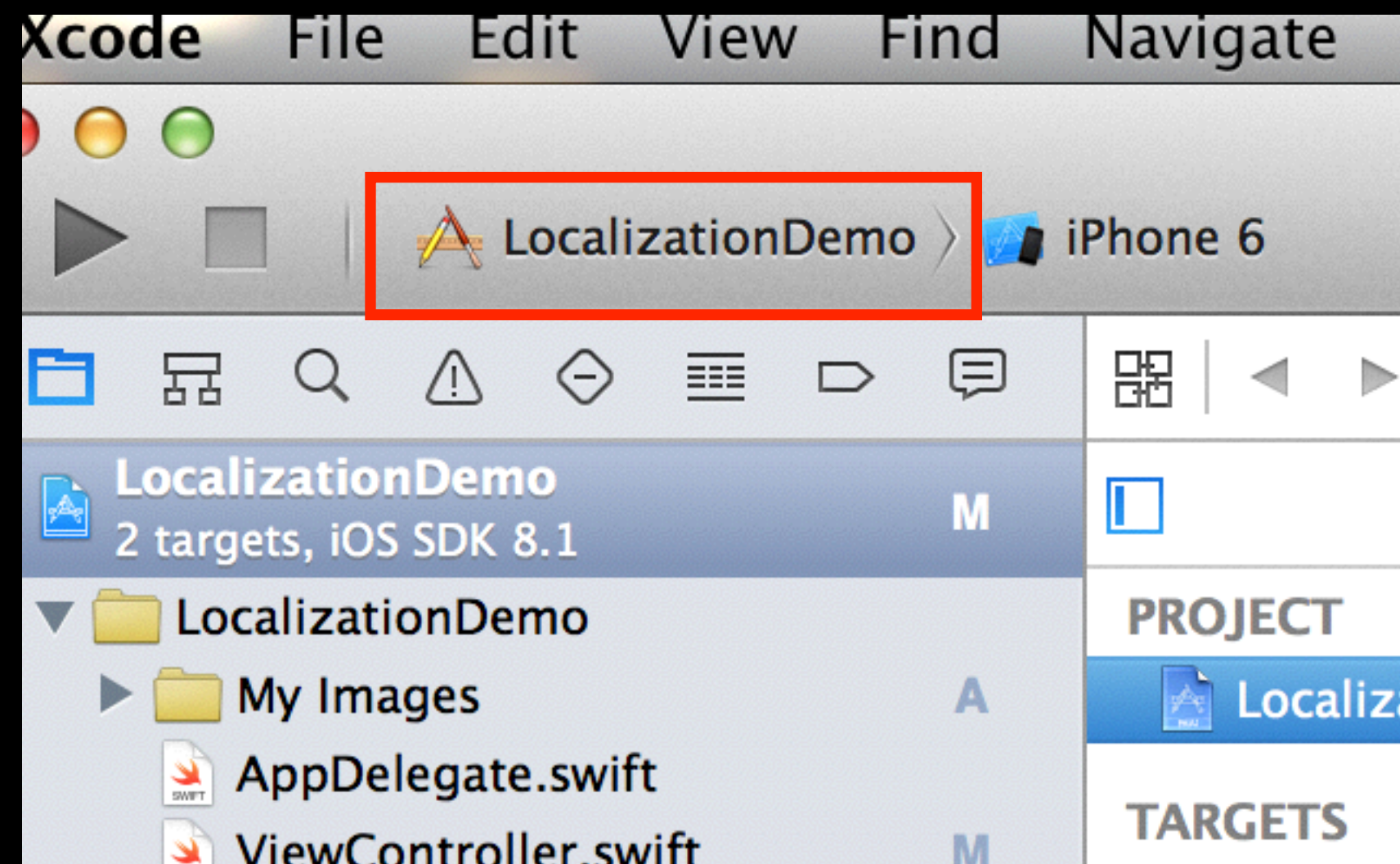
“It’s like flossing, everyone knows
they should do it, but they probably don’t do it” - NSHipster

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 (or you with google translate if you're cheap)
- Most people just refer to it all as localization. But its good to know the difference.

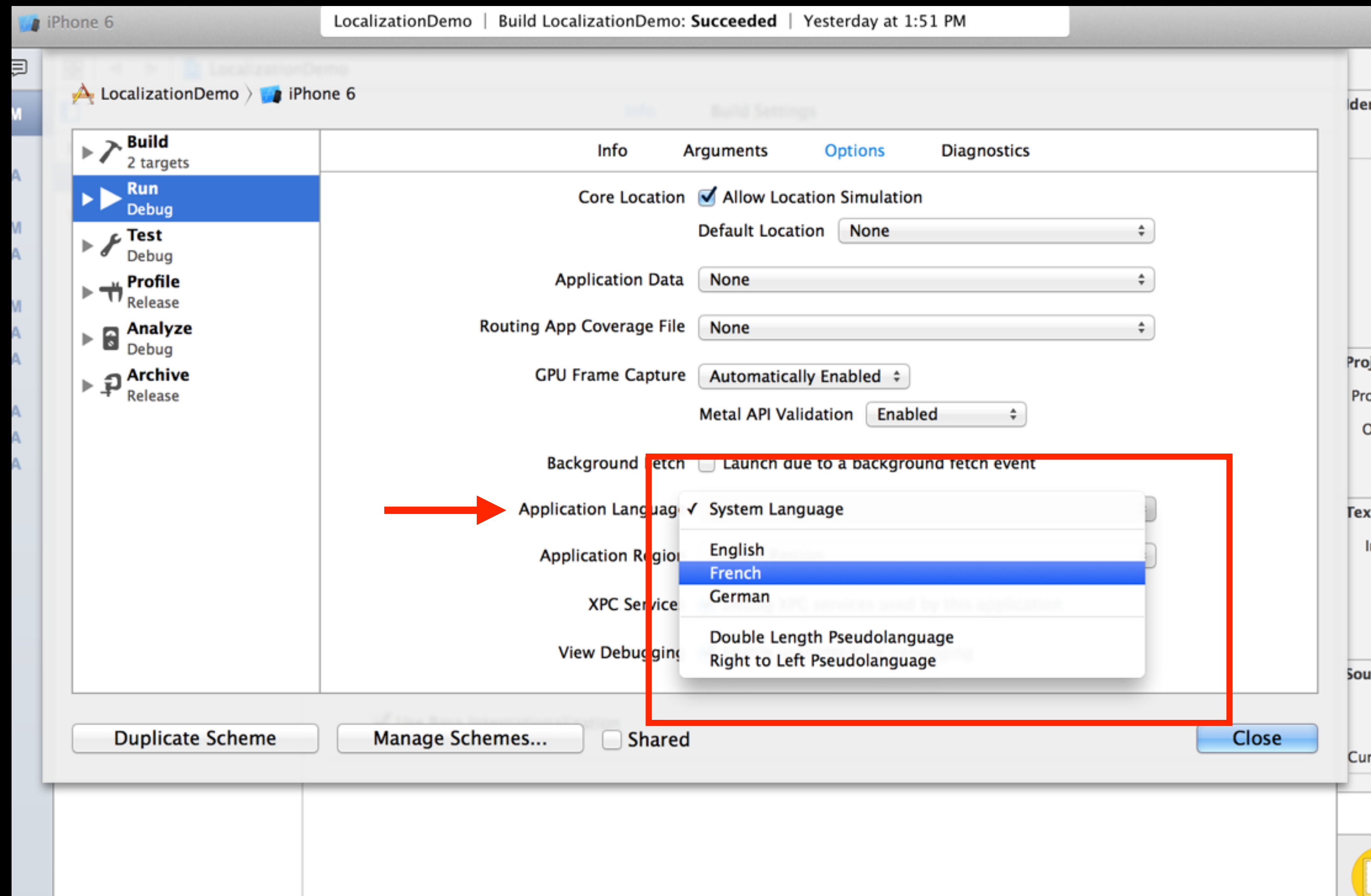
Switching Locales on the simulator via Xcode

3 easy steps!



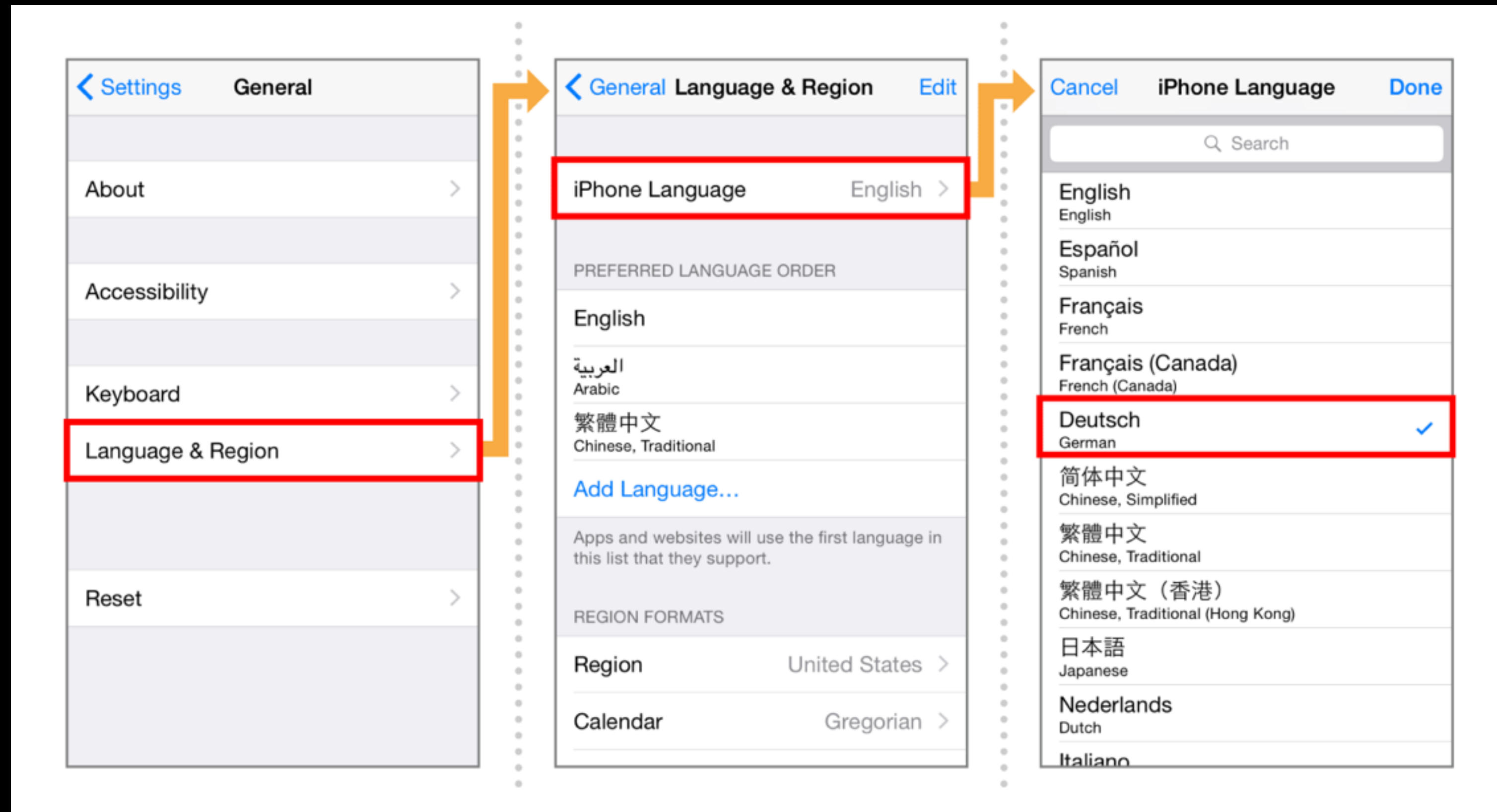
Switching Locales on the simulator via Xcode

last step



Switching Locales on a Device

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



Be careful! Difficult to return to previous Language because everything is not in English anymore

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
- **Just go to your Project's Info screen, and hit the + button under localizations to add a language.**

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%

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

AutoLayout Techniques

- Remove fixed width constraints
- Use intrinsic content size when you can
- 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

NSString

- Use this for all user facing strings. Keys will be replaced by localized strings.

- Where you might have typed out

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

- Instead you type:

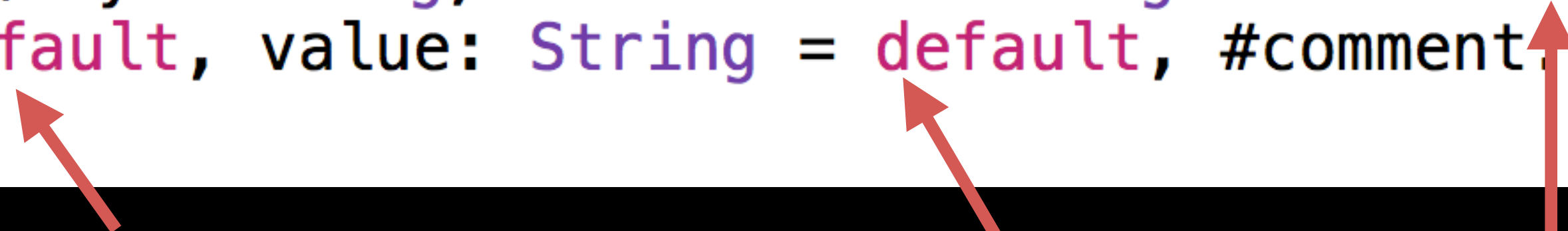
```
self.myLabel.text = NSLocalizedString("Hello",  
                                     tableName: nil,  
                                     bundle: NSBundle.mainBundle(),  
                                     value: "nope",  
                                     comment: "This is a greeting")
```

Well this sucks! Actually we can make it much shorter....

NSLocalizedString

- So NSLocalizedString is a global function in Foundation:

```
/// Returns a localized string, using the main bundle if one is not
/// specified.
func NSLocalizedString(key: String, tableName: String? = default,
    bundle: NSBundle = default, value: String = default, #comment:
    String) -> String
```



- The parameters marked with = default can actually be completely dropped when calling the function!

```
self.myLabel.text = NSLocalizedString("Hello", comment: "This is a greeting")
```

String file

- A file you can generate with Xcode (or a tool called genstring - better option)
- Must be named localizable.strings
- Works like a dictionary, key-value pairing
- You need this to localize strings that aren't set via storyboard or xibs.
- **Keep in mind you only need to translate strings that will be seen by the user!!**
- After creating your set of string files, you can use the NSLocalizedString() macro to reference the strings you defined in the key-value pairings.
- Every line must end with a semi colon;

genstrings

- genstrings is command line tool installed on your mac
- to use it, cd into the directory containing the swift source code (aka swift files) in terminal
- next, run the command `genstrings *.swift`
- this will parse through your code, looking for any `nslocalizedstrings` and generate a `localizable.strings` file with all your localized strings complete with the comments as well

String file

English/base file

```
1  /*
2   Localizable.strings
3   localtest
4
5   Created by Bradley Johnson
6   Copyright (c) 2015 BPJ. All rights reserved.
7  */
8
9
10 "Hello" = "Hello";
11 "Goodbye" = "Goodbye";
12 "Cool" = "Cool";
```

French file

```
1  /*
2   Localizable.strings
3   localtest
4
5   Created by Bradley Johnson
6   Copyright (c) 2015 BPJ. All rights reserved.
7  */
8
9
10 "Hello" = "Bonjour";
11 "Goodbye" = "Au Revoir";
12 "Cool" = "Frais";
```


Workflow

- Use `NSString` for all user facing strings
- Create Base Internationalized version for XIB and Storyboard files
- Setup AutoLayout with variable length text content in mind
- Create String files for each language supported
- Test app with different languages, looking for layout issues.

Overall Internationalization Guide

1. From the beginning, use NSLocalizedString Macro for any user facing strings
2. Besides that, wait to do internationalization until the last possible moment. This way your storyboard, nibs, and all user facing strings should already be created.
3. Once ready, enable the languages you want to support, which creates string files for all storyboard/nibs.
4. Then run genstrings in your directory to generate the strings file, and import that into xcode and create version of it for all languages to support.
5. Pay someone (or do it yourself if its simple phrases/words) to do the translations in one go.

Demo

Accessibility

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?

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

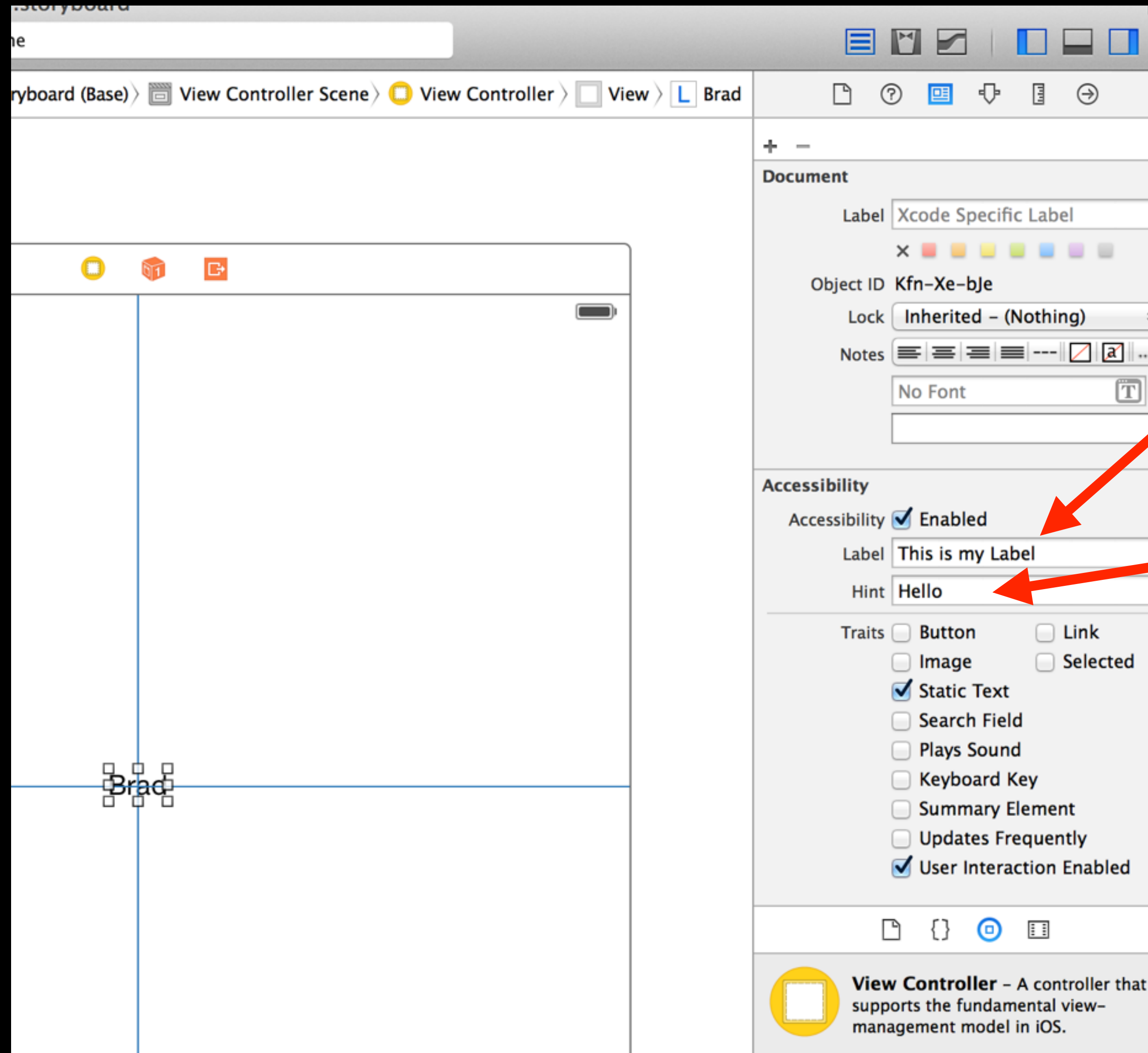
VoiceOver

- VoiceOver is a gesture based screen reader built into iOS that lets you use the phone if you can't see the screen.
- 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 today

VoiceOver Details

- Almost all Apple created UI elements have accessibility functionality built into them.
- Voice over functionality is very easy to setup, so theres no excuse!
- For the most part you can just do it all via storyboard, and doing it in code is also as easy as a few lines of code.

VoiceOver Storyboard



This is what the label will speak to the user, if its blank the system will just speak what the label actually says

The hint will play after a short pause if the user stays selected on the UI element

These properties can be accessed via code, like most other storyboard properties

Accessibility Attributes

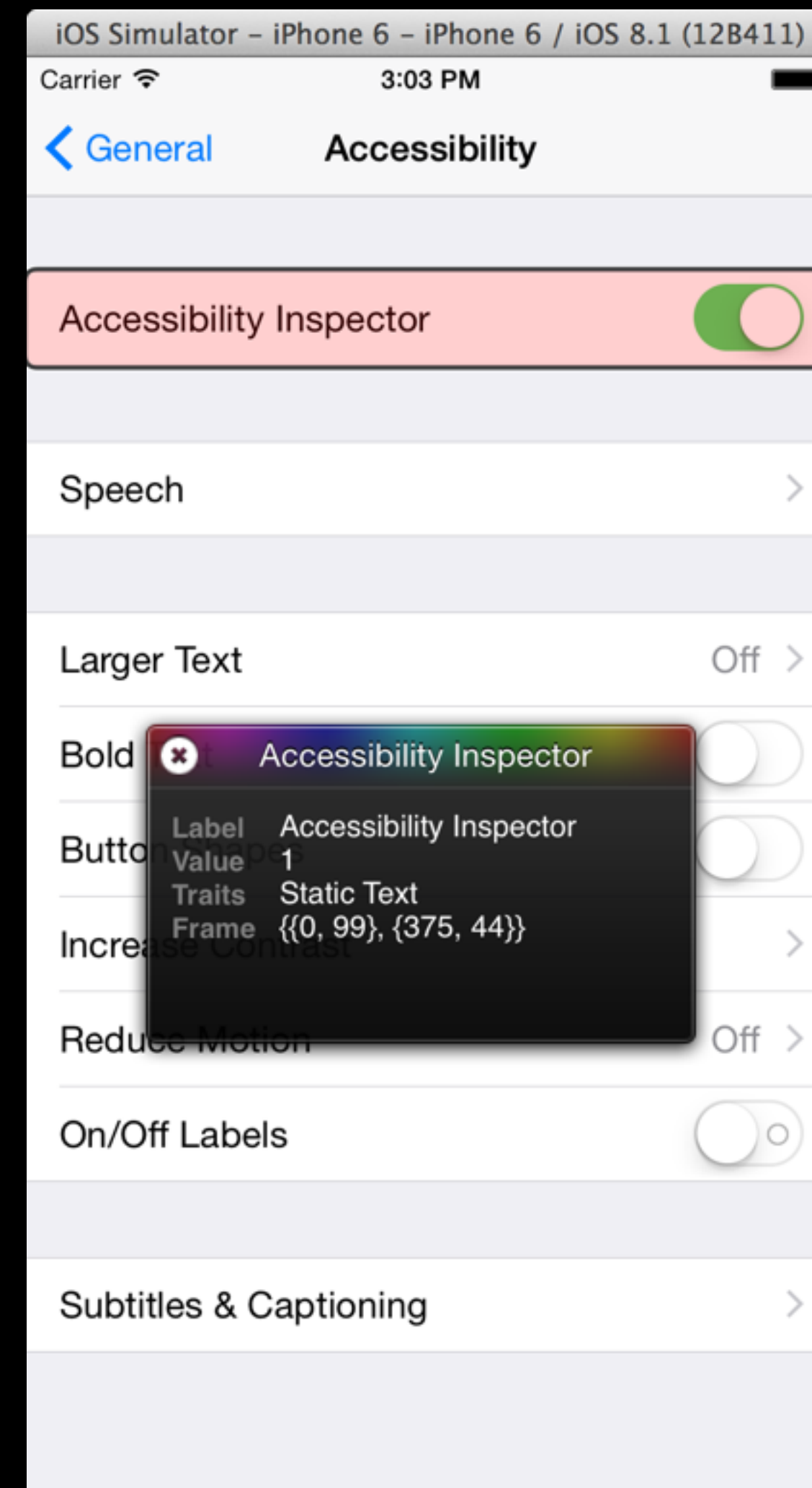
- You just saw 'Label' and 'Hint', there are 3 more:
- Traits: a combo of one or more individual traits, each describing a single aspect of an elements state, like if its a button or if user interaction is enabled or disabled on it.
- Frame: The frame of the element in screen coordinates (just like regular frame)
- Value: Used to store the value if that is applicable, like a slider "50%", or a switch "on/off"

VoiceOver Demo

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's pretty good but you will still want to test your app with VoiceOver on a physical device

Accessibility Inspector



Demo