Customizing App Behavior

iPhone and iPod touch Development
Fall 2009 — Lecture 26

Questions?

Announcements

- Details about final project deliverables will be posted tonight

Today's Topics

- App Icons
- App Icon Badges
- Launch Images
- Inter-App Communication
- Handling Interruptions
- Internationalization

Notes

- I'm showing the relevant portions of the view controller interfaces and implementations in these notes
- Remember to release relevant memory in the -dealloc methods — they are not shown here
- You will also need to wire up outlets and actions in IB
- Where delegates are used, they too require wiring in IB

App Icons

App Icons

- There are several different icons that you should prepare when creating an icon for the application...
  - Distribution artwork — 512 x 512 pixels
  - Home screen app icon — 57 x 57 pixels
  - Small app icon — 29 x 29 pixels
- The small app icon is used in 2 spots in the iPhone OS...
  - Spotlight search results
  - Settings bundle (if your app has a settings bundle)

Default Names

- The iPhone OS uses the following names for app icons...
  - Home screen icon — Icon.png
  - Settings bundle icon — Icon-Small.png
- You can change the home screen icon in the app's plist...

Key	Value
▼ Information Property List	(13 items)
Icon already includes gloss and bevel effects	☑
Localization native development region	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	blah.png
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc10
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone environment	☑
Main nib file base name	MainWindow

Home Screen Icon Styling

- By default, the iPhone OS automatically adds some visual effects so that it coordinates with the built-in icons
- Specifically, the iPhone OS adds the following...
  - Rounded corners
  - Drop shadow
  - Reflective shine

Disabling (Some) Home Screen Icon Styling

- If you don't want the shine and beveled edges, you can turn that off in your app's plist...

Key	Value
▼ Information Property List	(13 items)
Icon already includes gloss and bevel effects	☑
Localization native development region	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc10
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone environment	☑
Main nib file base name	MainWindow

The Resulting App

App Badges

Application Badges

- As you've probably seen, app icons on the home screen can be decorated with numbered badges...

Application Badges

- From within your application you can easily set this badge with the following UIApplication property...

```
@property(nonatomic) NSInteger applicationIconBadgeNumber;
```

- For example...

```
[[UIApplication sharedApplication] setApplicationIconBadgeNumber:3];
```

The Resulting App

Application Badges

- The badging system can display a 4-digit number without any issue, but if you set your number too large (5 or more digits) it will be truncated with ellipses
- To remove the badge, simply set the badge number to zero and it will go away
- Negative badge numbers aren’t supported

Truncating and Clearing

Launch Images

Launch Images

- Per Apple's Human Interface Guidelines (HIG),...
  - To enhance the user's experience at application launch, you should provide a launch image
  - A launch image looks very similar to the first screen your application displays
- iPhone OS displays this image instantly when the user taps your application icon on the Home screen.
- As soon as it's ready for use, your application displays its first screen, replacing the launch placeholder image

Launch Image Examples

Launch Image Examples

Launch Image Examples

Launch Images

- Per Apple's Human Interface Guidelines (HIG),...
  - It's important to emphasize that the reason to supply a launch image is to improve user experience; it is not an opportunity to provide:
    - An "application entry experience," such as a splash screen
    - An About window
    - Branding elements, unless they are a static part of your application's first screen

Launch Images

- That said, do some apps use them as splash screens? Yeah.

Adding a Launch Image

- The launch image should be a 320 x 480 pixel PNG image called Default.png
- Set the status bar color in the image to what you're using in your app (if your app is not full screen)
- Apple suggests making the first screen identical to the first screen of the application except for...
  - Text — text can vary per locale, so omit it
  - Table cell contents — show the table outline, not contents
  - User interface elements that might change

Inter-App Communication

Inter-App Communications

- Even though iPhone apps are sandboxed, you can still pass data between applications
- This data passing is accomplished via custom URL schemes

Custom URL Schemes

- You can register URL types for your application that include custom URL schemes
- A custom URL scheme is a mechanism through which third-party applications can interact with each other and with the system apps
- Through a custom URL scheme, an application can make its services available to other applications

Applications for Custom URL Schemes

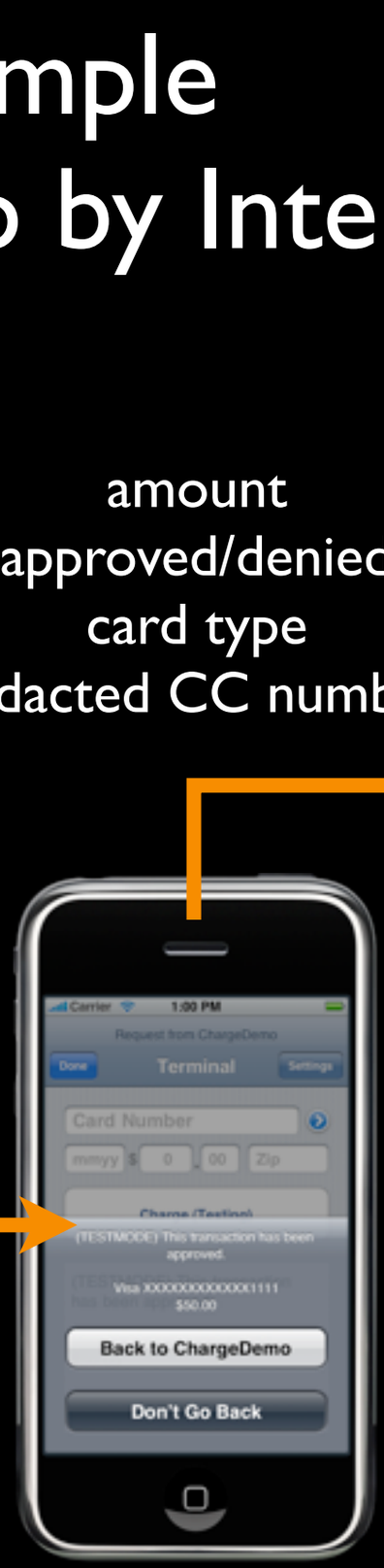
- Having another app perform some action, such as...
  - Handing data off to another app for processing
  - Delegate to another app and come back to the active app
- Getting data into your app...
  - Import data from a URL on a webpage in Mobile Safari
  - Import data from a URL from the Email app
- Migrating data between apps...
  - Move data from a “lite” app to the “full” app so the user doesn’t have to re-enter data or re-play levels

Real World Example
Credit Card Terminal App by Interfence

Sales App

amount
invoice number
description

process credit card



amount
approved/denied
card type
redacted CC number

Sales App

Credit Card Terminal App

Registering a Custom URL

- To register a custom URL for your application, you need to add some entries to your app's plist
- You need to add the "URL types" key — additionally within that key you need to specify the following...
  - a URL identifier — typically in reverse domain name notation (i.e. edu.umbc.xxxx)
  - a URL Schemes entry — with at least one specified scheme

Implementing a Sample Credit Card Processor

- We'll look at what's required to implement the data passing that's required for something like the credit card processor
- We'll have 2 apps — a Sales and a Processor app
- The Sales app will call the Processor app with an amount and tell the Processor which app to return to (the Sales app)
- The Processor app will accept amount from the Sales app and allows the user to authorize or decline the charge where the result gets passed back to the calling app (Sales in this case)
- Upon returning to the Sales app, we'll display the results

Handing Opening with a URL

- The underlying UIApplicationDelegate method that gets called when an app is opened via a URL is...

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url;
```

- This method should return NO, if for some reason it is unable to handle the specified URL

URL Encoding

- URL encoding is the process of encoding key/value pairs by encoding them as part of the URL, such as...
  - http://foo.com/?action=register&name=Dan%20Hood
- To make the processing of encoding and decoding these URLs a bit easier, I'm going to use 2 categories from the MuiKit library...
  - http://github.com/millenomi/muikit/blob/master/NSURL+L0URLParsing.h
  - http://github.com/millenomi/muikit/blob/master/NSURL+L0URLParsing.m

NSURL+L0URLParsing.[hm]

- The 2 categories provided by library allow me to specify my data in a dictionary and serialize it as an encoded string and vice-versa
- Specifically, the added methods are...

```
@interface NSURL (L0URLParsing)
- (NSDictionary*) dictionaryByDecodingQueryString;
@end
@interface NSDictionary (L0URLParsing)
- (NSString*) queryString;
@end
```

Our Apps Custom URL Schemes

Key	Value
▼ Information Property List	(13 items)
▼ URL types	(1 item)
▼ Item 0	(2 items)
URL identifier	edu.umbc.sales
▼ URL Schemes	(1 item)
Item 0	sales
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_N
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow

Key	Value
▼ Information Property List	(13 items)
▼ URL types	(1 item)
▼ Item 0	(2 items)
URL identifier	edu.umbc.processor
▼ URL Schemes	(1 item)
Item 0	processor
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc1034identifier
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow

SalesViewController.xib

- In the Sales NIB we'll provide a text entry field where the user can enter an amount to charge
- In a real app, this might be computed by your app as a result of some actions
- We'll also provide a "Ring Up Sale" button that'll kick the amount over to the Processor app for processing

SalesAppDelegate.m

```
#import "SalesAppDelegate.h"
#import "SalesViewController.h"

@implementation SalesAppDelegate

@synthesize window;
@synthesize viewController;

- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
  return [viewController handleURL:url];
}

- (void)applicationDidFinishLaunching:(UIApplication *)application {
  [window addSubview:viewController.view];
  [window makeKeyAndVisible];
}

- (void)dealloc {
  [viewController release];
  [window release];
  [super dealloc];
}

@end
```

SalesViewController.h

```
#import <UIKit/UIKit.h>

@interface SalesViewController : UIViewController {

  UITextField *amount;

}

@property(nonatomic, retain) IBOutlet UITextField *amount;

- (BOOL)handleURL:(NSURL *)url;
- (IBAction)openApp;

@end
```

SalesViewController.m

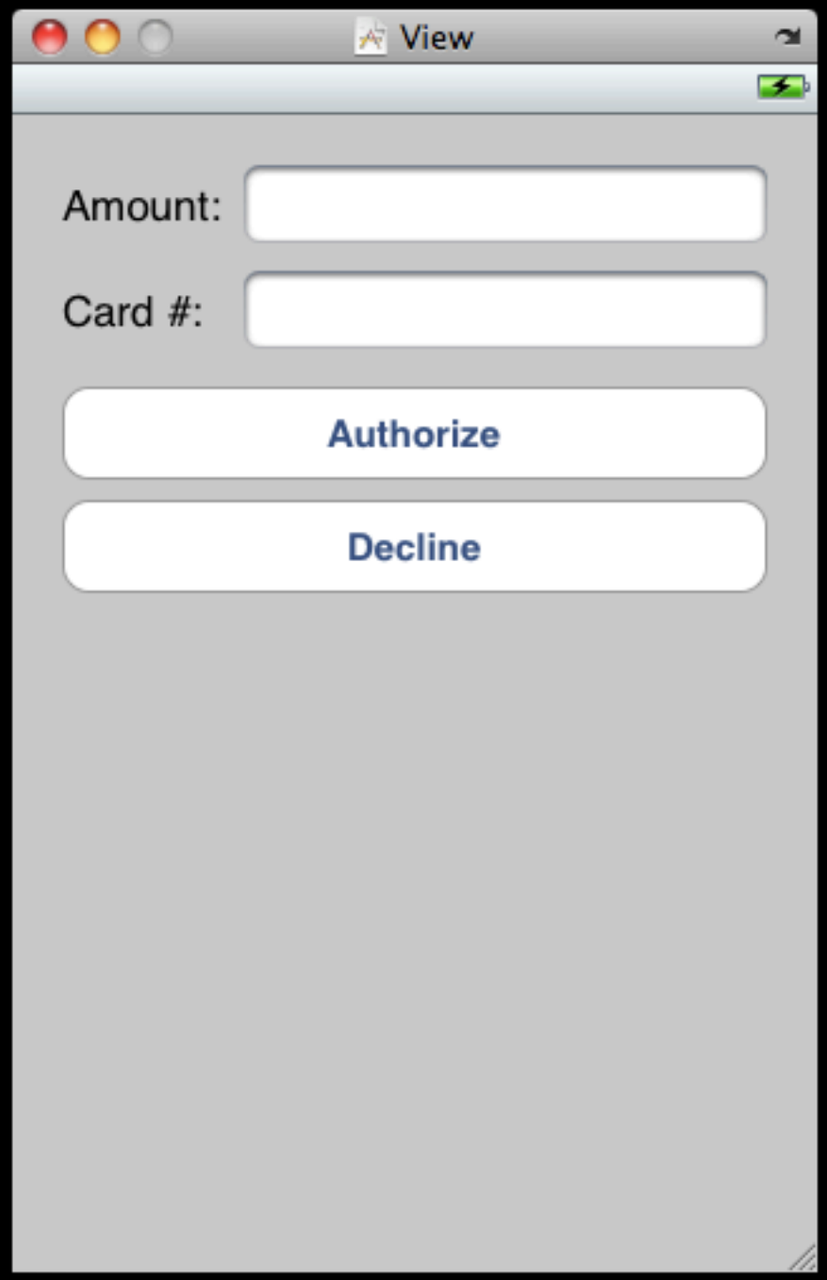
```objc
#import "SalesViewController.h"
#import "NSURL+L0URLParsing.h"

@implementation SalesViewController

@synthesize amount;

- (BOOL)handleURL:(NSURL *)url {
  NSDictionary *data = [url dictionaryByDecodingQueryString];
  NSString *code = [data objectForKey:@"code"];
  NSString *creditCard = [data objectForKey:@"creditCard"];
  NSString *msg = [NSString stringWithFormat:@"Card number %@", creditCard];
  if ([code isEqualToString:@"AUTHORIZED"]) {
    UIAlertView *alert = [[[UIAlertView alloc] initWithTitle:@"Authroized"
                          message:msg delegate:nil cancelButtonTitle:@"OK"
                          otherButtonTitles:nil] autorelease];
    [alert show];
  } else if ([code isEqualToString:@"DECLINED"]) {
    UIAlertView *alert = [[[UIAlertView alloc] initWithTitle:@"Declined"
                          message:msg delegate:nil cancelButtonTitle:@"OK"
                          otherButtonTitles:nil] autorelease];
    [alert show];
  }
  return YES;
}
// ...
```

SalesViewController.m

```
// ...

- (IBAction)openApp {
  NSDictionary *data = [NSDictionary dictionaryWithObjectsAndKeys:
                        @"sales://", @"returnURL",
                        self.amount.text, @"amount",
                        nil];
  NSString *str = [NSString stringWithFormat:@"processor://?%@",
                                             [data queryString]];
  NSURL *url = [NSURL URLWithString:str];
  [[UIApplication sharedApplication] openURL:url];
}

- (void)dealloc {
  self.amount = nil;
  [super dealloc];
}

@end
```

ProcessorViewController.xib

- For the Processor NIB, we'll display the amount the came in via a URL parameter
- We'll allow the user to enter a credit card number which could be used in processing
- In lieu of doing actual processing we'll simply provide an authorize and decline button to simulate credit card processing and returning back to the calling app



ProcessorAppDelegate.m

```
#import "ProcessorAppDelegate.h"
#import "ProcessorViewController.h"

@implementation ProcessorAppDelegate

@synthesize window;
@synthesize viewController;

- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
  return [viewController handleURL:url];
}

- (void)applicationDidFinishLaunching:(UIApplication *)application {
  [window addSubview:viewController.view];
  [window makeKeyAndVisible];
}

- (void)dealloc {
  [viewController release];
  [window release];
  [super dealloc];
}

@end
```

ProcessorViewController.h

```
#import <UIKit/UIKit.h>

@interface ProcessorViewController : UIViewController {

  NSString *returnURL;
  UITextField *amount;
  UITextField *creditCard;

}
```

```
@property(nonatomic, copy) NSString *returnURL;
@property(nonatomic, retain) IBOutlet UITextField *amount;
@property(nonatomic, retain) IBOutlet UITextField *creditCard;

- (IBAction)authorize;
- (IBAction)decline;
- (BOOL)handleURL:(NSURL *)url;

@end
```

ProcessorViewController.m

```
#import "ProcessorViewController.h"
#import "NSURL+L0URLParsing.h"

@implementation ProcessorViewController

@synthesize returnURL;
@synthesize amount;
@synthesize creditCard;

- (void)callReturnURLWithCode:(NSString *)code {
  int length = [self.creditCard.text length];
  NSString *lastFour = [self.creditCard.text substringFromIndex:length - 4];
  NSString *mask = [@"" stringByPaddingToLength:length - 4
                                     withString:@"x" startingAtIndex:0];
  NSString *maskedCard = [NSString stringWithFormat:@"%@%@", mask, lastFour];
  NSDictionary *data = [NSDictionary dictionaryWithObjectsAndKeys:
                        code, @"code",
                        maskedCard, @"creditCard",
                        nil];
  NSString *str = [NSString stringWithFormat:@"%@://?%@",
                   self.returnURL, [data queryString]];
  NSURL *url = [NSURL URLWithString:str];
  [[UIApplication sharedApplication] openURL:url];
}

// ...
```

ProcessorViewController.m

```
// ...

- (IBAction)authorize {
  [self callReturnURLWithCode:@"AUTHORIZED"];
}

- (IBAction)decline {
  [self callReturnURLWithCode:@"DECLINED"];
}
```

```
- (BOOL)handleURL:(NSURL *)url {
  NSDictionary *data = [url dictionaryByDecodingQueryString];
  self.returnURL = [data objectForKey:@"returnURL"];
  self.amount.text = [data objectForKey:@"amount"];
  return YES;
}

- (void)dealloc {
  self.returnURL = nil;
  self.amount = nil;
  self.creditCard = nil;
  [super dealloc];
}

@end
```

Our Example Apps

Sales App

amount
returnURL

Processor App

approved/denied
code

Processor App

Sales App

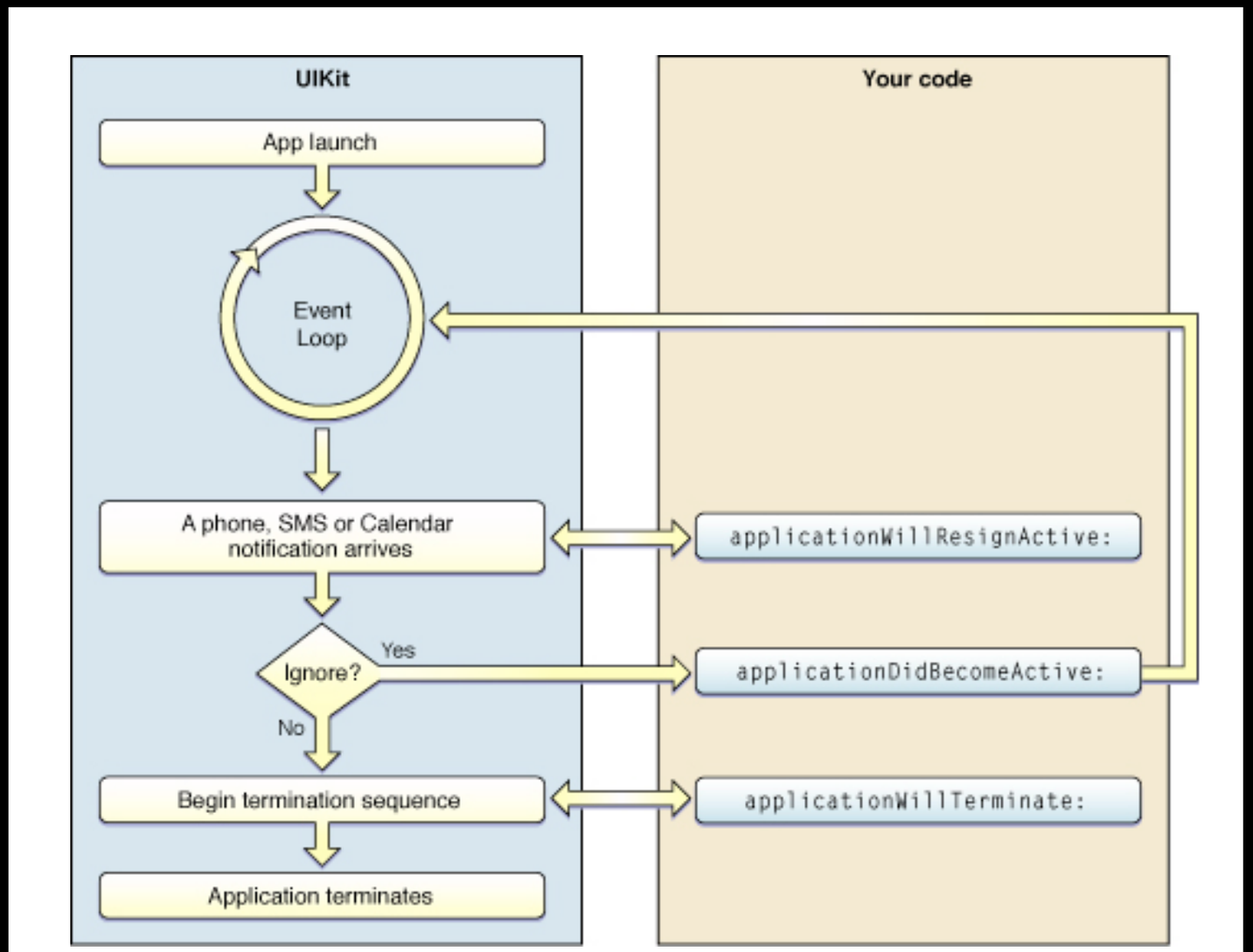
Custom URL Scheme Considerations

- Security
  - Since this is a way to get data into your app, you want to make sure that you treat data with the same level of suspicion as any other user provided data
- There's no central URL scheme registry
  - No way to discover what's available
  - If you squat on a URL scheme for a built in app, you're out of luck — Apple's takes precedence
  - If two third-party apps squat on the same URL scheme the behavior is undefined

Handling Interruptions

App Lifecycle

- Remember earlier when we talked about application lifecycle events...



Interruptions

- In addition to startup and shutdown lifecycle events, your app delegate is also fed events when an overlay is displayed atop your app by the iPhone OS
- This happens for a number of reasons...
  - Incoming phone call
  - Incoming text message
  - Calendar event notice
  - Bringing up the iPod app controls

Handling Interruptions

- During the time that the overlay is displayed the user cannot interact with your application
- This can be a problem, especially in apps that are doing things without user input
- For example, if you're playing a game with timers, it may make sense to pause the game which the dialog is overlayed

UIApplicationDelegate Methods

- When an overlay dialog is displayed, the following UIApplicationDelegate method is called...

```
- (void)applicationWillResignActive:(UIApplication *)application;
```

- Whenever the app becomes the foreground application (including after app launch) the following method is called...

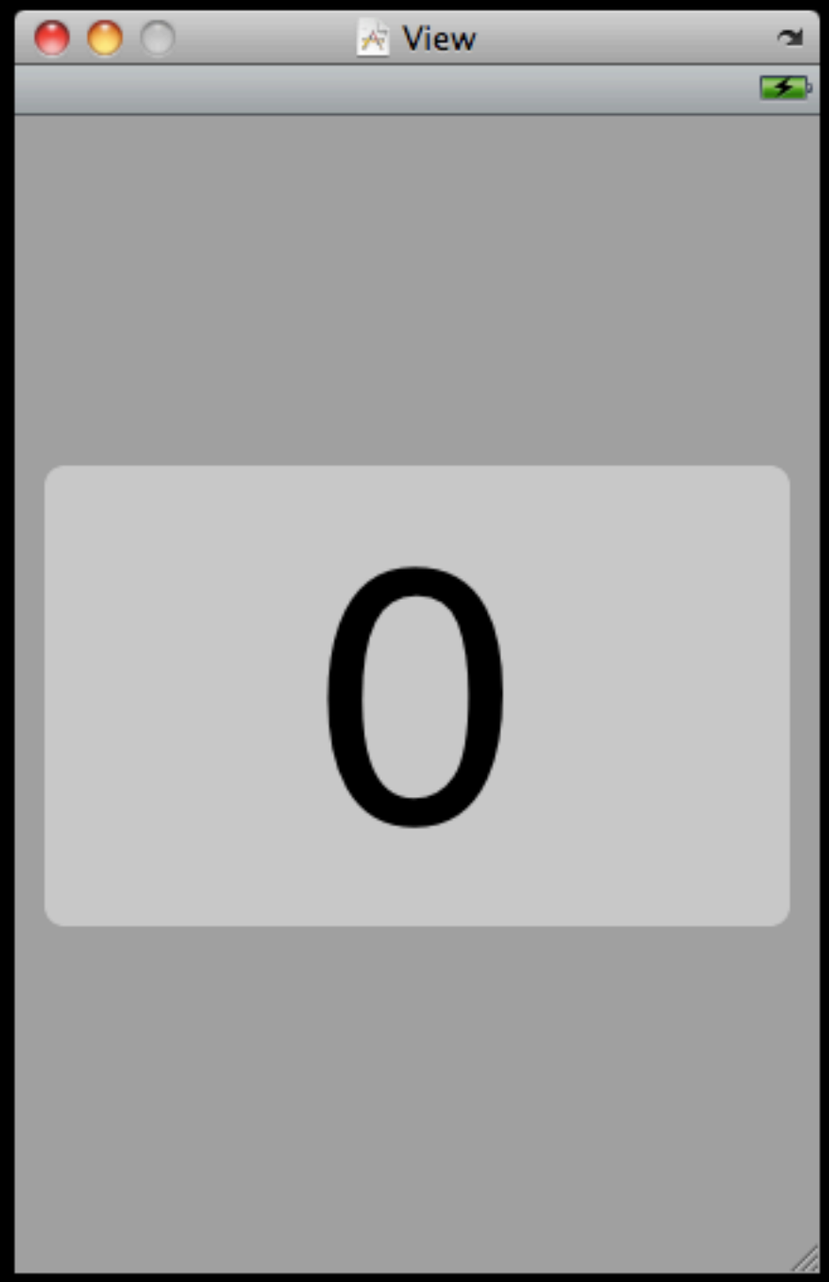
```
- (void)applicationDidBecomeActive:(UIApplication *)application;
```

Example

- Since many games have timers of one type or another, we're going to simulate manipulating a simple timer using these lifecycle callbacks
- The overall strategy is as follows...
  - Set a timer to advance the label each second
  - If the app is told that it is resigning being active, we'll pause this timer
  - When the user returns back to the app, we'll pick back up the timer and label updates

TimerViewController.xib

- For this application, we'll have a simple UILabel that's being updated via a timer every second
- In a real game, you might be animating the scene or running count down/up timers of your own



TimerAppDelegate.m

```
#import "TimerAppDelegate.h"
#import "TimerViewController.h"

@implementation TimerAppDelegate

@synthesize window;
@synthesize viewController;

- (void)applicationWillResignActive:(UIApplication *)application {
    [viewController pause];
}
- (void)applicationDidBecomeActive:(UIApplication *)application {
    [viewController resume];
}
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}

@end
```

TimerViewController.h

```
#import <UIKit/UIKit.h>

@interface TimerViewController : UIViewController {
  UILabel *label;
  NSTimer *timer;
}

@property(nonatomic, retain) IBOutlet UILabel *label;
@property(nonatomic, retain) NSTimer *timer;

- (void)pause;
- (void)resume;

@end
```

TimerViewController.m

```
#import "TimerViewController.h"

@implementation TimerViewController

@synthesize label;
@synthesize timer;

- (void)pause {
  [self.timer invalidate];
  self.label.text = @"Paused";
}

- (void)resume {
  self.timer = [NSTimer scheduledTimerWithTimeInterval:1
                                                target:self
                                              selector:@selector(tictoc)
                                              userInfo:nil
                                               repeats:YES];
}

// ...
```

TimerViewController.m

```
// ...

- (void)tictoc {
  static int time = 0;
  self.label.text = [NSString stringWithFormat:@"%d", ++time];
}

- (void)dealloc {
  self.label = nil;
  self.timer = nil;
  [super dealloc];
}

@end
```

The Resulting App

Internationalization

Internationalization

- The process of preparing an app for content localized for particular languages is called internationalization
- Ideally, the text, images, and other content displayed to users should be localized for multiple languages
- Candidates for localization include:
  - App text such as date, time, and number formatting
  - Static text such as an HTML-based help file
  - Icons and other images that contain text or have some culture-specific meaning
  - Sound files containing spoken language
  - NIB files

Getting Started

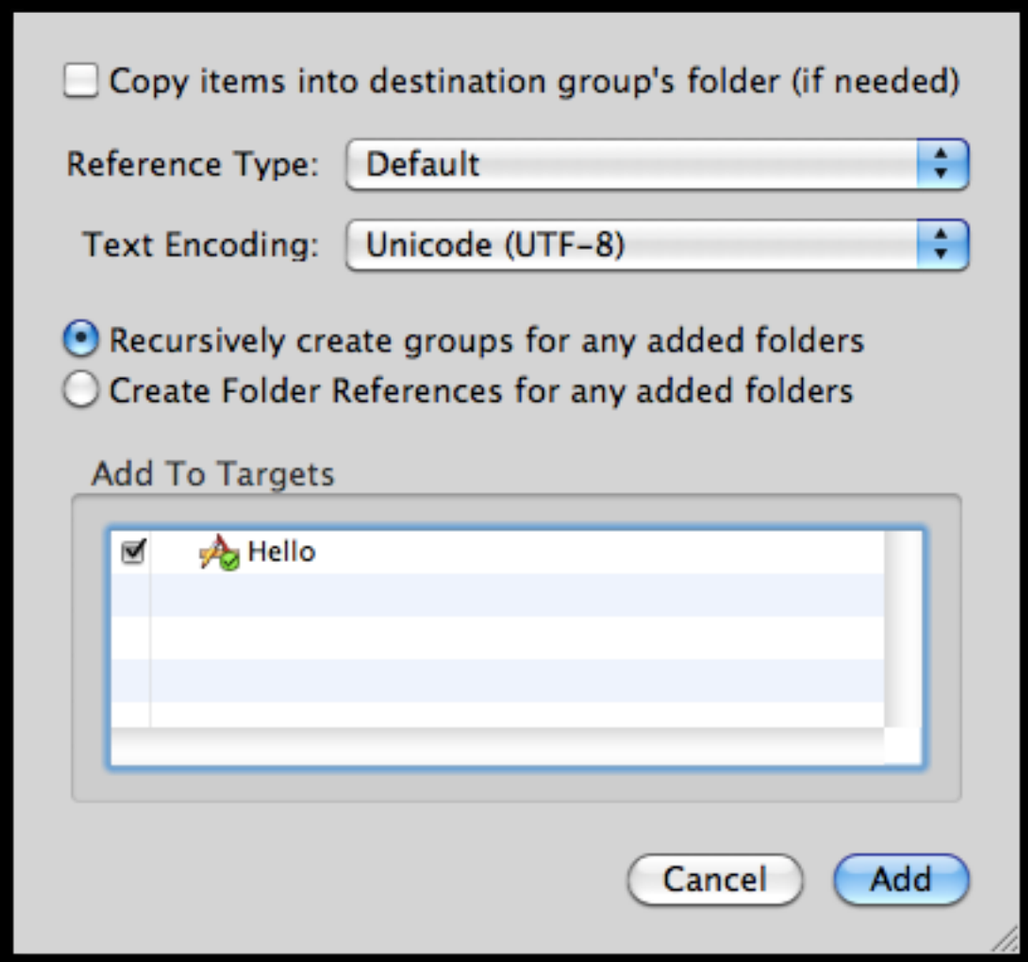
- For each language you wish to support, you will need to create a subdirectory in your app bundle containing localized resources for each language you wish to support
- These subdirectories are names using the ISO 639-1 language codes followed a ".lproj" suffix
- For example...
  - English → en.lproj
  - Spanish → es.lproj
  - Japanese → jp.lproj

Creating *.lproj Directories

- Remember that when you create a group (folder) in Xcode, that doesn't necessarily correspond to a directory on the underlying filesystem
- However, these language bundle directories must be actual underlying directories
- I create these by cd-ing into the project directory and using the mkdir command via the Terminal, though Finder is fine as well

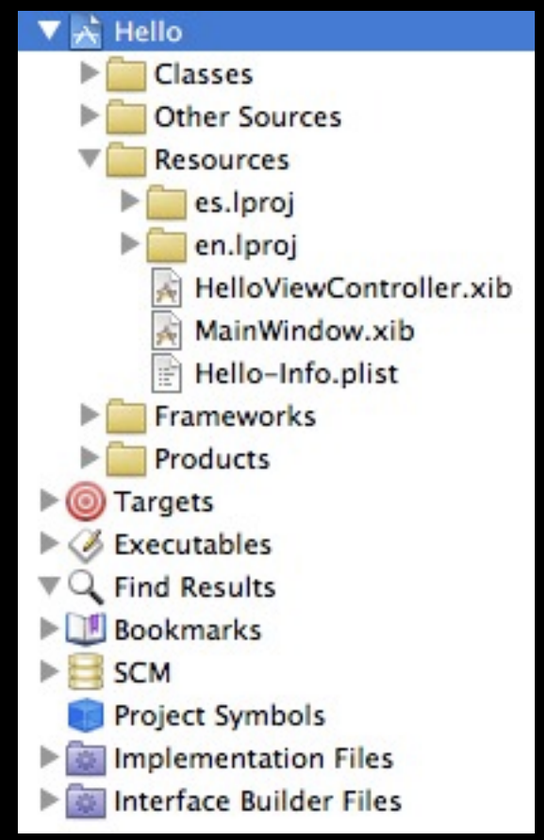
Importing *.lproj Directories

- Once you've created the directories under your project, you'll need to import them into Xcode
- You'll want to uncheck the copy box, as they're already there under your project
- I usually import them under resources



Imported Direcories

- Here we see the result of creating English and Spanish language directories for our app



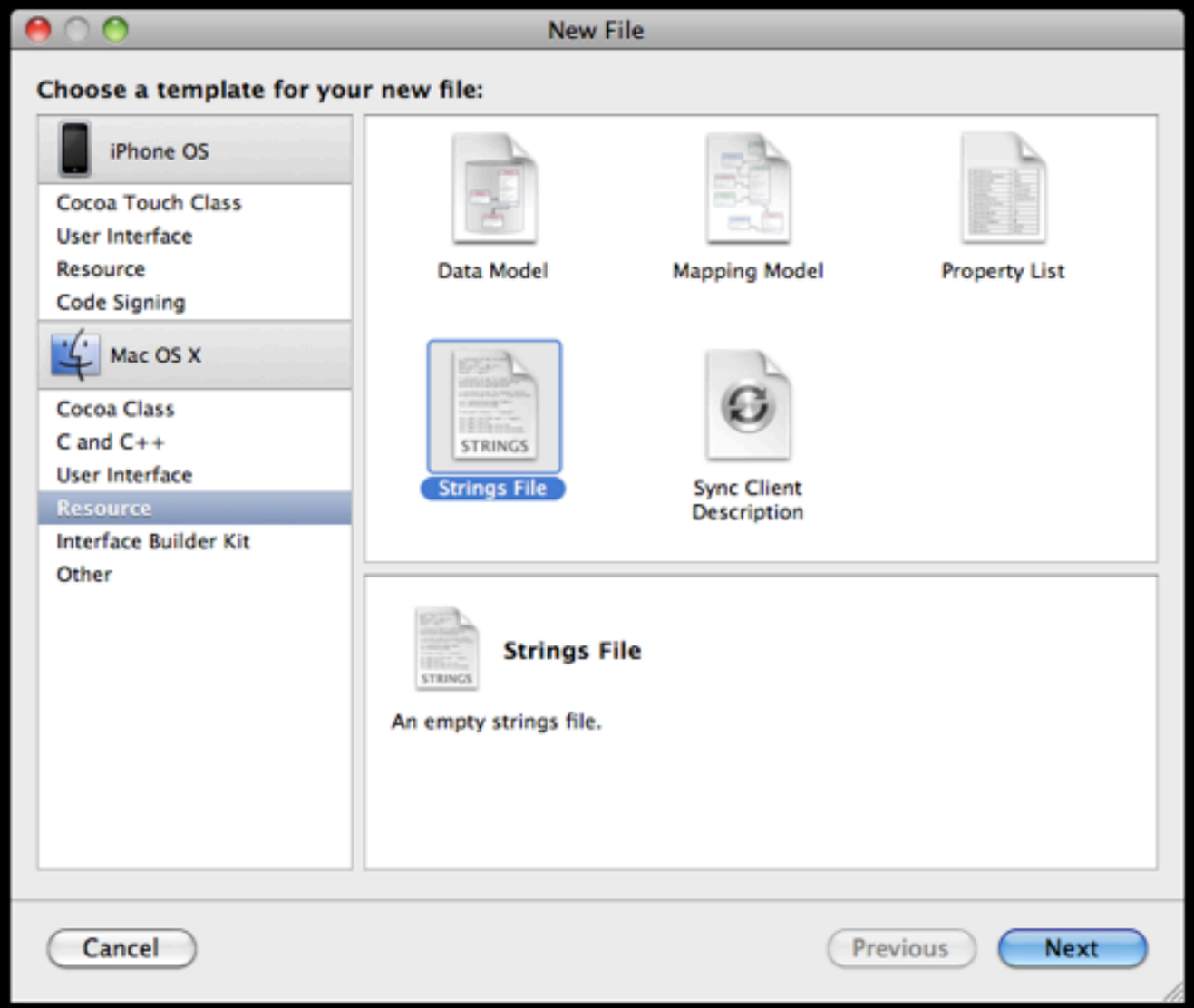
Strings Files

- Our localized information are contained in a file type known simply as “strings” files
- These files are basically key value pairs separated by and equals sign, each terminated by a semicolon
- For example, a strings file might look something like this...

```
/*
   Comments are C-style
 */

UNQUOTED_KEY1 = "Value 1";
"QUOTED_KEY2" = "Value 2";
```

Creating a Strings File

- Under the new file dialog there's an option for strings file...



App Name and Icon

- Let's start by customizing both the name and icon of your app using the language bundles
- To do so, under each *.lproj directory create a new strings file called InfoPlist.strings
- The keys of interest for InfoPlist.strings files are...
  - CFBundleDisplayName — the name to display for your app
  - CFBundleIconFile — the icon file to use for your app
- Note that these keys correspond to the values which are also in the *-Info.plist file

en.lproj/InfoPlist.strings

```
/*

   InfoPlist.strings
   Hello
   English

 */

CFBundleDisplayName = "Hello";
CFBundleIconFile = "Icon-en-57px.png";
```

Icon-en-57px.png

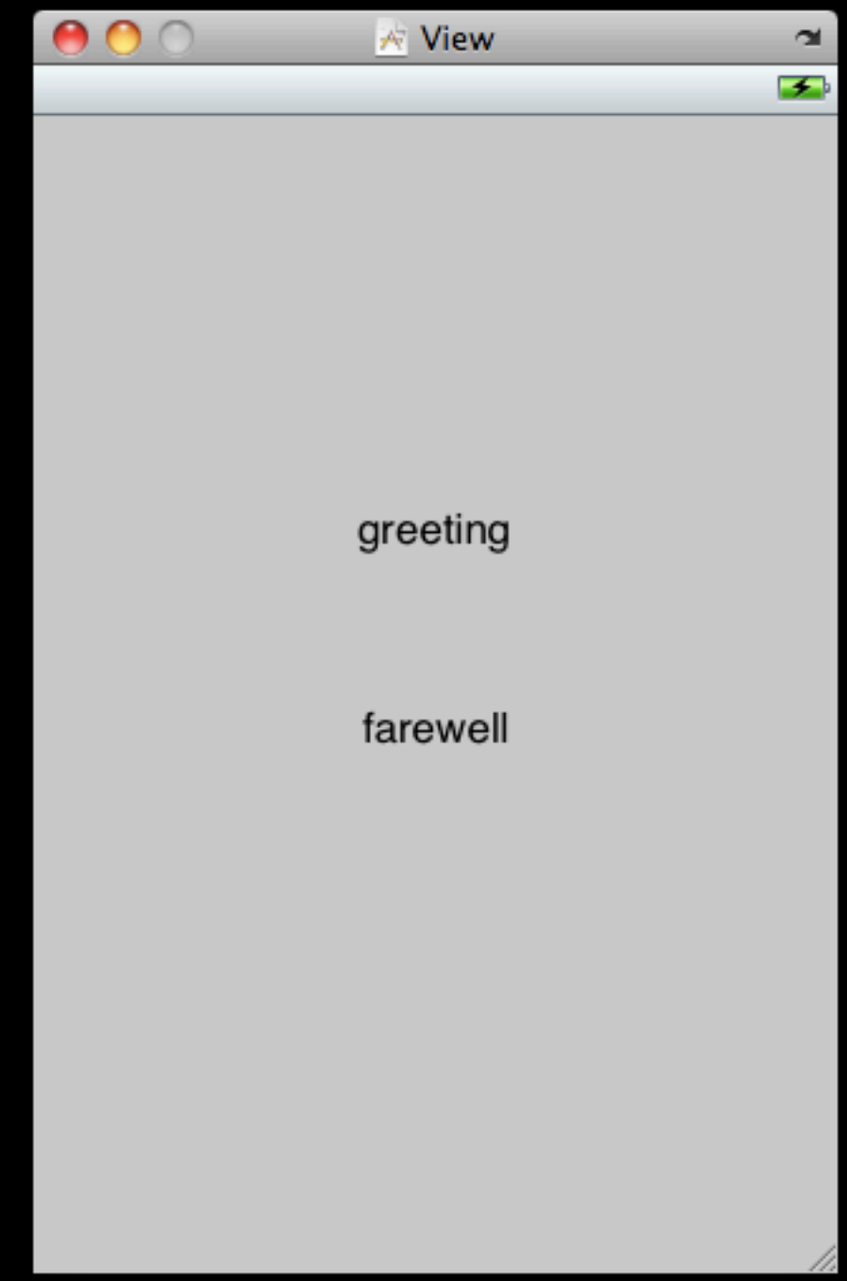
es.lproj/InfoPlist.strings

```
/*

   InfoPlist.strings
   Hello
   Spanish

 */

CFBundleDisplayName = "Hola";
CFBundleIconFile = "Icon-es-57px.png";
```

Icon-es-57px.png

HelloViewController.xib

- As a basic demo, we'll simply create a view based app which contains 2 labels
  - A greeting
  - A farewell
- We'll provide localized versions of these strings in our language bundles and substitute them into the app for these values at runtime



Creating a Localized String File

- To provide values for these placeholders, we need to provide localized strings for each language
- Again, we do this with a strings file — in each *.lproj directory you will need to create a file called “Localizable.strings” to define the localized strings
- The keys for the strings in these file will be used to lookup the localized strings, so they are generally abstracted
- For example “greeting” might be the key used for “Hello” in the English Localizable.strings file

en.lproj/Localizable.strings

```
/*
   Localizable.strings
   Hello
   English

 */

greeting = "Hello!";
farewell = "Goodbye!";
```

es.lproj/Localizable.strings

```
/*
   Localizable.strings
   Hello
   Spanish

 */

greeting = "¡Hola!";
farewell = "¡Adiós!";
```

Reading the Localized Strings

- To lookup the localized form of a string we utilize the following macro...

```
NSLocalizedString(key, comment)
```

- This method takes the lookup key as its first argument and simply looks up the value for that key in the appropriate language bundle and returns the string

HelloViewController.h

```
#import <UIKit/UIKit.h>

@interface HelloViewController : UIViewController {

  UILabel *greeting;
  UILabel *farewell;

}
@property(nonatomic, retain) IBOutlet UILabel *greeting;
@property(nonatomic, retain) IBOutlet UILabel *farewell;

@end
```

HelloViewController.m

```
#import "HelloViewController.h"

@implementation HelloViewController

@synthesize greeting;
@synthesize farewell;

- (void)viewWillAppear:(BOOL)animated {
  [super viewWillAppear:animated];
  self.greeting.text = NSLocalizedString(@"greeting", @"Label for greeting");
  self.farewell.text = NSLocalizedString(@"farewell", @"Label for farewell");
}

- (void)dealloc {
   self.greeting = nil;
   self.farewell = nil;
    [super dealloc];
}

@end
```

Switching Languages in the iPhone OS

- You can change the current language by opening the Settings app, selecting “General”, then “International” then “Language”...

The Resulting App

La App Resultante

Localization Comments

- You shouldn't rely on something like Google translate to obtain localized strings for a given language — as these things tend to not end well
- There are also issues of color, symbols, etc. that should be considered
- Some languages use less space, others more this may require different NIBs for different languages
- Right to left languages may requires different layouts as well

Additional Resources

- Handling Critical Application Tasks section of the iPhone Application Programming Guide
  - http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ApplicationEnvironment/ApplicationEnvironment.html
- iPhone Human Interface Guidelines
  - http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/
- Internationalization Programming Topics
  - http://developer.apple.com/iphone/library/documentation/MacOSX/Conceptual/BPInternational/