Stanford CS193p

Developing Applications for iOS
Winter 2015

Today

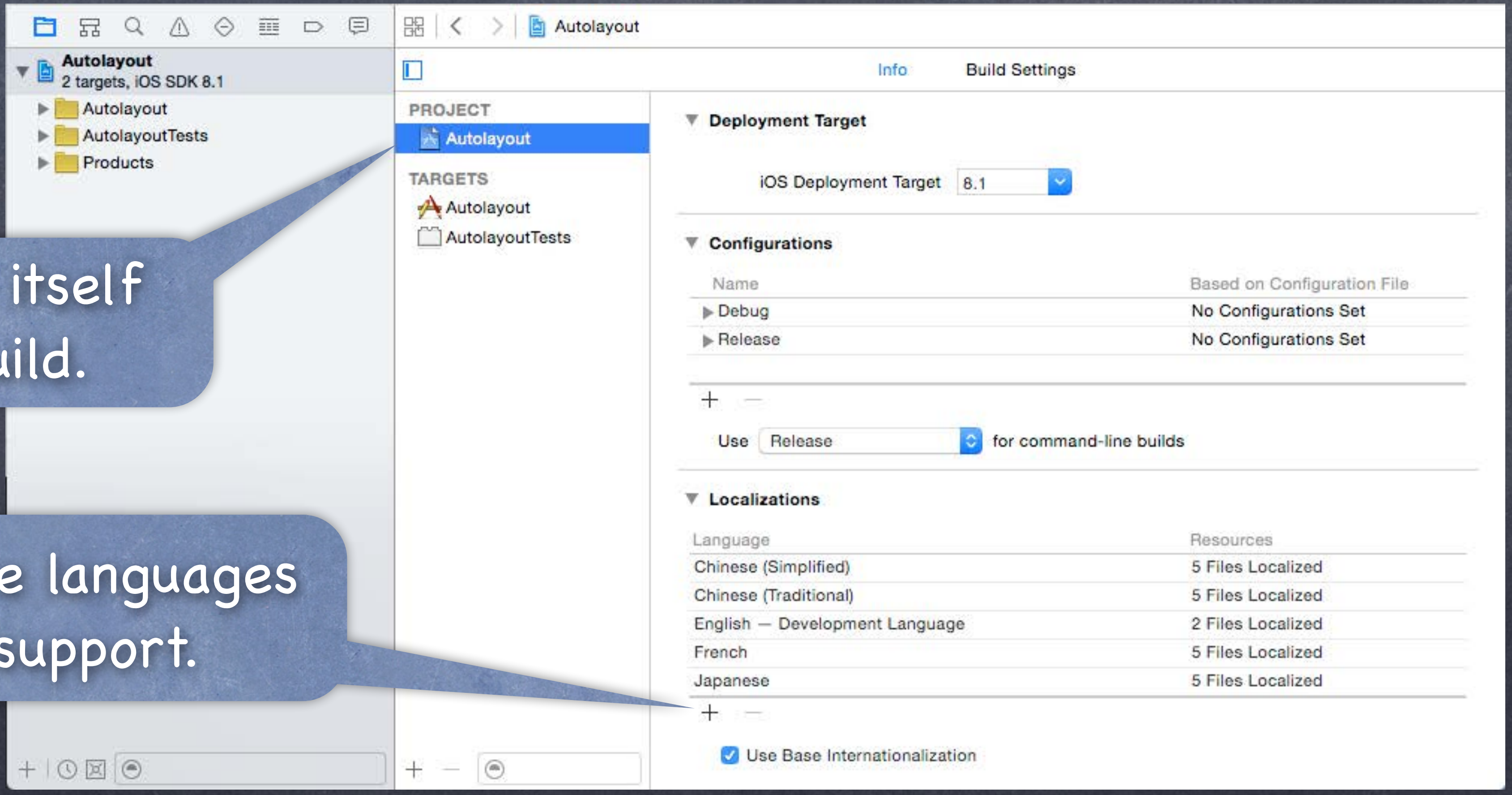
- Internationalization and Localization
  Making your app marketable around the world
- Settings
  Adding UI to the General Settings application

Internationalization

- Two steps to making international versions of your application
  - Internationalization (i18n)
  - Localization (l10n)
- Internationalization
  - This is a process of making strings externally editable (from storyboard or code).
  - It also involves using certain "formatting" classes for things like dates, numbers, etc.
  - You (the developer) get to do this work.
- Localization
  - A process of translating those externalized strings for a given language.
  - You usually hire a localization company to do this work.

Internationalization

- Storyboards are localized by changing the strings inside only
  - And we rely on Autolayout to make it all look nice.
- First step: Registering Localizable Languages
  - Go to the Project pane in Xcode (top in Navigator), then Info tab to add Localizations.



Deployment Target

iOS Deployment Target 8.1

Configurations

Name	Based on Configuration File
Debug	No Configurations Set
Release	No Configurations Set

Use Release for command-line builds

Localizations

Language	Resources
Chinese (Simplified)	5 Files Localized
Chinese (Traditional)	5 Files Localized
English — Development Language	2 Files Localized
French	5 Files Localized
Japanese	5 Files Localized

[✓] Use Base Internationalization

You must inspect the project itself here, not the Target you build.

Click this + to add more languages that you intend to support.

Internationalization

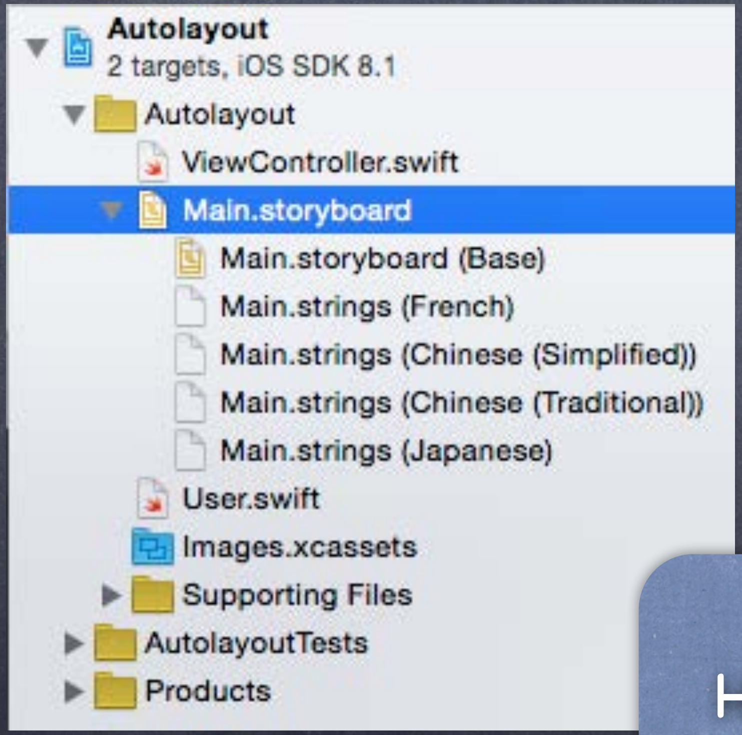
- Storyboards in the Navigator will now support "localizations"

  If you select a storyboard in the navigator, you'll see the localizations there …

  … and in the File Inspector (leftmost tab in Utilities area)

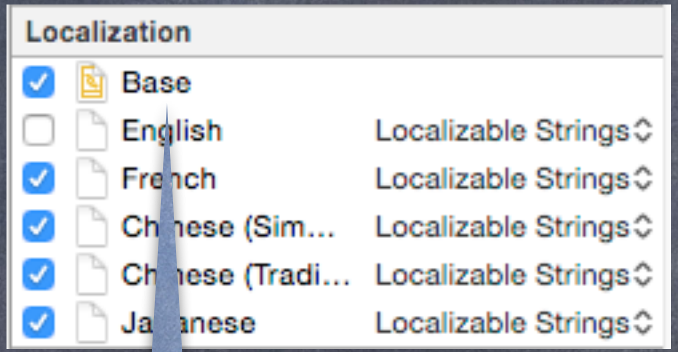
  That's all there is to it. Your storyboards are now localizable (you've internationalized them).

Navigator



Autolayout
2 targets, iOS SDK 8.1
Autolayout
ViewController.swift
Main.storyboard
Main.storyboard (Base)
Main.strings (French)
Main.strings (Chinese (Simplified))
Main.strings (Chinese (Traditional))
Main.strings (Japanese)
User.swift
Images.xcassets
Supporting Files
AutolayoutTests
Products

File Inspector



Localization

✓	Base	
	English	Localizable Strings
✓	French	Localizable Strings
✓	Chinese (Sim…	Localizable Strings
✓	Chinese (Tradi…	Localizable Strings
✓	Japanese	Localizable Strings

Base is the "language" in the storyboard.
Hopefully nothing in there ever shows up in your UI
for users who don't speak the same language as you do.
Don't uncheck Base though!

Internationalization

- What about strings not in storyboards?

  i.e., literal strings "string"

  We replace them with calls to `NSLocalizedString()` …

```
func NSLocalizedString(
     key: String, // usually this is the literal string (in the developer's language)
     tableName: String, bundle: NSBundle, // can be used to organize strings (not required)
     value: String, // value to use if key is not found in a localization (defaults to key)
     comment: String // instructions for the person translating this string
) -> String
```

- Most of the parameters have good defaults

  Therefore most of the time, we just use …

```
let localString = NSLocalizedString("Foo", comment: "This string is shown when …")
```

  The `bundle` defaults to `NSBundle.mainBundle()` and `tableName` to "`Localizable`"

Formatted Strings

- Strings created from a format (`println`-like) string

  Any string you create for the UI with a format string, should use this method …

```
let string = String.localizedStringWithFormat(
     NSLocalizedString("the result is %g", comment:"gives numeric result to the user"),
     theNumber
)
```

- Localizers can specify the order of arguments like this …

```
"result number %d is %d" = "%2$d is the result at position %1$d"
```

Formatters

- Anything else? Yes … formatting for locale
  - A locale is a region of the world
  - It is separate from language
  - Your application might be localized to Spanish, but the user might be in Spain or in Mexico
  - In different regions some things might display differently: numbers, dates, currency, etc.
- iOS will format these things for you if you let it
  - But you must use the appropriate formatter class …
  - `NSNumberFormatter` (including currency)
  - `NSDateFormatter`
  - `NSDateComponentsFormatter`
  - `NSByteCountFormatter`
  - `MKDistanceFormatter`
  - Health Kit (see `HKUnit`) formatters
  - Formatters will even let you set their locale to work in if you deal with multiple locales at once

NSNumberFormatter

- Lots going on here. Check out the documentation.

  But we'll look at two simple cases …

- Displaying numbers

  `let formatter = NSNumberFormatter()`

  … configure it …

  `var numberStyle: NSNumberFormatterStyle // .DecimalStyle, .CurrencyStyle, etc.`

  Can also control number of digits before/after decimal, rounding, currency, etc., then …

  `let stringToDisplay = formatter.stringFromNumber(theNumber)`

- Parsing numbers

  `formatter.numberStyle = .DecimalStyle`

  `let parsedNumber = formatter.numberFromString(userInputtedString) { … }`

  Note that this will return `nil` if a number of the proper format is not found.

NSDateFormatter

- Dates are rather complicated to display properly

  Hopefully one of the built-in styles will work for you

  The style of date can be set separately for time and date

  `var timeStyle: NSDateFormatterStyle`

  `var dataStyle: NSDateFormatterStyle`

  `.ShortStyle` or `.MediumStyle` or `.LongStyle` or `.FullStyle` or `.NoStyle` (exclude)

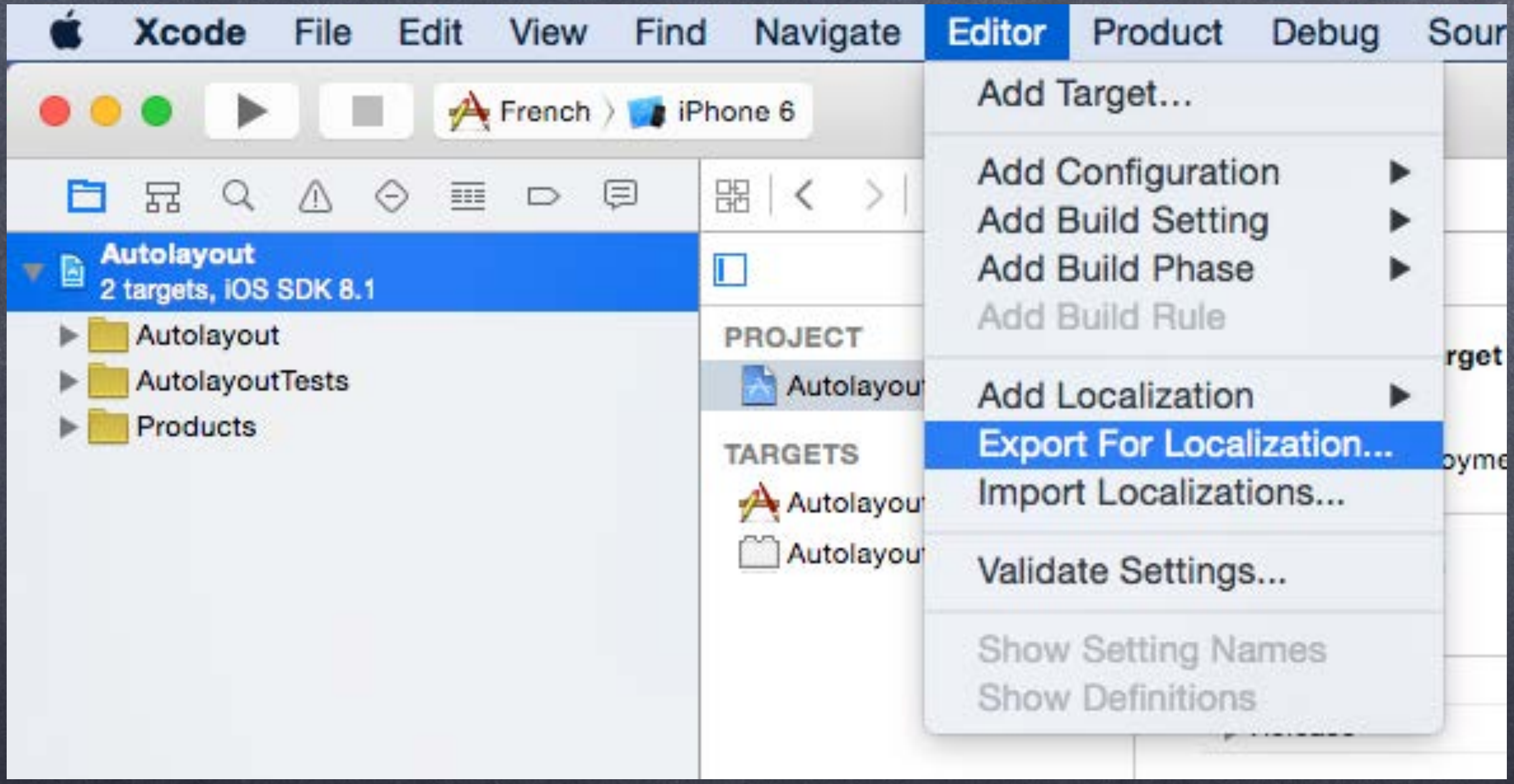
  If you need more control over presentation than this, you'll need to study up!

UIImage

- How to localize images?
  There are a number of ways to approach this.
  The image files themselves (if not in `Images.xcassets`) can be made localizable (File Inspector)
  Or you can keep them in `Images.xcassets` and use `NSLocalizedString` for the image name

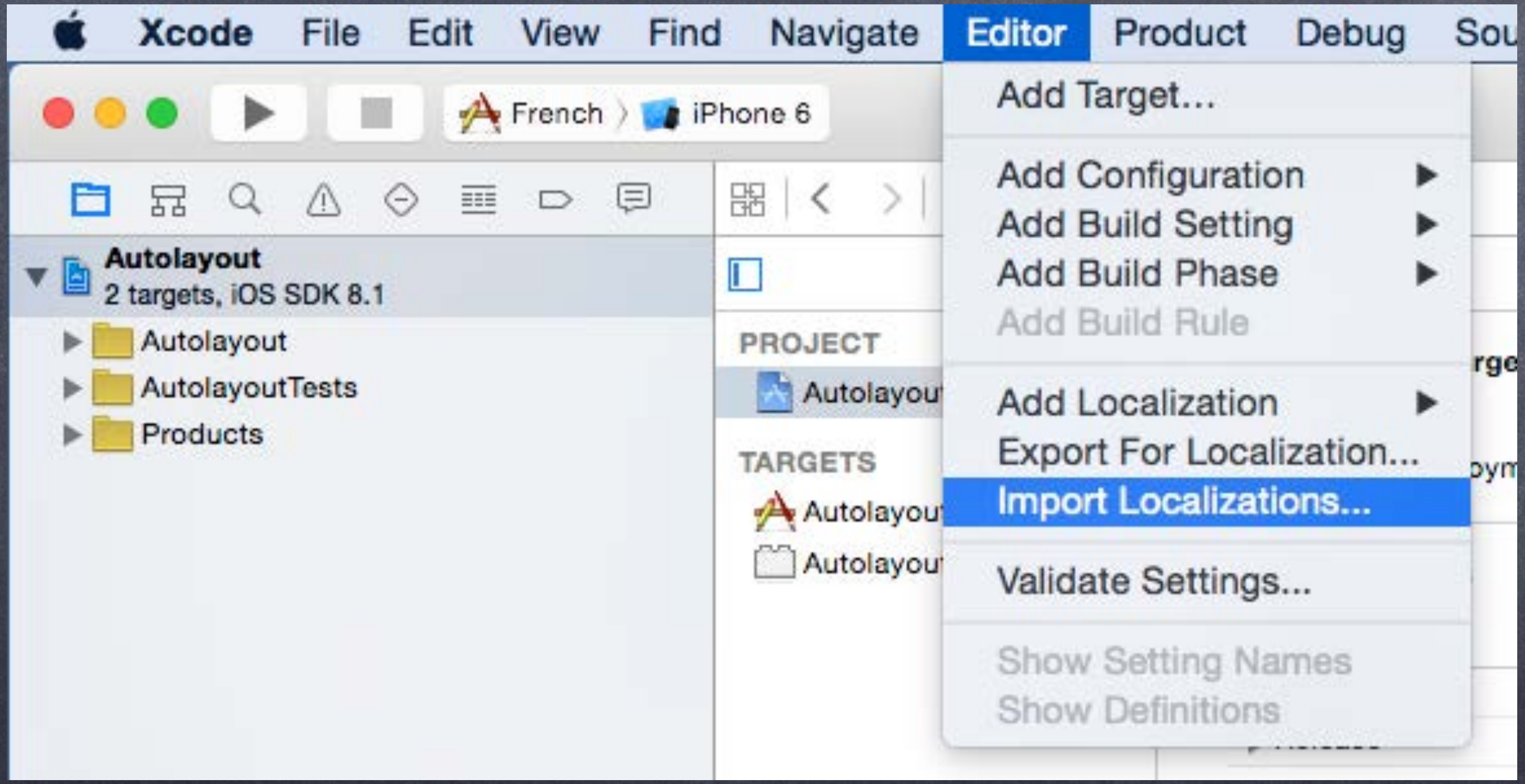
Localizing

- Congratulations! Your application is Internationalized!
  Now what?
  Simply generate an `.xliff` file for each language (an industry-standard "for localizers" file)
  The `.xliff` will contain all of your strings and comments



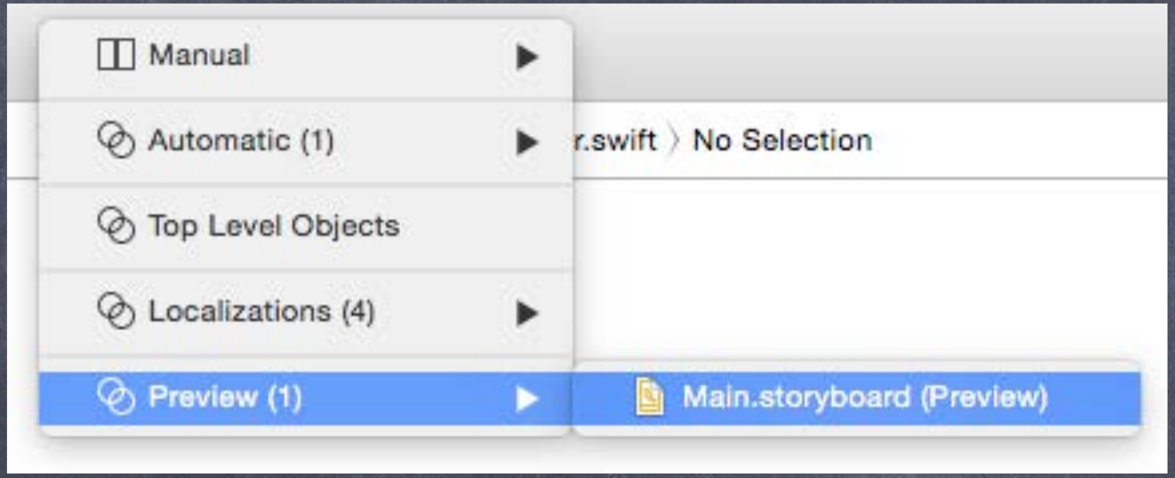
Localizing

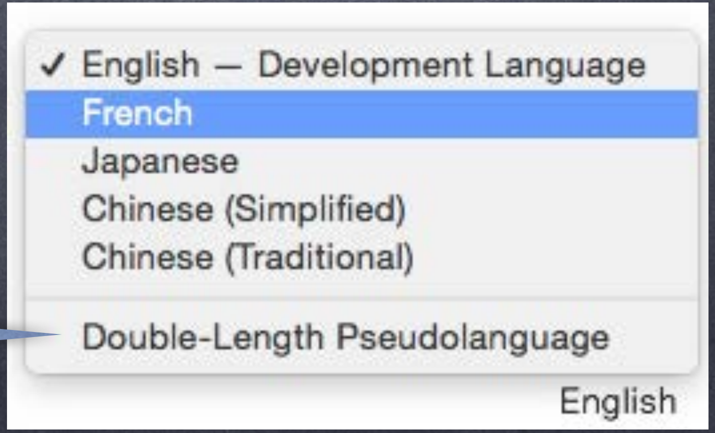
- Congratulations! Your application is Internationalized!
  Now what?
  Simply generate an `.xliff` file for each language (an industry-standard "for localizers" file)
  The `.xliff` will contain all of your strings and comments
  When the `.xliff` comes back from the localizers, just import it
  You can then export/import/export/import repeatedly without losing any translations



Previewing a Localization

- When you get a localization back from your localizer
  - You can preview what the localization will look like
  - You can do this in Xcode (if you just want to see the storyboard localized)

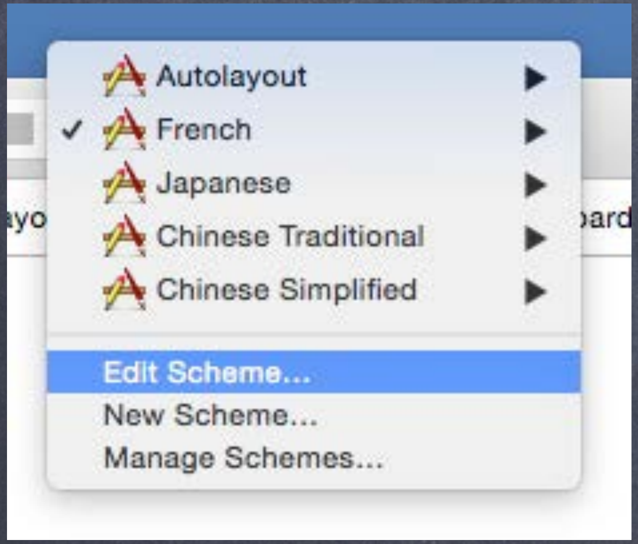


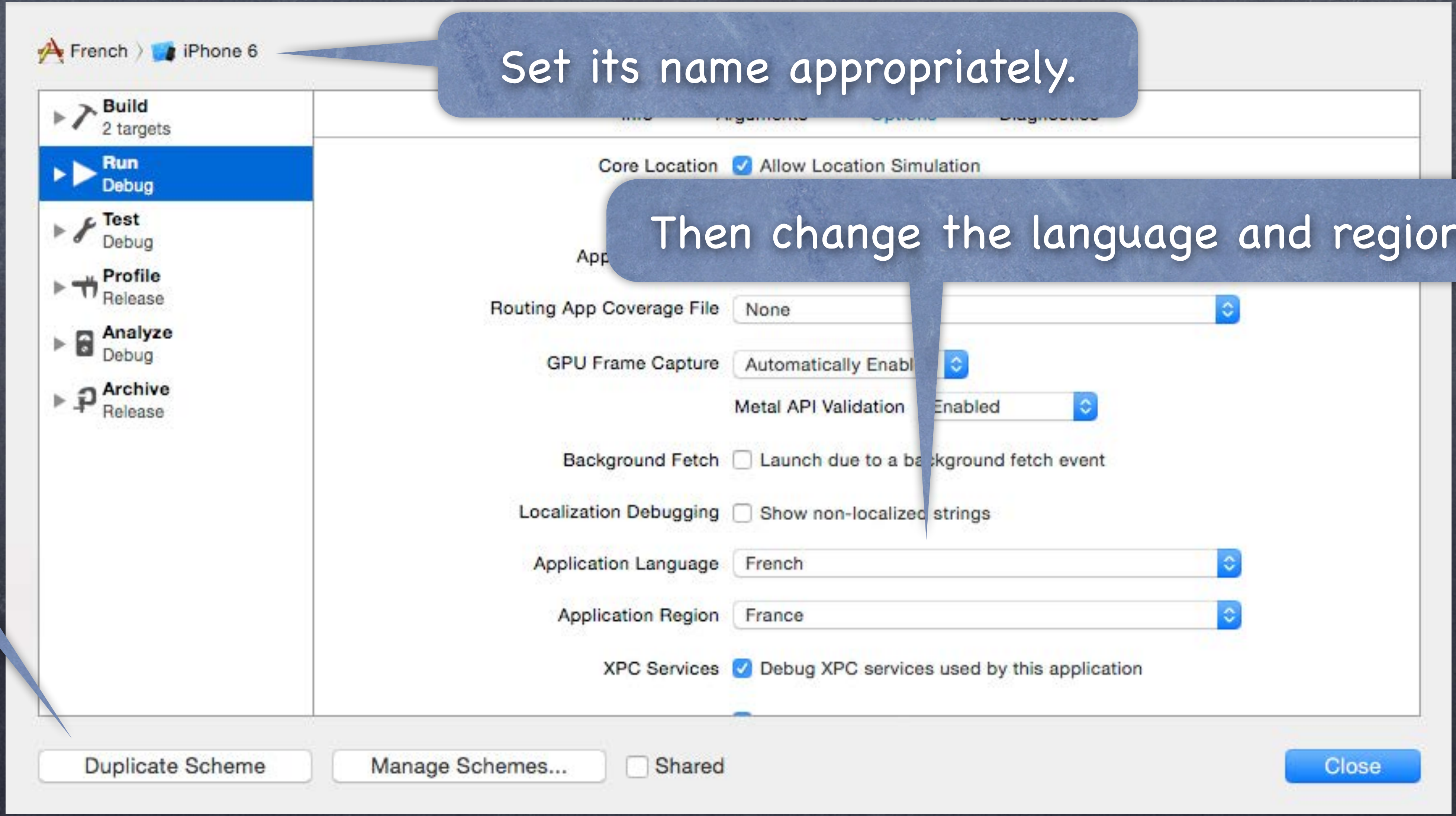


Note this "pseudo-language".
It's a quick way to check your autolayout against "long stringed languages" (e.g. German).

Previewing a Localization

- When you get a localization back from your localizer
  - You can preview what the localization will look like
  - You can do this in Xcode (if you just want to see the storyboard localized)
  - Or you can create a "Scheme" to Run your application in that language





Core Location: ✓ Allow Location Simulation
Routing App Coverage File: None
GPU Frame Capture: Automatically Enabl
Metal API Validation: Enabled
Background Fetch: ☐ Launch due to a background fetch event
Localization Debugging: ☐ Show non-localized strings
Application Language: French
Application Region: France
XPC Services: ✓ Debug XPC services used by this application

Set its name appropriately.

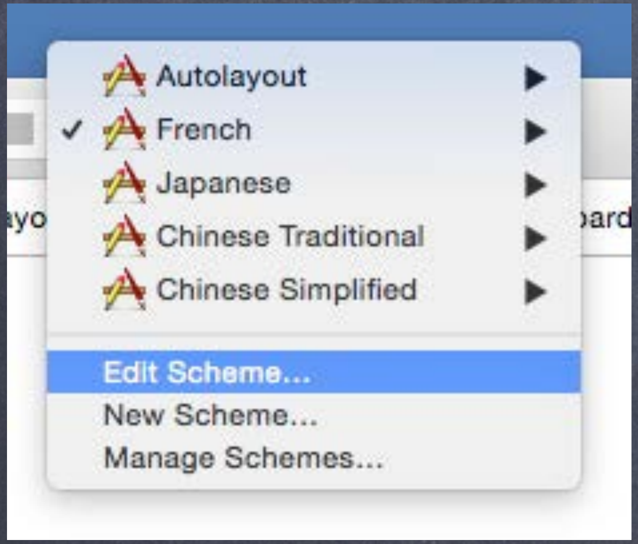
Then change the language and region here.

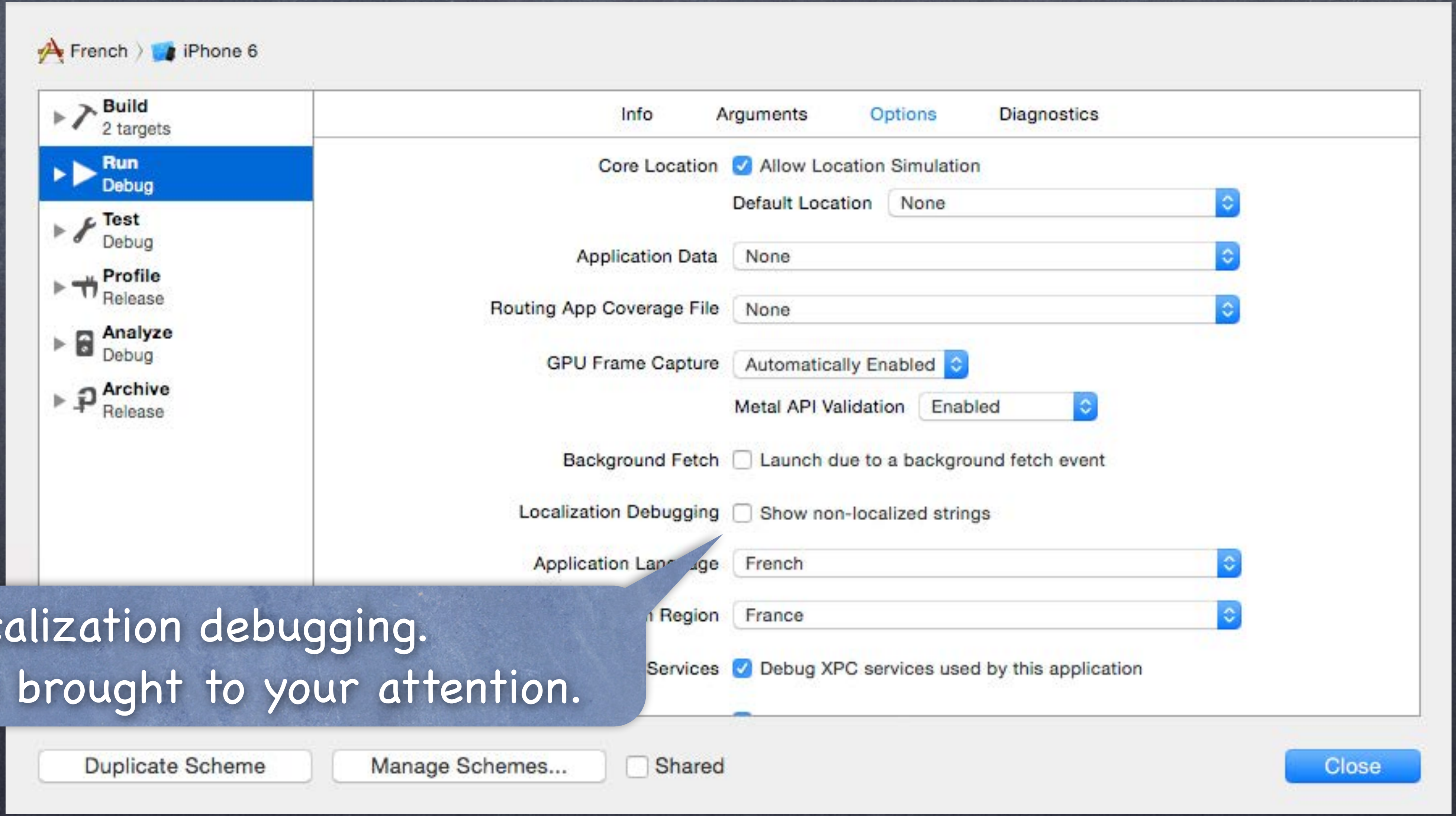
Duplicate an existing scheme.

CS193p
Winter 2015

Previewing a Localization

- When you get a localization back from your localizer
  - You can preview what the localization will look like
  - You can do this in Xcode (if you just want to see the storyboard localized)
  - Or you can create a "Scheme" to Run your application in that language





Core Location: ✓ Allow Location Simulation
Default Location: None
Application Data: None
Routing App Coverage File: None
GPU Frame Capture: Automatically Enabled
Metal API Validation: Enabled
Background Fetch: ☐ Launch due to a background fetch event
Localization Debugging: ☐ Show non-localized strings
Application Language: French
Region: France
Services: ✓ Debug XPC services used by this application

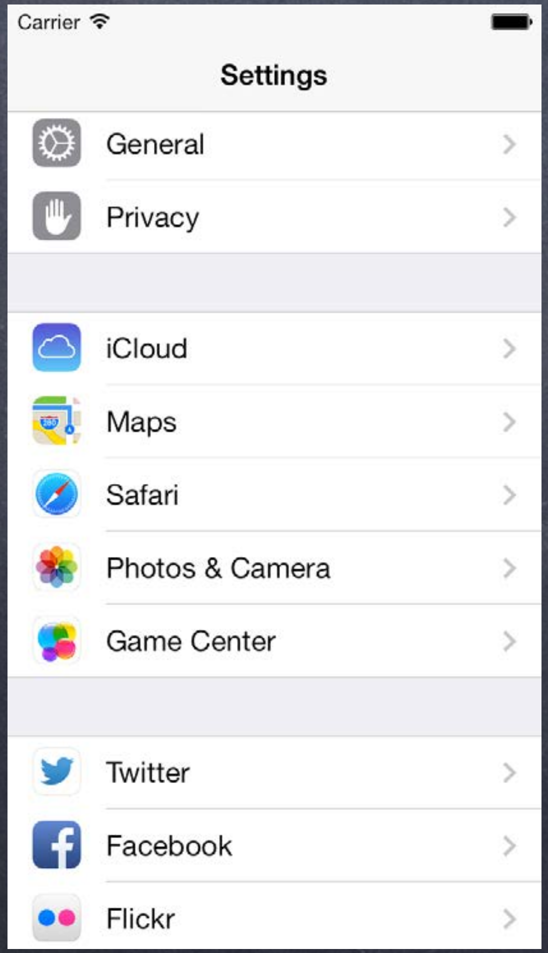
You can also turn on localization debugging.
Unlocalized strings will now be brought to your attention.

Localization

- That's all it takes
  - Localization in iOS 8 has gotten substantially more streamlined, so there's no excuse!
  - Get that autolayout right!
  - Get those `NSLocalizedString` calls in there!
  - Use those formatters!

Settings

- A little bit of UI for your application in the Settings application
  - You should use this sparingly (if at all).
  - It's appropriate only for very rarely used settings or default behavior.
  - You don't want to make your users ever have to go here for normal use of your application.
  - The settings appear in your application via NSUserDefaults.
  - You specify the UI and the associated defaults in a property list file.

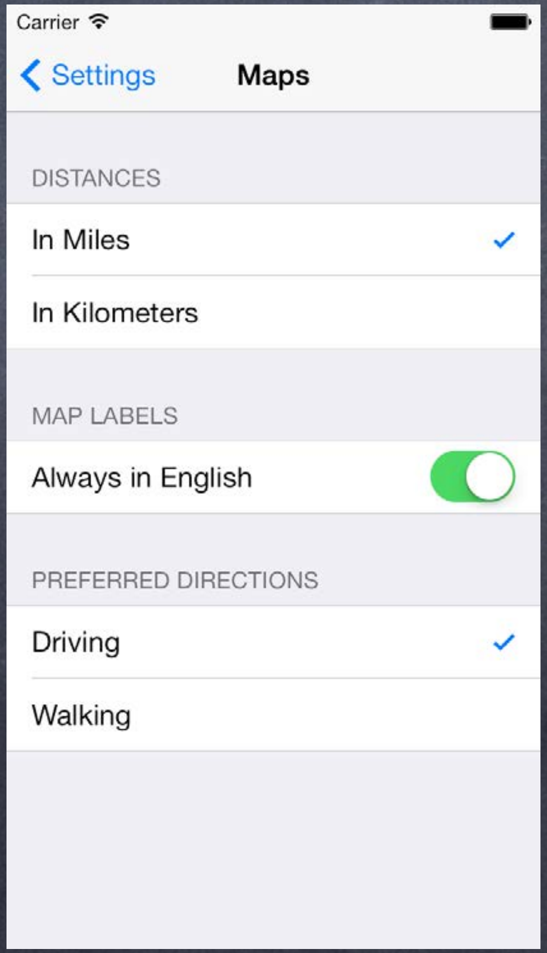

Carrier

Settings

General >
Privacy >

iCloud >
Maps >
Safari >
Photos & Camera >
Game Center >

Twitter >
Facebook >
Flickr >

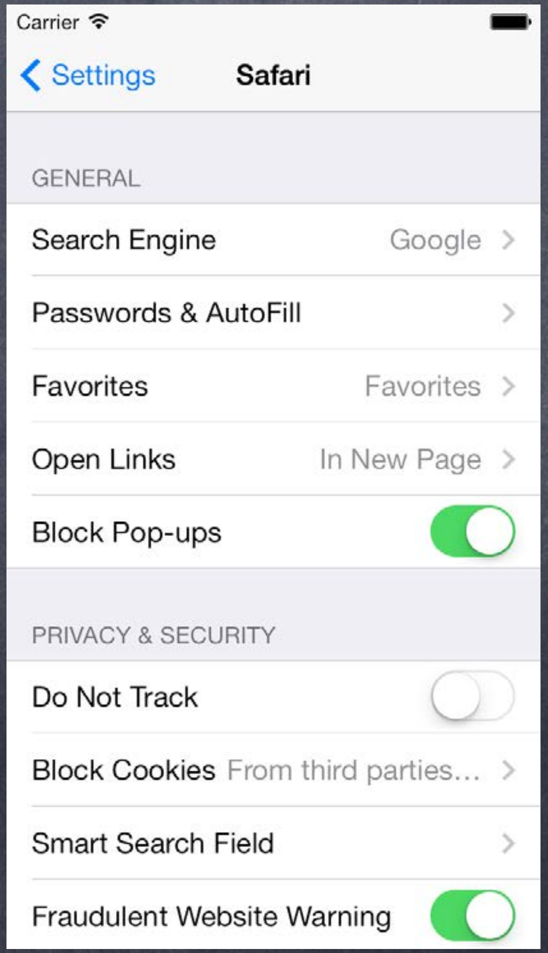

Carrier

< Settings **Maps**

DISTANCES
In Miles ✓
In Kilometers

MAP LABELS
Always in English ✓

PREFERRED DIRECTIONS
Driving ✓
Walking


Carrier

< Settings **Safari**

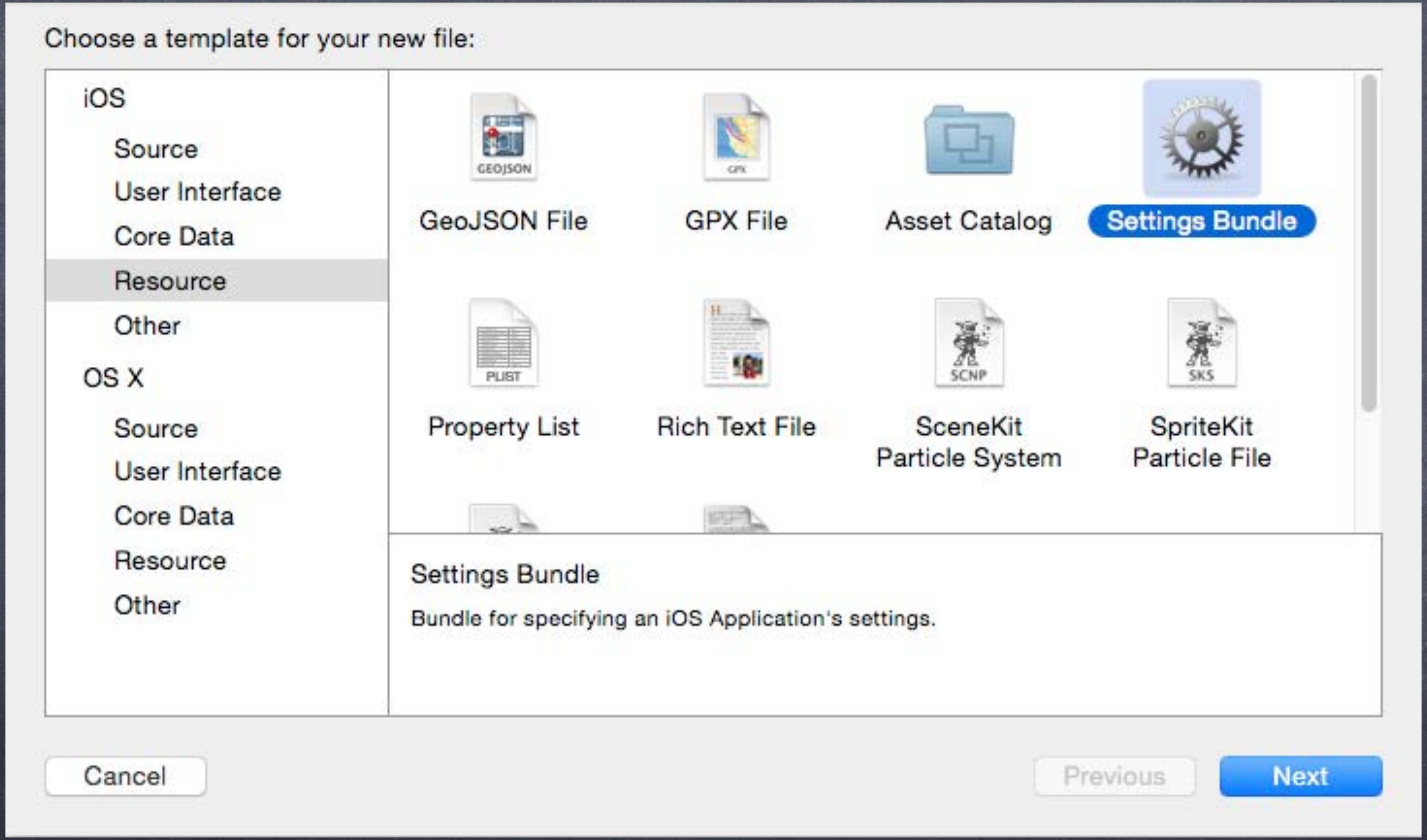
GENERAL
Search Engine Google >
Passwords & AutoFill >
Favorites Favorites >
Open Links In New Page >
Block Pop-ups ✓

PRIVACY & SECURITY
Do Not Track ○
Block Cookies From third parties… >
Smart Search Field >
Fraudulent Website Warning ✓

Settings

- Creating a Settings Bundle

  Choose New File from the menus and then ...

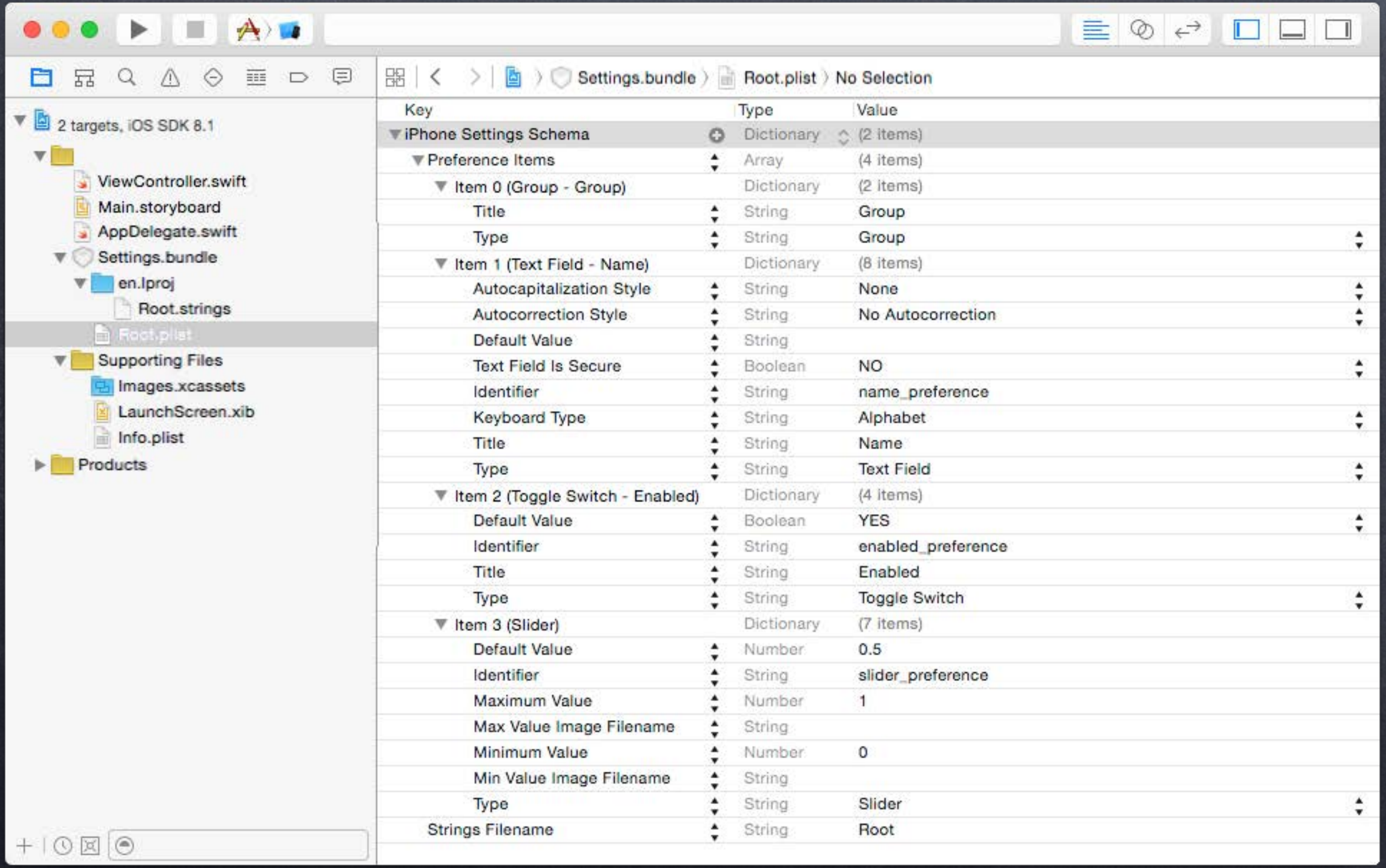


Settings Bundle

Bundle for specifying an iOS Application's settings.

Settings

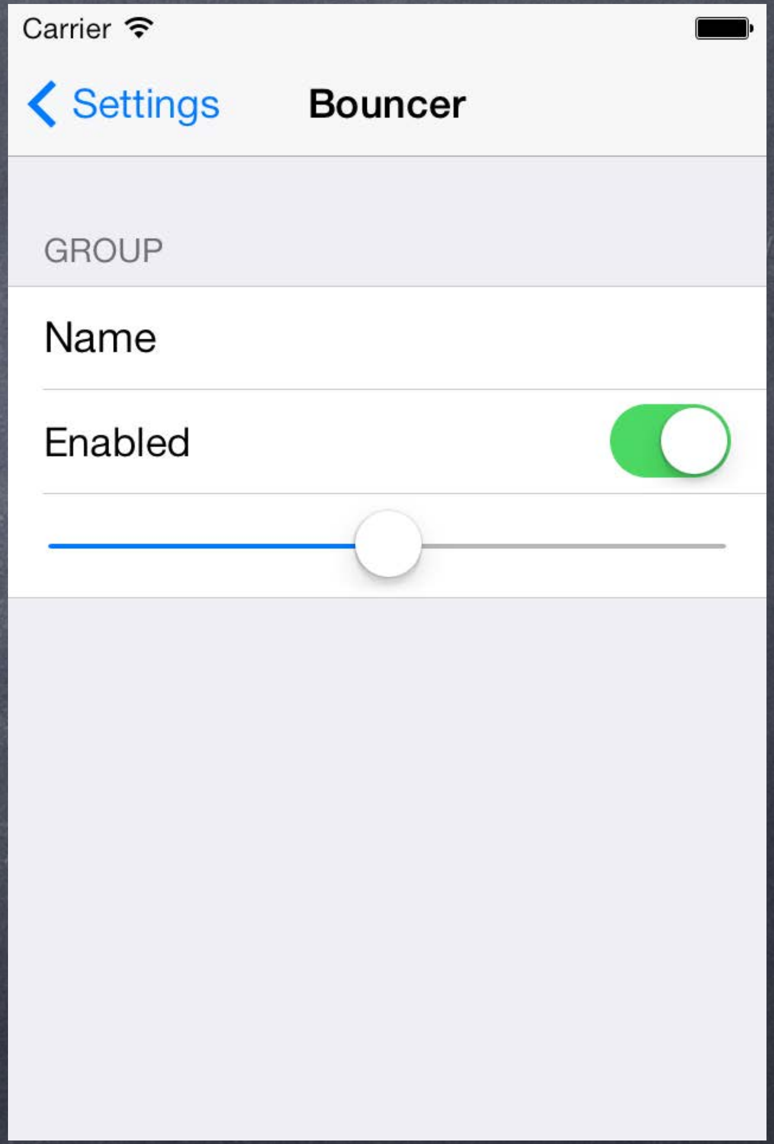
- A "sample" settings bundle is created
  - Check the documentation for all the possible things you can create
  - The sample is a good place to start when trying to understand how things work



2 targets, iOS SDK 8.1
ViewController.swift
Main.storyboard
AppDelegate.swift
Settings.bundle
en.lproj
Root.strings
Root.plist
Supporting Files
Images.xcassets
LaunchScreen.xib
Info.plist
Products

Settings.bundle 〉 Root.plist 〉 No Selection

Key	Type	Value
▼iPhone Settings Schema	Dictionary	(2 items)
▼Preference Items	Array	(4 items)
▼Item 0 (Group - Group)	Dictionary	(2 items)
Title	String	Group
Type	String	Group
▼Item 1 (Text Field - Name)	Dictionary	(8 items)
Autocapitalization Style	String	None
Autocorrection Style	String	No Autocorrection
Default Value	String	
Text Field Is Secure	Boolean	NO
Identifier	String	name_preference
Keyboard Type	String	Alphabet
Title	String	Name
Type	String	Text Field
▼Item 2 (Toggle Switch - Enabled)	Dictionary	(4 items)
Default Value	Boolean	YES
Identifier	String	enabled_preference
Title	String	Enabled
Type	String	Toggle Switch
▼Item 3 (Slider)	Dictionary	(7 items)
Default Value	Number	0.5
Identifier	String	slider_preference
Maximum Value	Number	1
Max Value Image Filename	String	
Minimum Value	Number	0
Min Value Image Filename	String	
Type	String	Slider
Strings Filename	String	Root



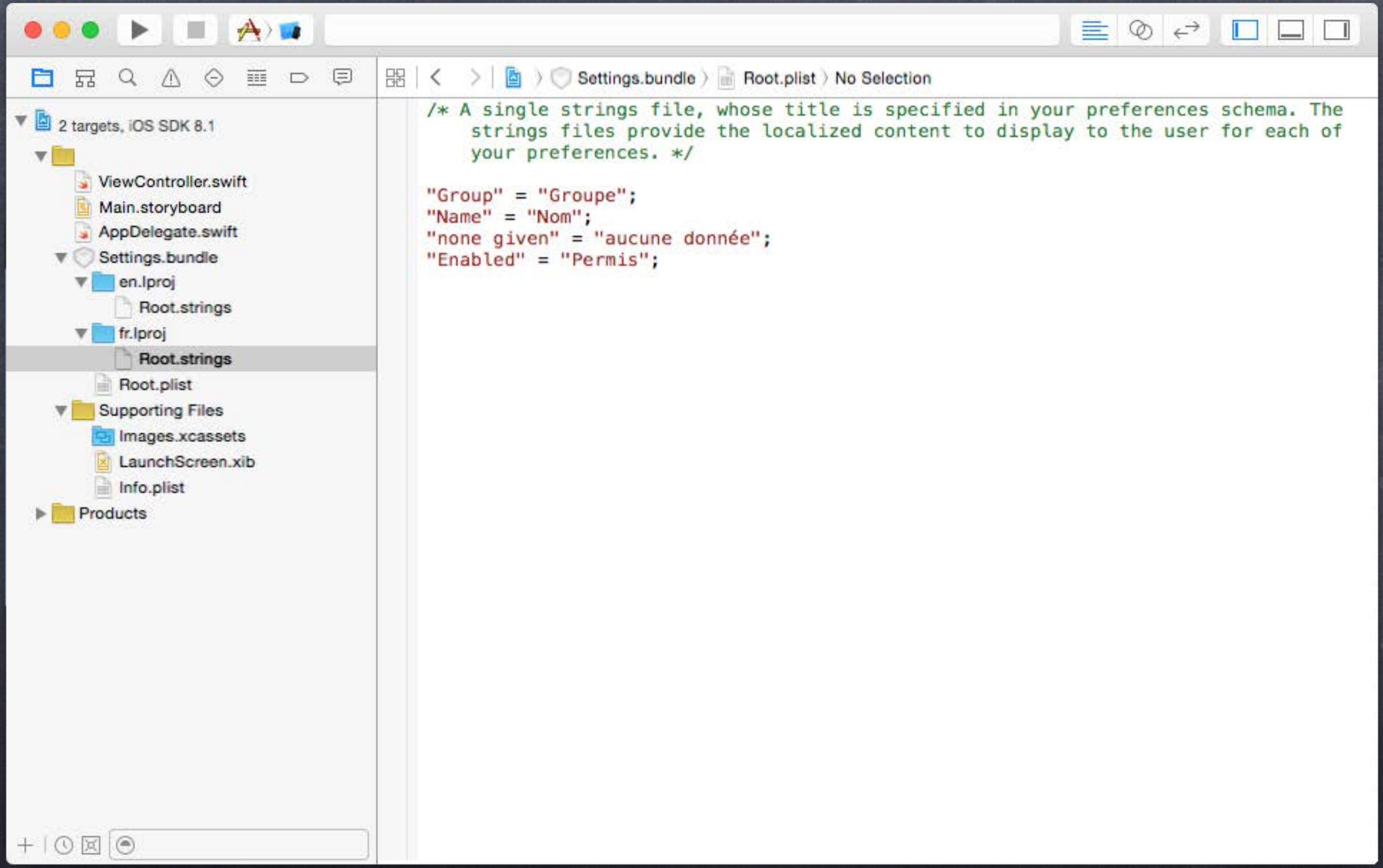
Settings

Localization of Settings

Localization does not use the normal mechanism in Settings

Instead, you create strings files inside the `.lproj` directories in your Settings bundle

Evidently you'll have to send them to your translators separately from your `.xliffs`



```
/* A single strings file, whose title is specified in your preferences schema. The
    strings files provide the localized content to display to the user for each of
    your preferences. */

"Group" = "Groupe";
"Name" = "Nom";
"none given" = "aucune donnée";
"Enabled" = "Permis";
```