Getting Started

Introduction to iPhone Development
IAP 2010 ❄

iphonedev.csail.mit.edu

edward benson / eob@csail.mit.edu

Today

- The Toolchain
- Starting a Project
- iPhone Application Structure
- Objective-C Crash Course
- Data Persistence with CoreData

The iPhone Toolchain

XCode

Objective-C, GDB

Interface Builder

Graphical UI Development

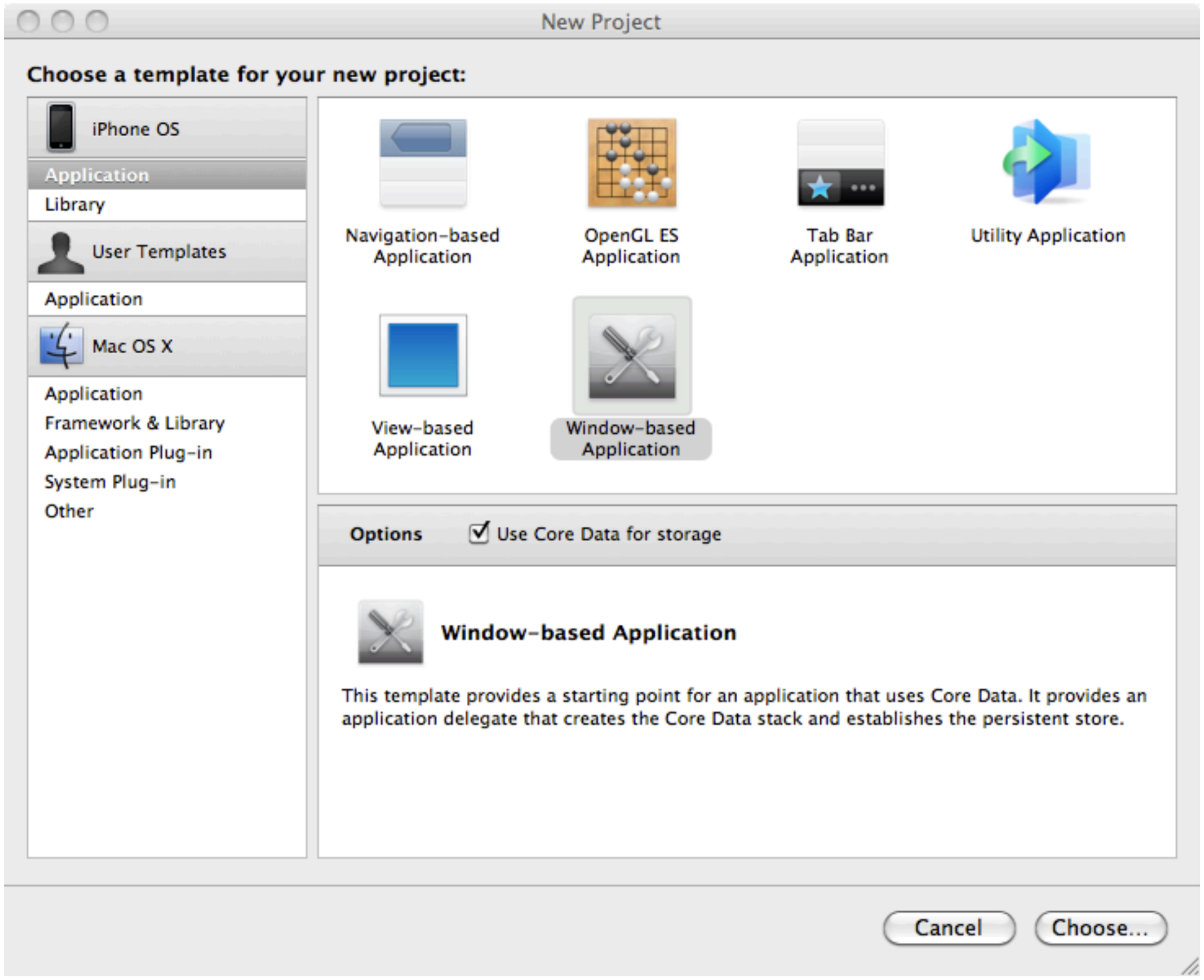
Instruments

Profiling, Leak Finding

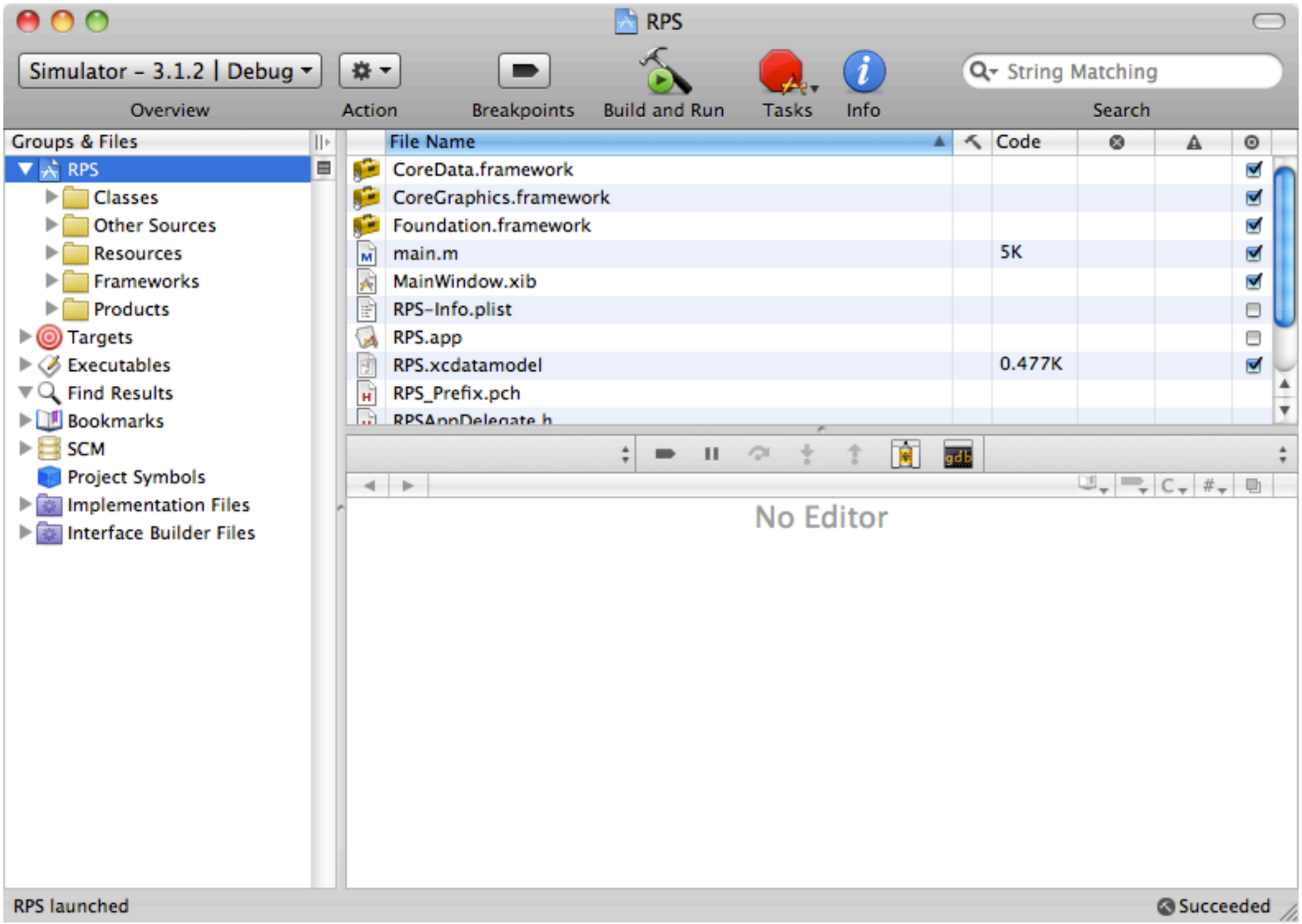
Simulator

Testing

Starting a Project

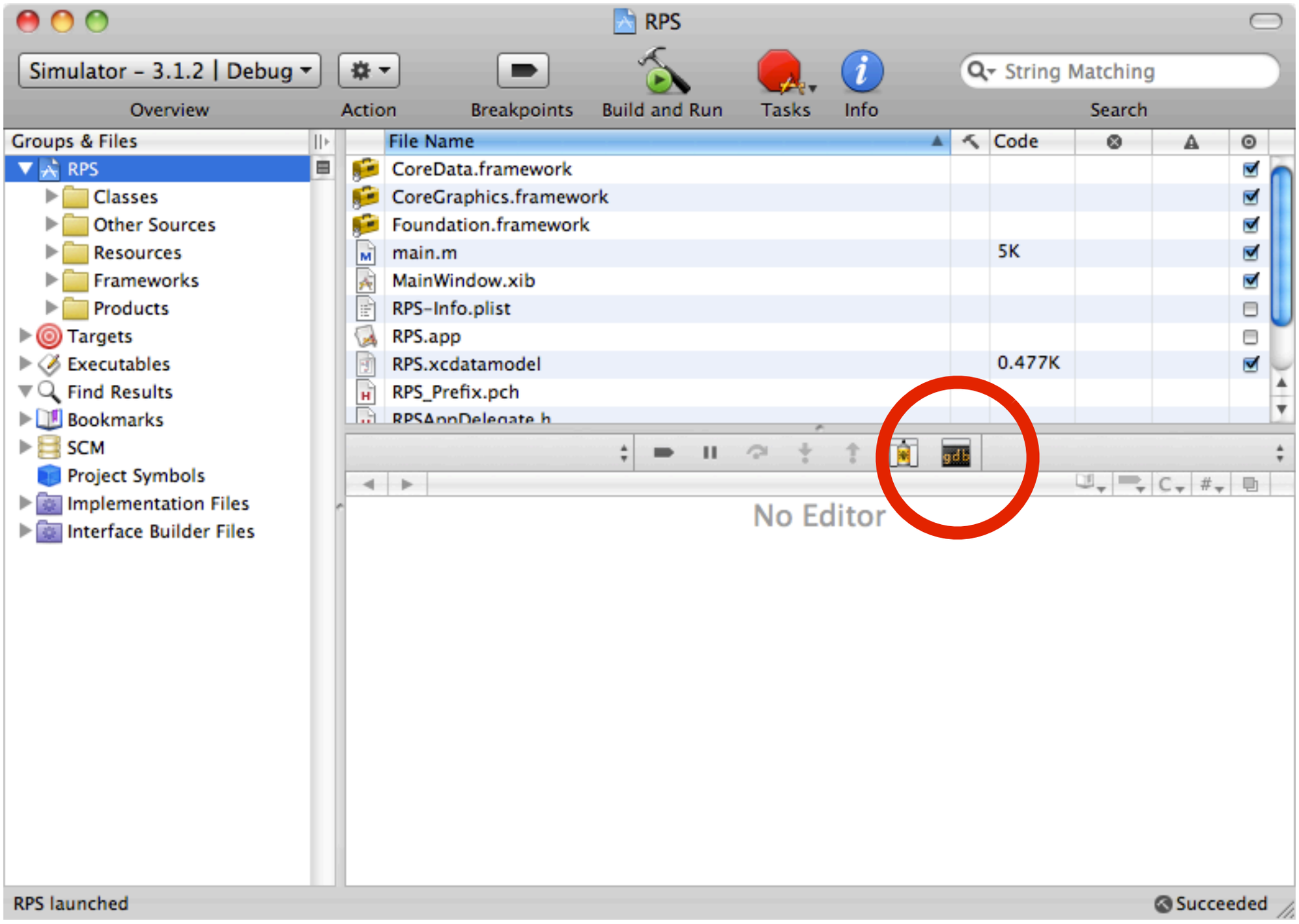

New Project
Choose a template for your new project:
iPhone OS
Application
Library
User Templates
Application
Mac OS X
Application
Framework & Library
Application Plug-in
System Plug-in
Other
Navigation-based Application
OpenGL ES Application
Tab Bar Application
Utility Application
View-based Application
Window-based Application
Options
Use Core Data for storage
Window-based Application
This template provides a starting point for an application that uses Core Data. It provides an application delegate that creates the Core Data stack and establishes the persistent store.
Cancel
Choose...

Your Project

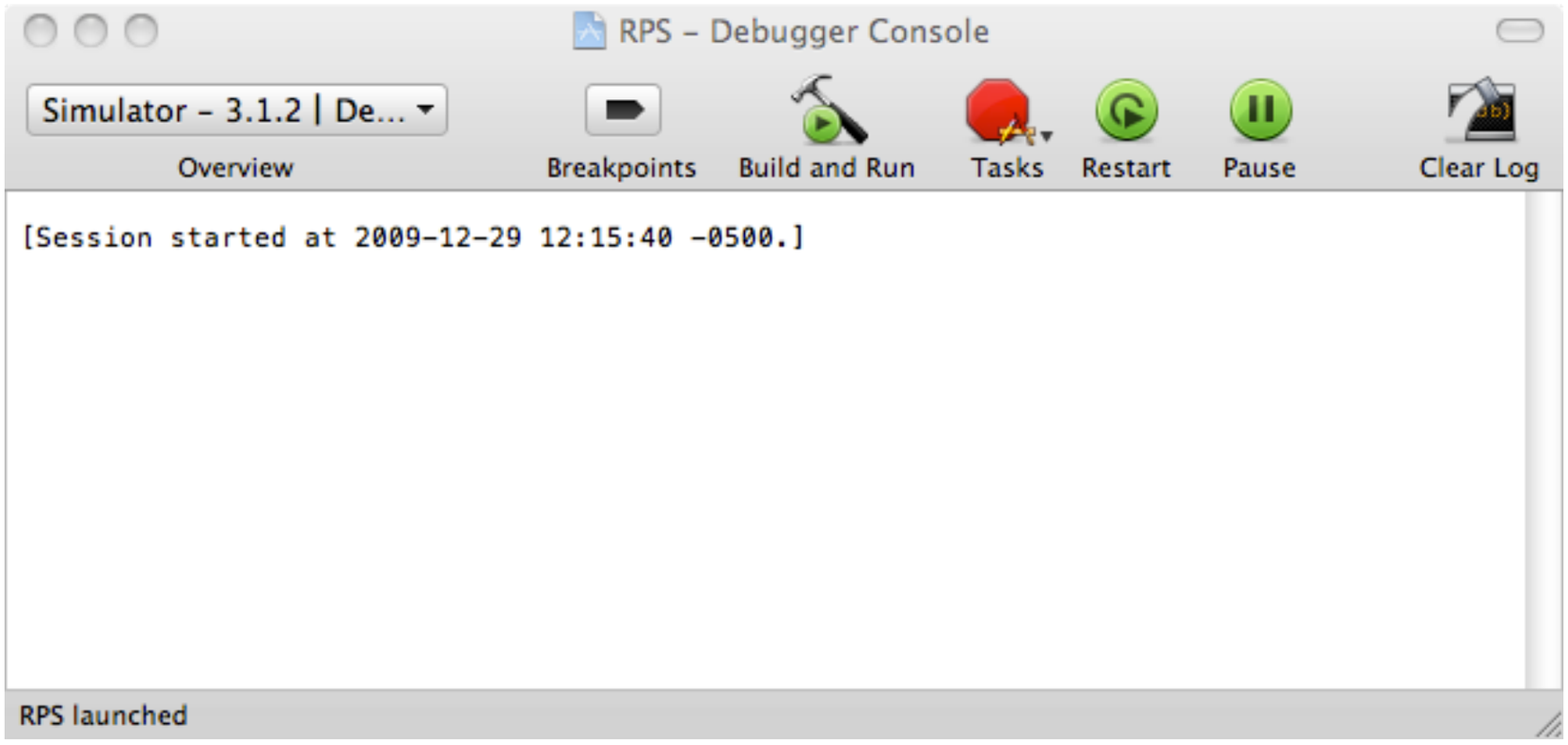


iPhone Simulator

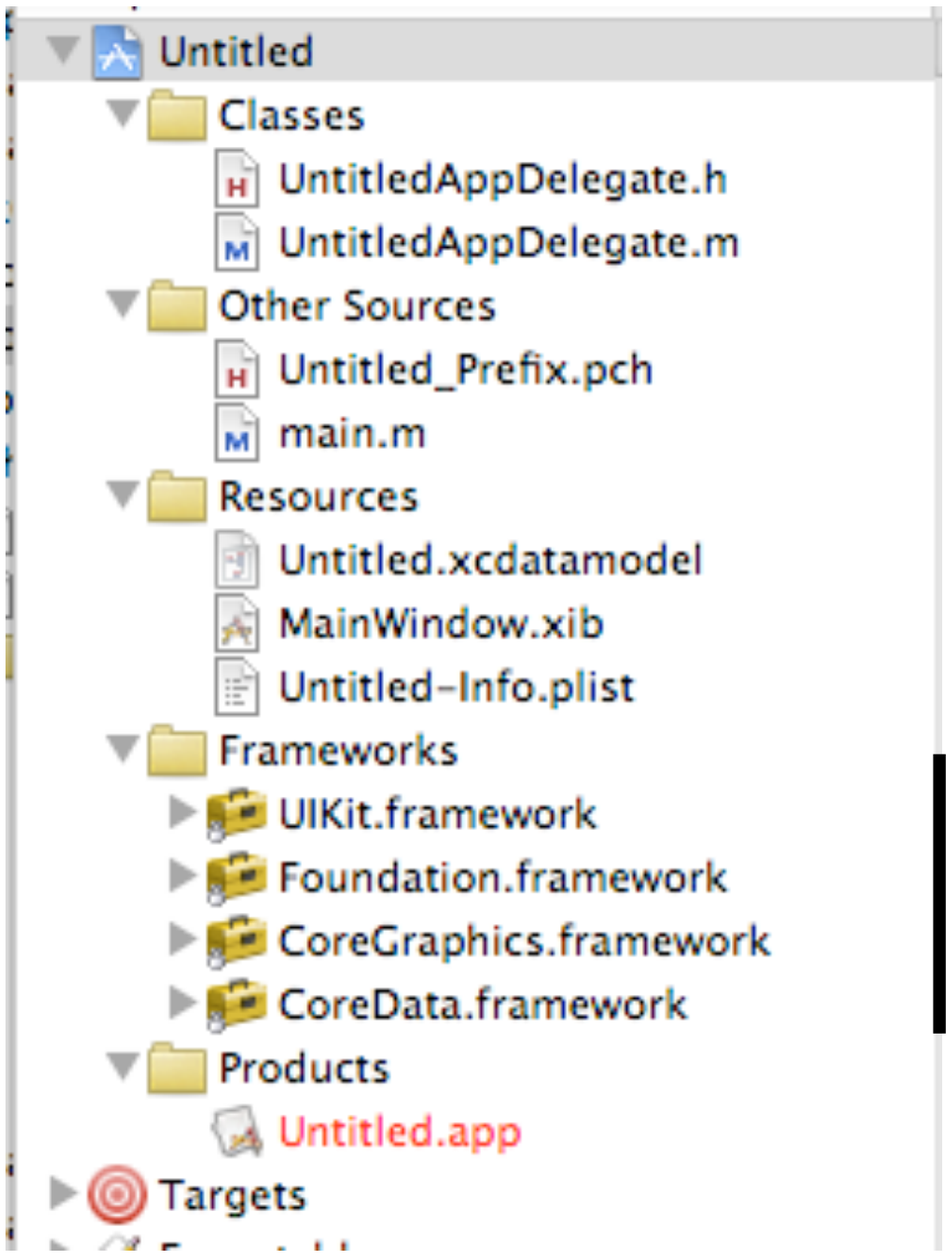
The Debug View



The Debug View

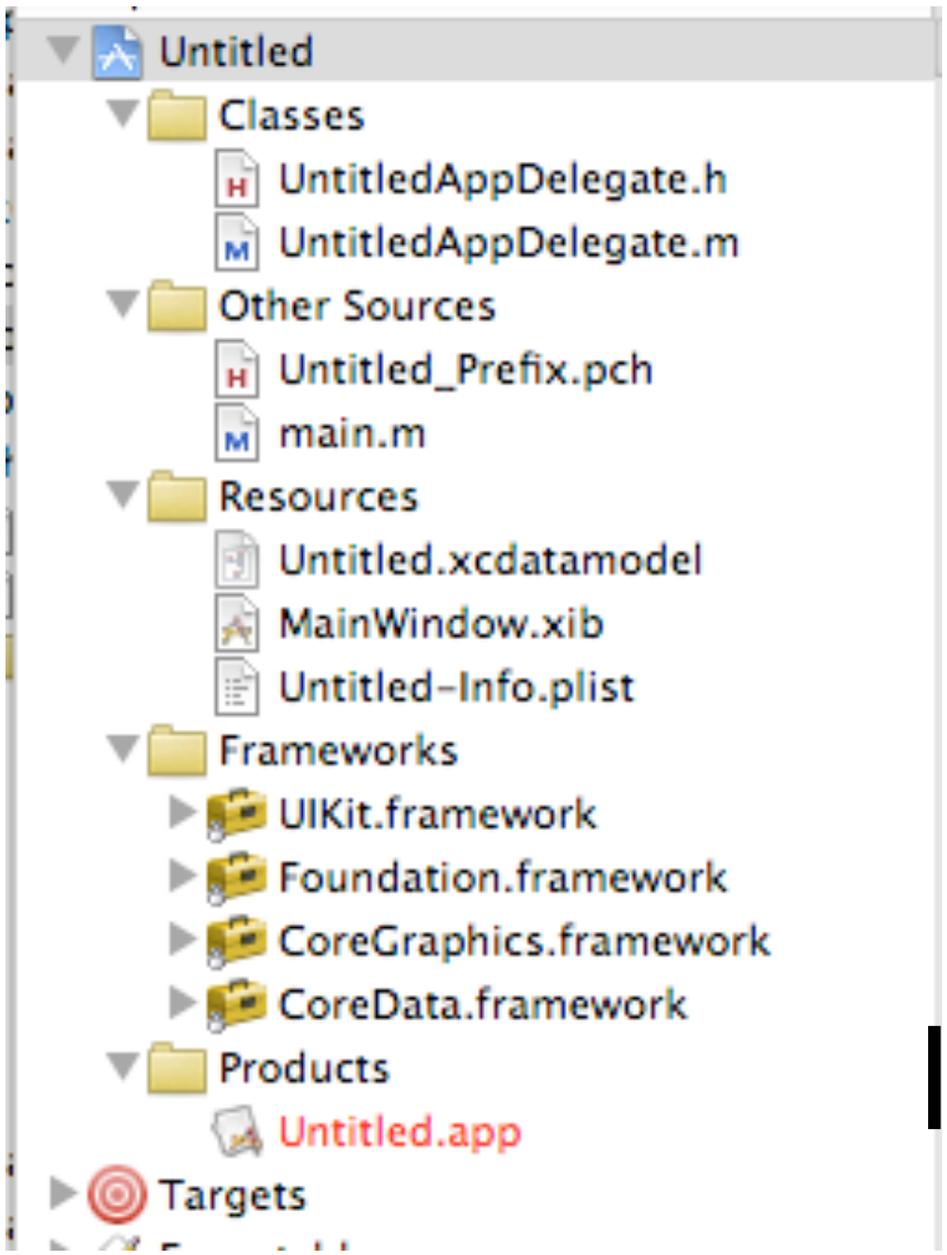


The iPhone Application Structure

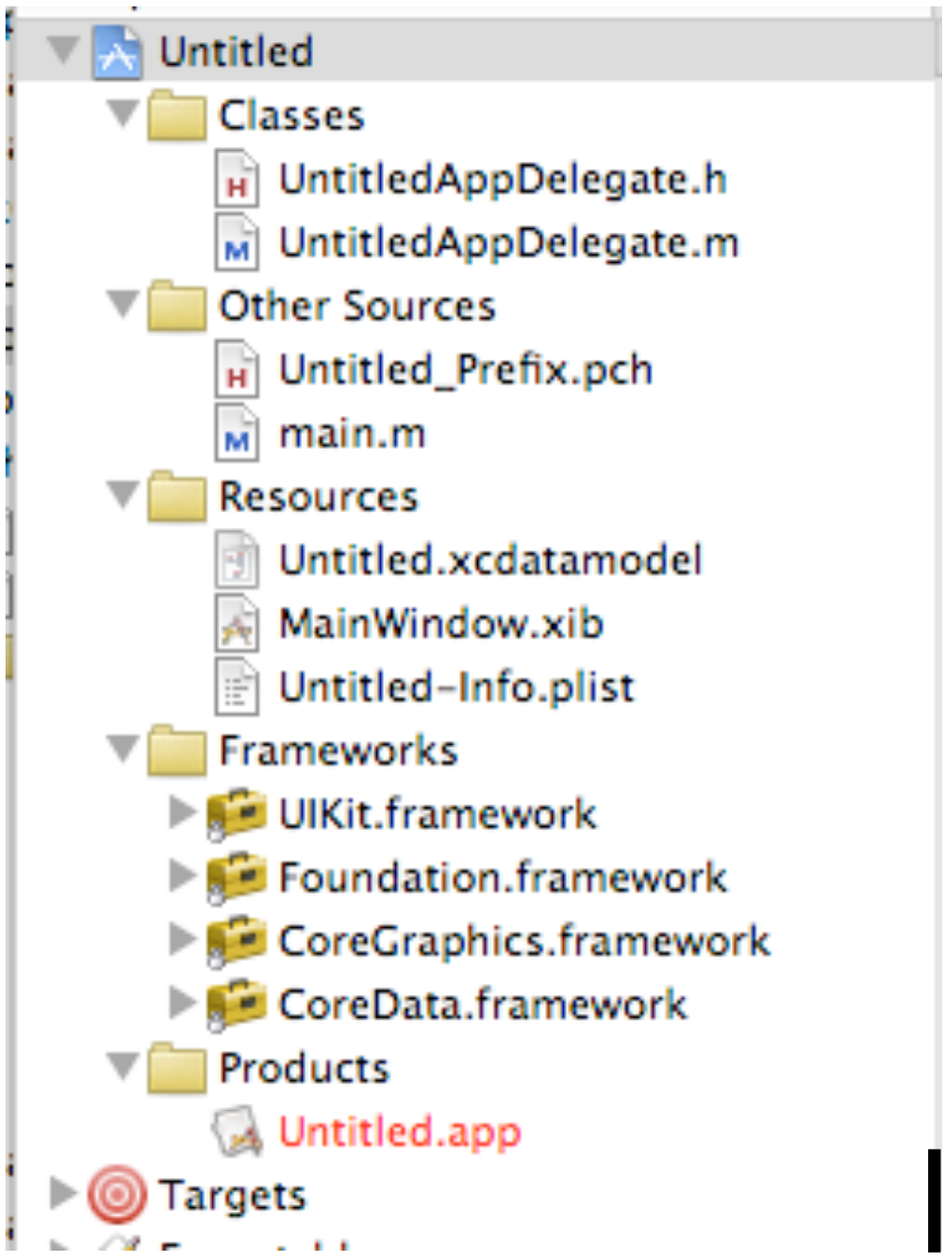


Linked Frameworks

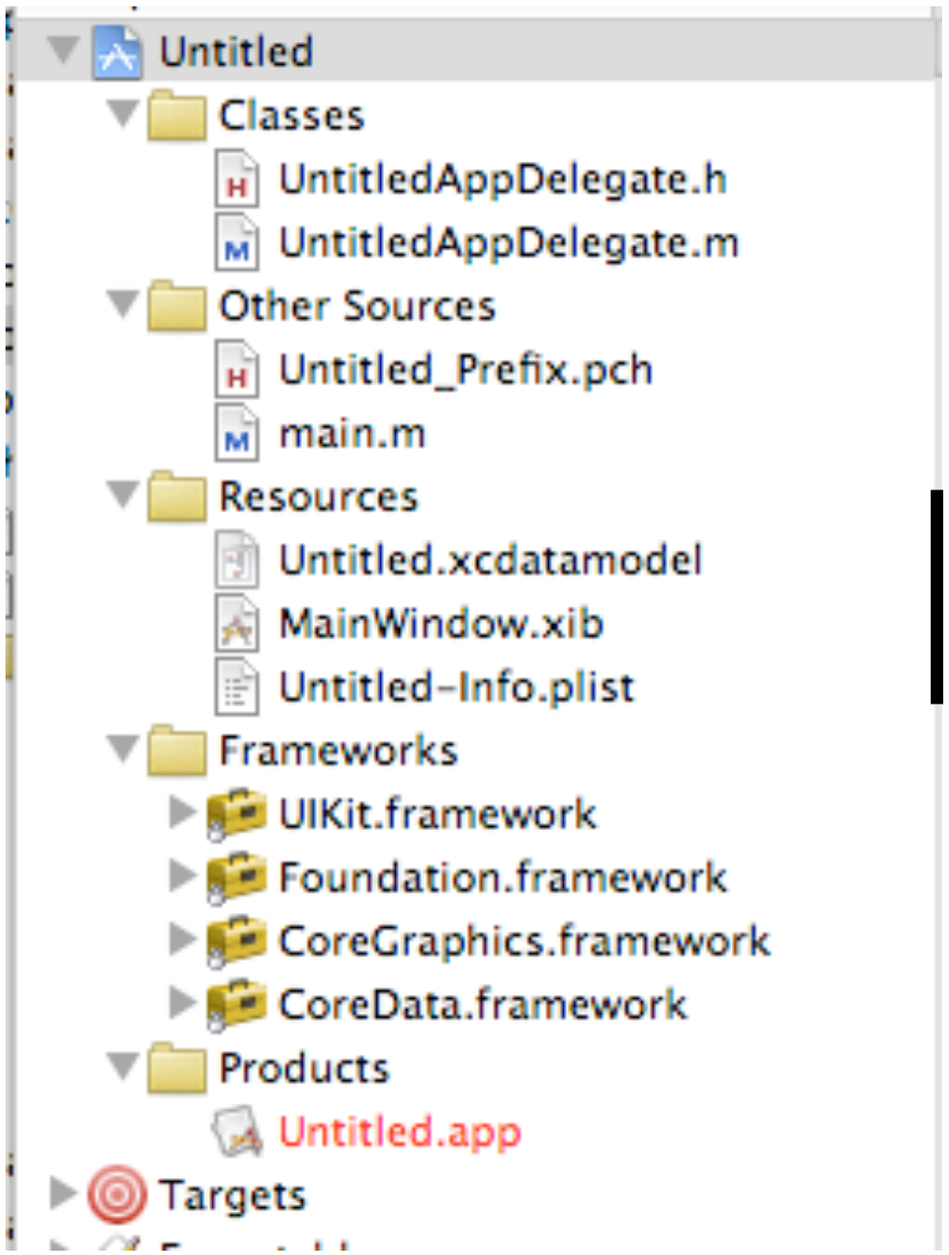
Graphics, sound, bluetooth, etc



The Executable

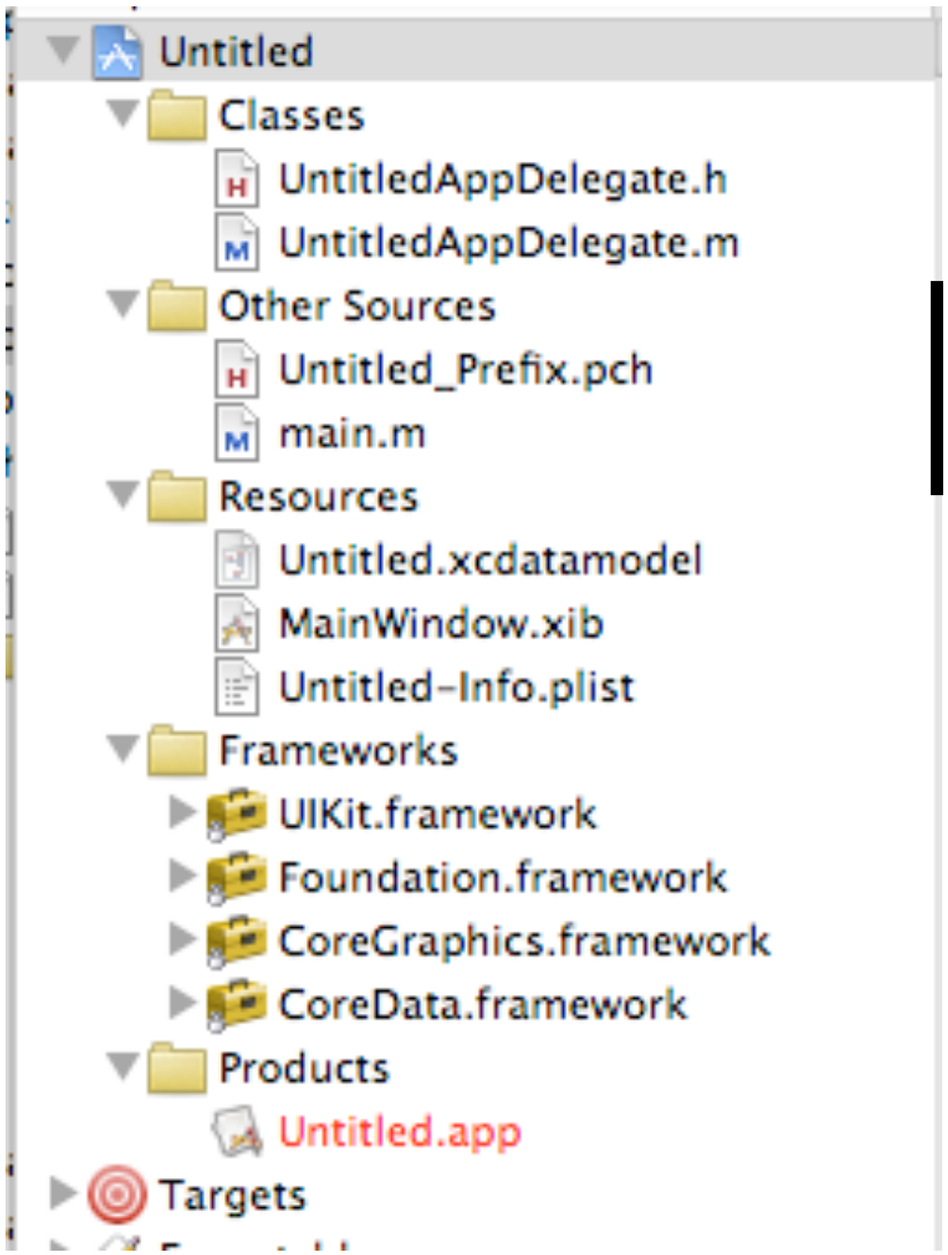


Targets (different build settings)

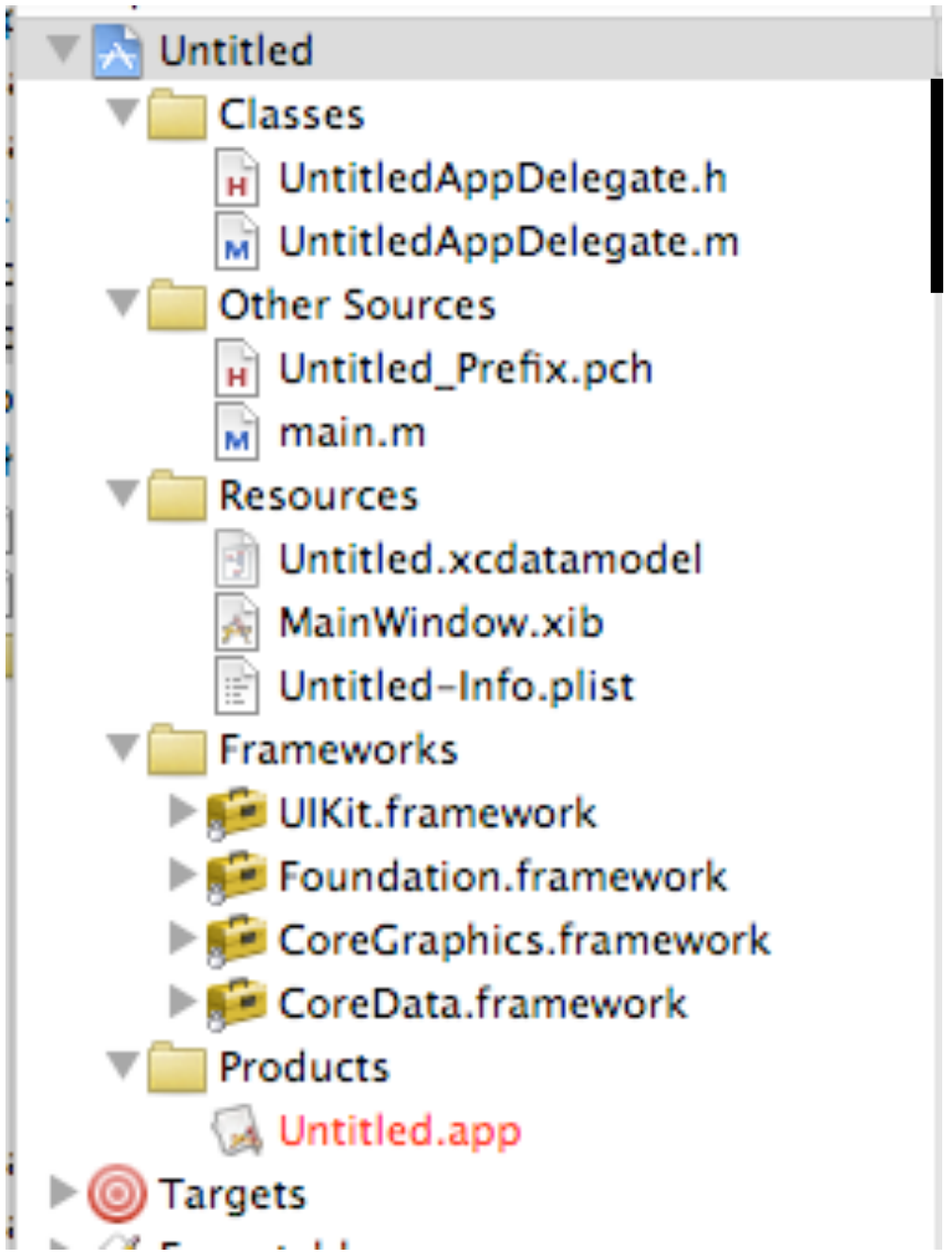


Resources

Images, sounds, data, IB files



Boilerplate Code



Your App's Code

These folders, called *groups* are just abstractions to help you organize your project -- they don't even exist in the filesystem.

Rearrange however you want.

UI-Driven Programming

Nearly everything in your **entire** project is essentially just a callback.

main.m

```
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

This is the entire main routine!

UI-Driven Programming

So where is your hook to implement code?

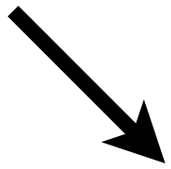
`UntitledAppDelegate.m`

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after app launch

    [window makeKeyAndVisible];
}
```

... the **applicationDidFinishLaunching** callback

UI-Driven Programming

UIApplication



UIAppDelegate

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
```

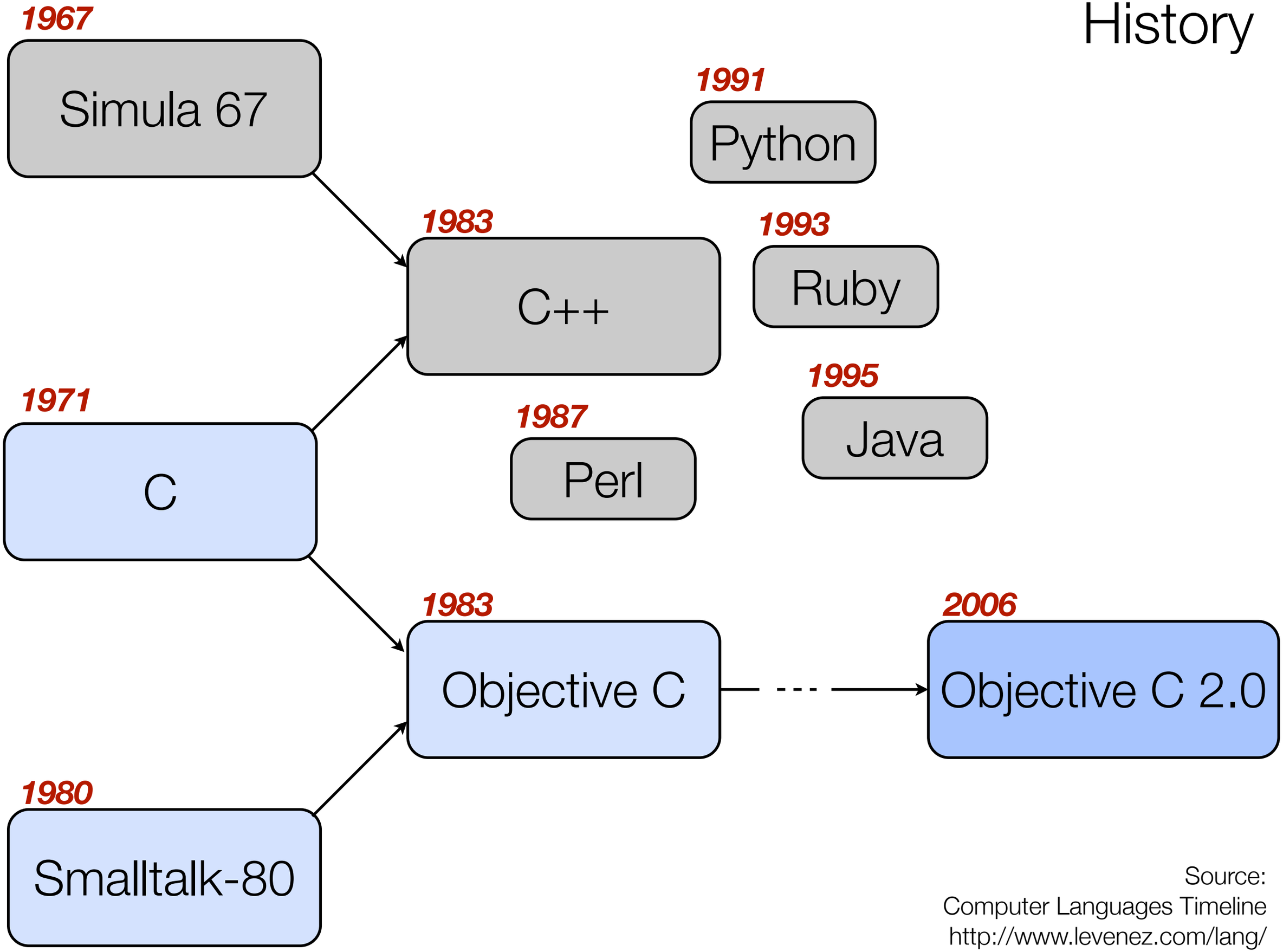
Initialize your User Interface

```
    [window makeKeyAndVisible];
}
```

After which point your app is almost entirely

Objective-C

History



Source:
Computer Languages Timeline
http://www.levenez.com/lang/

Objective-C

- **Primitives & Strings**
- Objects, Messages, and Properties
- Memory Management

Primitives

The usual C Types

`int, float, ...`

It's own boolean *(ObjC forked before C99)*

`BOOL`

*Takes values **NO**=0 and **YES**=1*

Some special types

`id, Class, SEL, IMP`

nil is used instead of **null**.

Strings

*Always use (NSString *) instead of C Strings unless you know what you're doing!!*

Inline

```
@"This is an inline string";
```

Assigned

```
NSString *str = @"This is assigned to a variable";
```

If you accidentally leave out the @, expect to crash!

NSLog

While you're getting to know Objective-C,

NSLog

is your best friend.

(Or just use the debugger)

Exercise 1 - See, it's like C

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    int i;
    for (i=0; i<10; i++) {
        NSLog(@"Hello, word!");
    }

    [window makeKeyAndVisible];

}
```

Exercise 2 - Broken strings and printf-style logging

1) Remove the @ before the string and see what happens

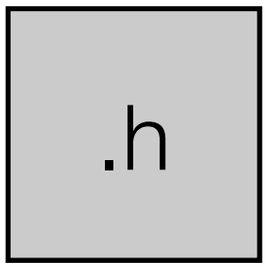
2) Try `NSLog(@"Hello, word! %i", i);`

Overview

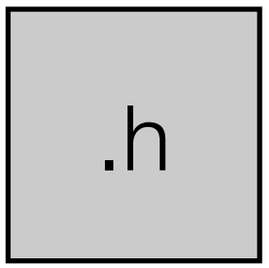
- Primitives & Strings
- **Objects, Messages, and Properties**
- Memory Management

Declaring

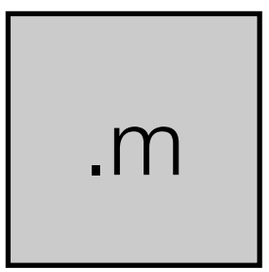
Objects



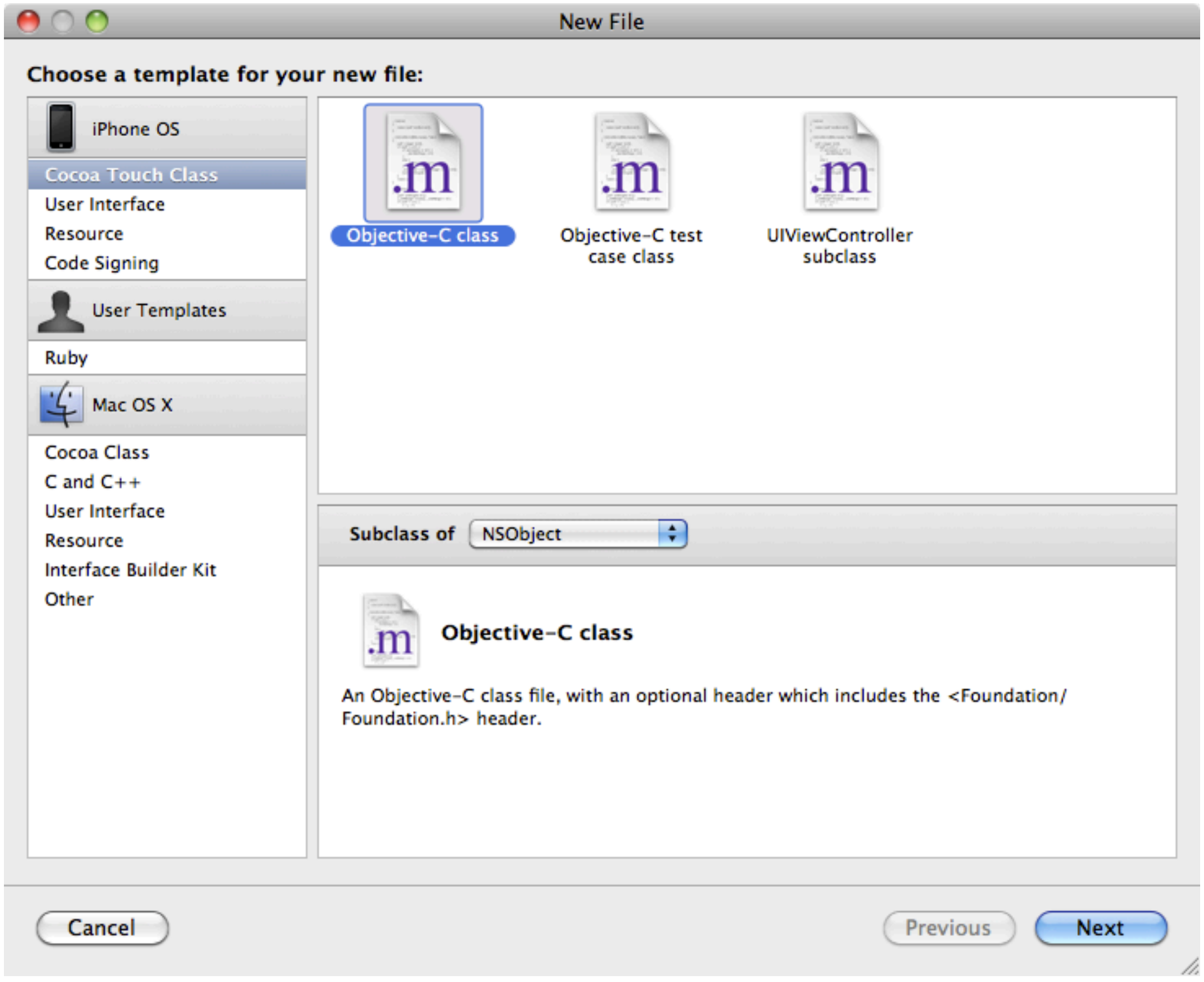
Interface



Protocol



Implementation



Name it **RPSGame**

Exercise 3 - Creating an object

Back in the app delegate...

```
#import "RPSGame.h"
```

And then

```
RPSGame *game = [[RPSGame alloc] init];
NSLog(@"I have a game: %@", game);
```

Objects - Typing

Every object is of type

`id`

This is a pointer to the instance data of the object.

```
id game;
```

Of course, you can also declare a more specific type.

```
RPSGame * game;
```

Equivalent Statements

```
RPSGame *game = [[RPSGame alloc] init];
```

```
id game = [[RPSGame alloc] init];
```

Methods and Messages

Messages

Method Calling v. Message Passing

Messages

With no arguments

[object message];

Messages

With no arguments

[object message];

With 1 arguments

[object message:value];

Messages

With no arguments

```
[object message];
```

With 1 arguments

```
[object message:value];
```

With 2 arguments

```
[object message:value arg2:value];
```

Messages

With no arguments

```
[aPerson init];
```

With 1 arguments

```
[aPerson initWithFirst:@"Ted"];
```

With 2 arguments

```
[aPerson initWithFirstAndLast:@"Ted" last:@"Benson"];
```

You can send messages to **classes**

```
[Person alloc];
```

You can **nest** messages

```
Person* p = [[Person alloc] initWithName:@”Ted”];
```

equal to

```
Person* p = [Person alloc];
[p initWithName:@”Ted”];
```

Defining Methods

To Call

```
[aPerson initWithFirstAndLast:@"Ted" last:@"Benson"];
```

To Define

```
- (id)initWithFirstAndLast:(NSString*)firstName
                      last:(NSString*)lastName;
```

Exercise 4 - A simple method, a simple message

RPSGame.h

```
@interface RPSGame : NSObject {
}

-(NSString *)getWinnerName;

@end
```

RPSGame.m

```
@implementation RPSGame

-(NSString *)getWinnerName {
  return @"Ted";
}

@end
```

App Delegate

```
NSLog(@"The winner was: %@", [game getWinnerName]);
```

Instance Variables

```objc
@interface RPSGame : NSObject {
  NSString *winnerName;
  NSString *loserName;
}
```

```objc
int someInt;
float someFloat;
id untypedObject;
// etc etc
```

Initialization

The **init** convention

- Objective-C has a lot of conventions that are only enforced by its programmers, not the compiler
- Unfortunately, you just have to learn these

```
[[RPSGame alloc] init]
```

`+(id)alloc;` Allocates memory and returns a pointer.

`-(id)init;` Initializes the newly allocated object.

The **init** convention

```
-(id)init;
[[RPSGame alloc] init]


-(id)initWithAwesomeness:(BOOL)isAwesome;
[[RPSGame alloc] initWithAwesomeness:YES]


-(id)initWithPlayer1:(NSString *)p1 player2:(NSString *)p2;
[[RPSGame alloc] initWithPlayer1:@"Mario" player2:@"Luigi"]
```

Exercise 5 - Initialization

RPSGame.h

```
@interface RPSGame : NSObject {
  NSString *winnerName;
  NSString *loserName;
}

-(id)init;
```

RPSGame.m

```
@implementation RPSGame

-(id)init {
  if (self = [super init]) {
     winnerName = nil;
     loserName = nil;
  }
  return self;
}


-(NSString *)getWinnerName {
  return winnerName;
}
```

Exercise 6 - Mutators

```
-(NSString *)setWinnerName:(NSString *)name;
```

```
-(NSString *)setWinnerName:(NSString *)name {
  winnerName = [name copy];
}
```

```
[game setWinnerName:@"Mario"];
```

Properties

Properties

```
@interface TodoItem : NSObject {

    int dbkey;
   BOOL complete;
   int priority;
   NSString * title;
   NSDate * due;
}

@end
```

These all need getters and setters.

Writing getters and setters is annoying.

Answer: **Properties**.

Think of them as compiler macros that generate the getter and setter for you.

Interface

```
@property (nonatomic, copy) NSString *winnerName;
```

Implementation

```
@synthesize winnerName, loserName;
```

Properties

```
@interface TodoItem : NSObject {
   int dbkey;
  NSString * title;
}

@property (readonly) int dbkey;
@property (nonatomic, retain) NSString *title;

@end
```

```
#import "TodoItem.h"

@implementation TodoItem

@synthesize title, dbkey;

@end
```

You are still responsible for cleaning up memory for this object!

Property Attributes

@property (**attributes**) type name;

Writability

readwrite (default)
readonly

Atomicity

nonatomic
(no “atomic” attribute
but this is the default)

Setter Semantics

assign (default)
retain
copy

Source

http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Articles/chapter_5_section_3.html

Calling Properties

```
@property (nonatomic, copy) NSString *winnerName;
```

Will allow you to use “dot notation”

```
game.winnerName = @"Something";

a = game.winnerName;
```

Or message passing

```
[game setWinnerName:@"Something"];

a = [game getWinnerName];
```

Exercise 7 - Properties

Interface

```
@interface RPSGame : NSObject {
  NSString *winnerName;
  NSString *loserName;
}

-(id)init;

@property (nonatomic, copy) NSString *winnerName;
@property (nonatomic, copy) NSString *loserName;

@end
```

Implementation

```
@implementation RPSGame
@synthesize winnerName, loserName;
```

Change the AppDelegate to use “dot” notation.

Recap

Objects
Instance Variables
Methods
Messages
Properties

Overview

- Primitives & Strings
- Objects, Messages, and Properties
- **Memory Management**

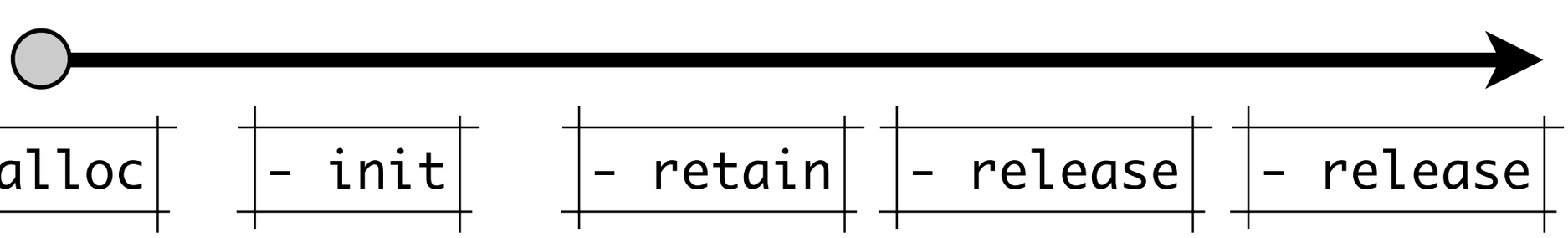
(if you're coming from a Python/Java/C# background, this is where things can get tricky)

Basic Idea

You need to help the Garbage Collector know when it is allowed to clean up an object.

Objective-C accomplishes this with a technique similar to reference counting.

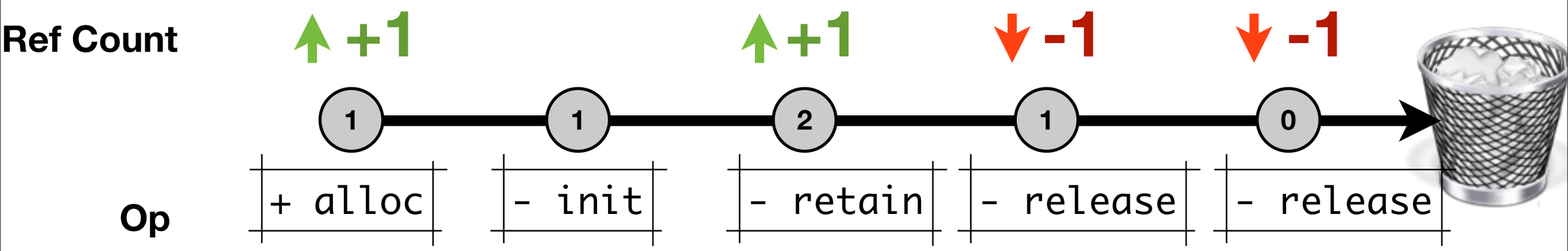
Object Lifecycle



main()	Create array	Initialize		Release from use
some_func()			Retain for use	Release from use

Memory Management

Object Lifecycle



main()	Create array	Initialize		Release from use
some_func()			Retain for use	Release from use

Image Credit: blog.tice.de

Recall creating an object

Almost always follows the pattern

```
TodoItem *item = [[TodoItem alloc] init];
```

`+ alloc`	Allocates the memory
`- init`	Performs the initialization

..So leaves you with a retain count of 1

Exercise 7 - Retain Count

In the App Delegate...

```
NSLog(@"The game's retain count is: %i", [game retainCount]);
```

Now try:

```
[game release];
NSLog(@"The game's retain count is: %i", [game retainCount]);
```

Why does it crash?

Most important commands that affect retain count

+alloc	**+1**	*Creating a new object*
-copy	**+1**	*Duplicating an object*
-retain	**+1**	*Reserving an object for your use*
-release	**-1**	*Releasing an object from your use*
-autorelease	**-1**	*Delayed release*

Best way to think about it

Forget about the count!

It means nothing to you, because the runtime will do crazy things to it.

Instead, think of *ownership*

When you want an object, **retain** (or **alloc**) it.
When you are done with an object, **release** it.

So in our app delegate

"I want an RPSGame"

```
+1  RPSGame *game = [[RPSGame alloc] init];
    NSLog(@"I have a game: %@", game);

    [game setWinnerName:@"Mario"];
    NSLog(@"The winner was: %@", [game getWinnerName]);

-1  [game release];
```

"OK, I'm done with the RPS Game"

If you just follow that mindset, you'll be memory leak free. But you must be vigilant!

Autorelease

Sometimes, you are **done** with an object
(so should release it!)

But you also want to return the object from a method.

Exercise 8 - Why do we need autorelease

In the App Delegate...

```
-(RPSGame *)createGame {
  RPSGame *game = [[RPSGame alloc] init];
  [game release];
  return game;
}
```

Now create your game like this:

```
RPSGame *game = [self createGame];
```

Why does it crash?

autorelease is like a delayed version of **release**.

It gives other parts of the code time to claim ownership of an object before it is swept up by the GC process.

Exercise 9 - Using autorelease

In the App Delegate...

```
-(RPSGame *)createGame {
  RPSGame *game = [[RPSGame alloc] init];
  return [game autorelease];
}
```

Now create your game like this:

```
RPSGame *game = [self createGame];
```

But this still isn't safe.... why?

Exercise 9 - Using autorelease

In the App Delegate...

Claim ownership

```
RPSGame *game = [[self createGame] retain];

.....

[game release];
```

Release ownership

Deconstructors

When is an object destroyed?

When it's retain count reaches 0

Then the deconstructor `- dealloc` is called

*Never call **dealloc** yourself -- this is always called automatically for you.*
(Except when you're calling [super dealloc] from within your dealloc implementation)

Exercise 10 - Fix our deconstructor

In RPSGame.m

```
-(void)dealloc {
  [super dealloc];
  [winnerName release];
  [loserName release];
}
```

Phew!

Tomorrow we start the iPhone part

Great Objective C Resources

- Cocoa Dev Central
  http://cocoadevcentral.com/d/learn_objectivec/

- The Objective-C 2.0 Programming Language
  http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

- Stanford's CS 193
  http://www.stanford.edu/class/cs193p/cgi-bin/index.php

- BYU's CocoaHeads Chapter
  http://cocoaheads.byu.edu/resources