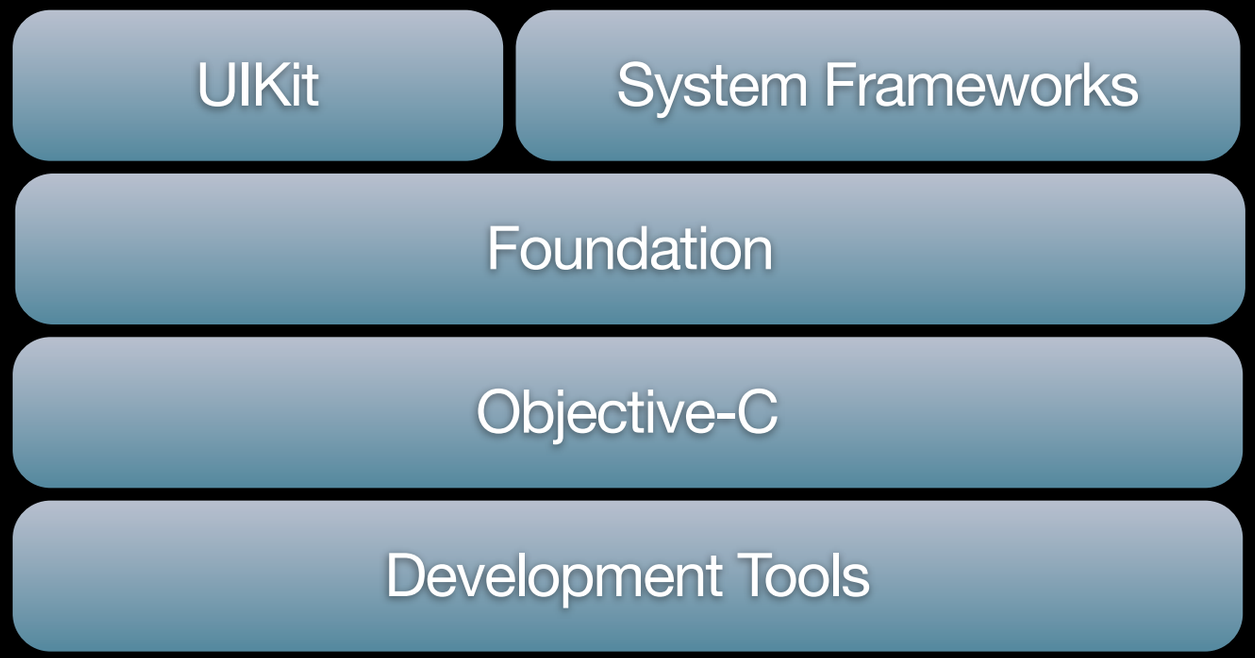
Foundation

Session 103

Jeremy Foo
VP Mobile, Lobang Club

Development Infrastructure

How everything fits together

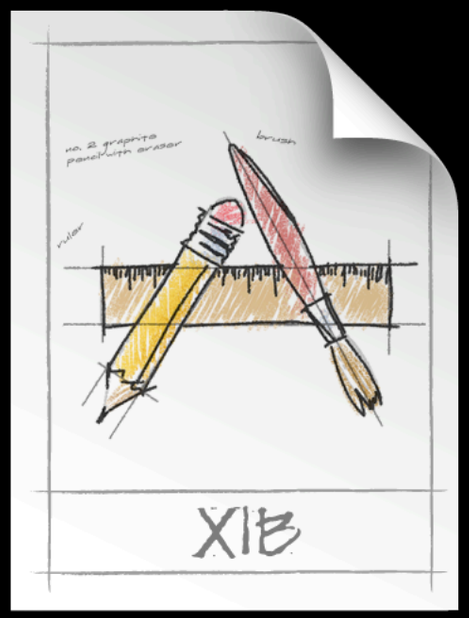


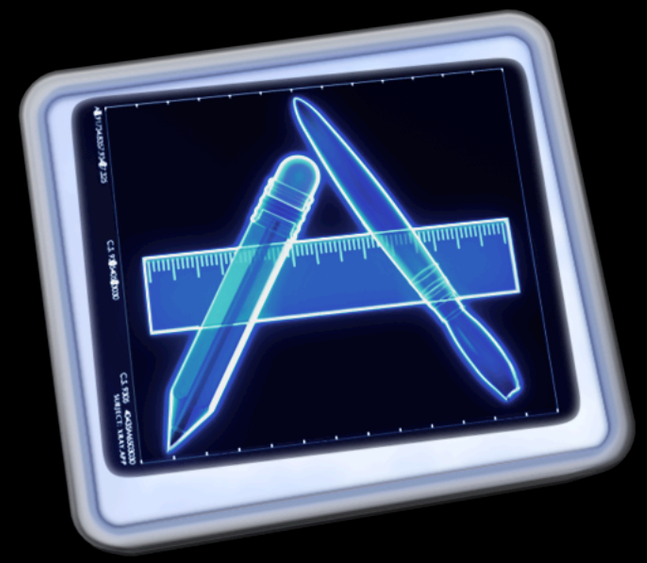
Development Tools

Xcode Lay of the Land

Development Tools

Free on the Mac App Store





Xcode

The lay of the land

- LLVM Integrated IDE
  - Code correction/completion
  - Syntax Highlighting
- Revision Control and Snapshotting
- Developer portal Integration

Xcode

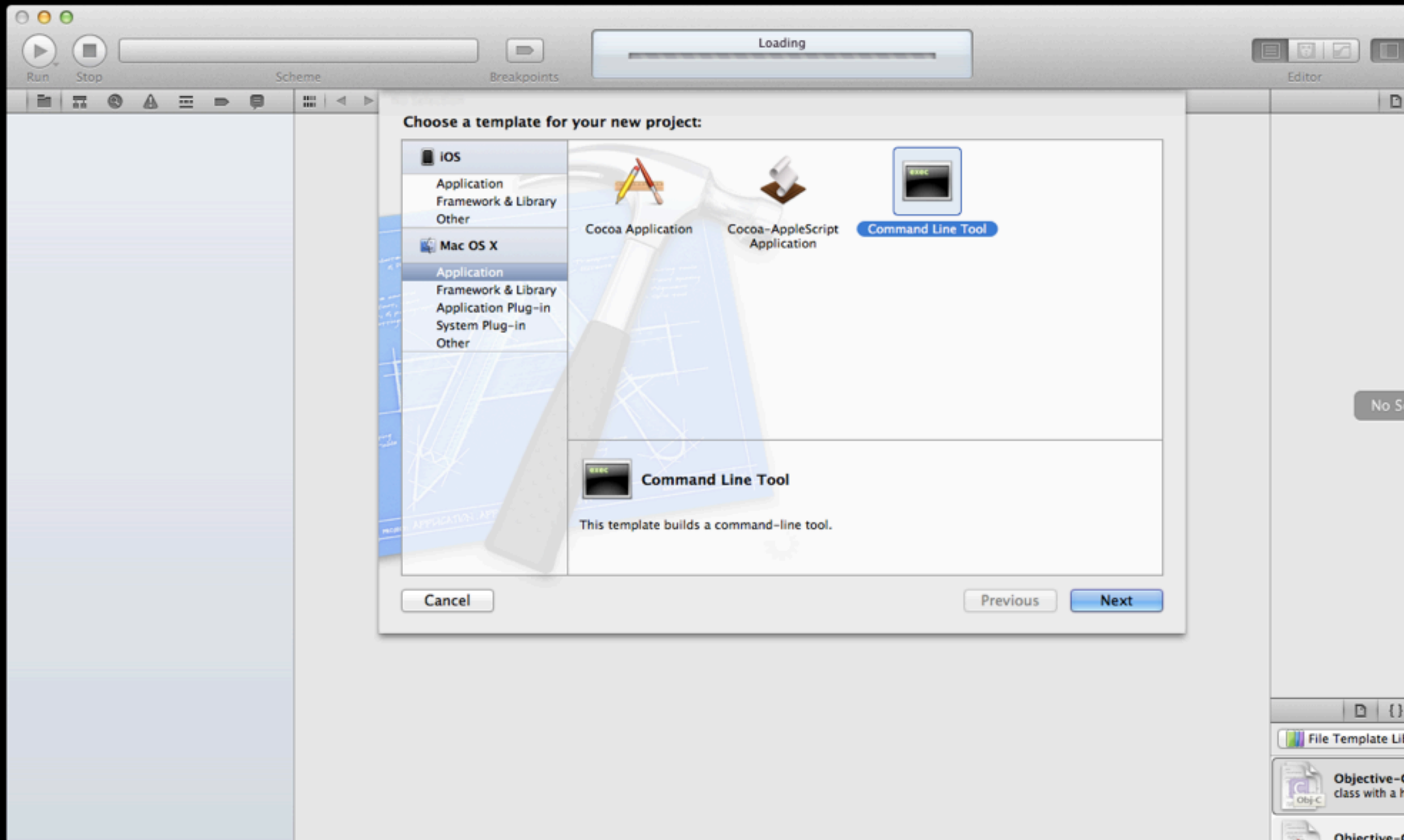
Create a new project

Xcode

Create a new project

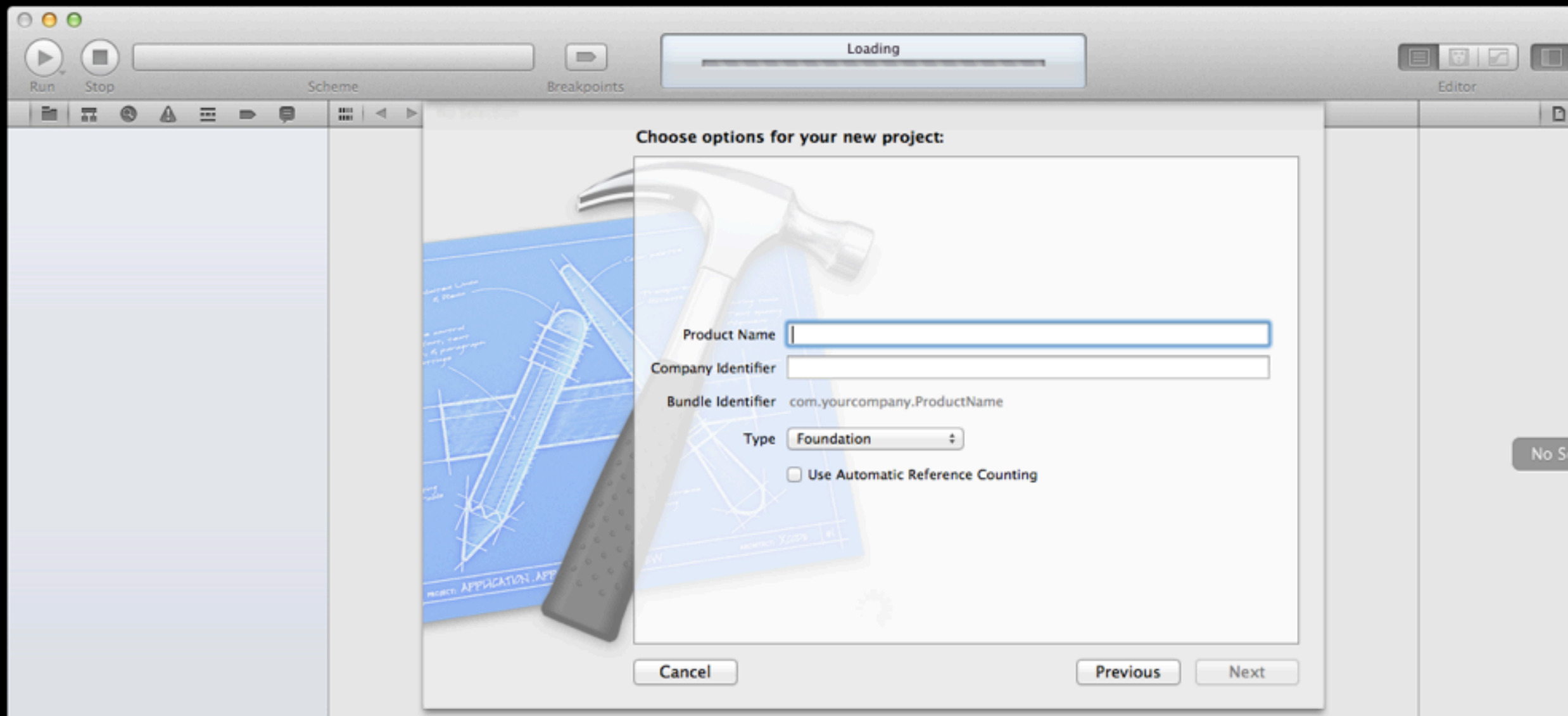
Xcode

New project



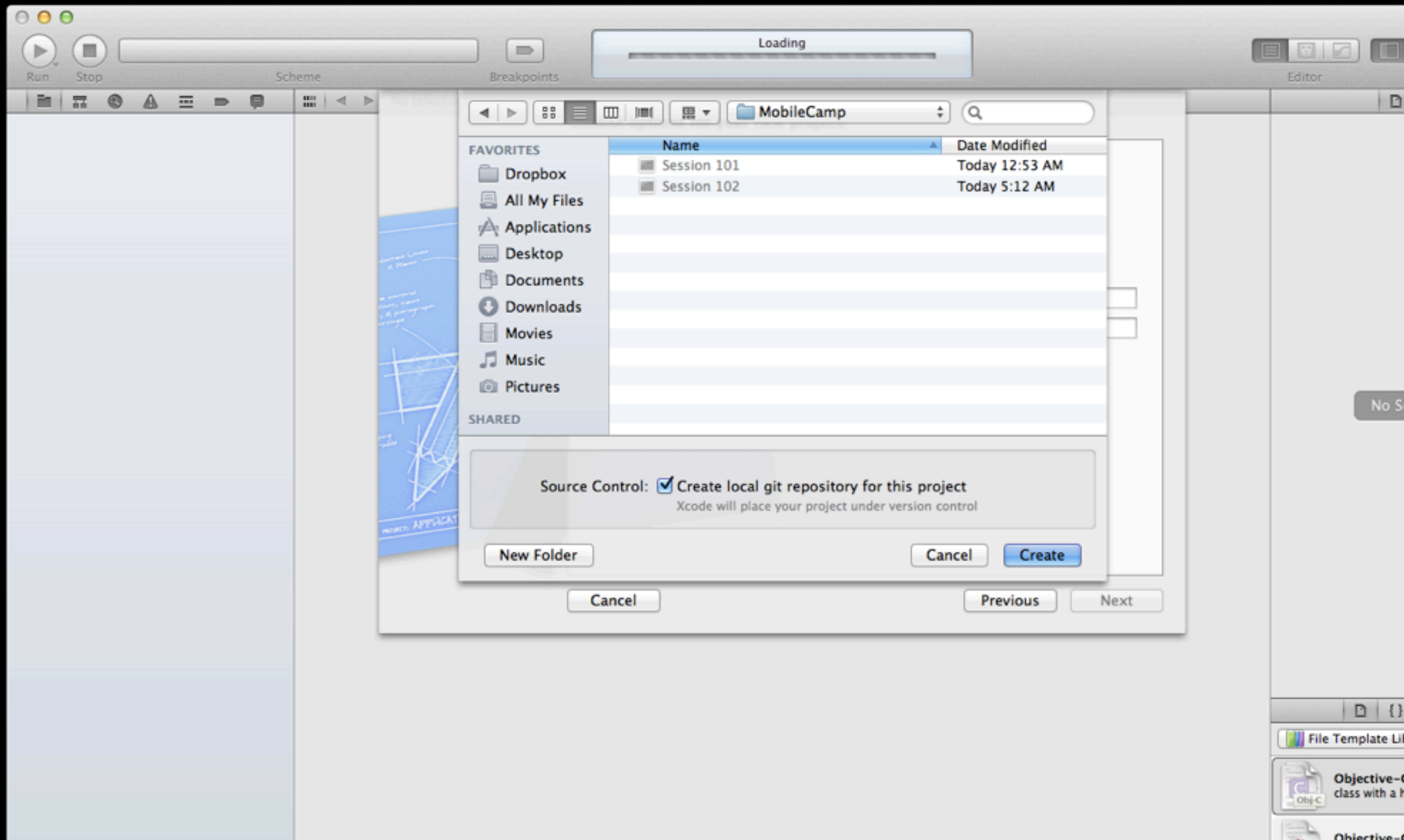
Xcode

New project



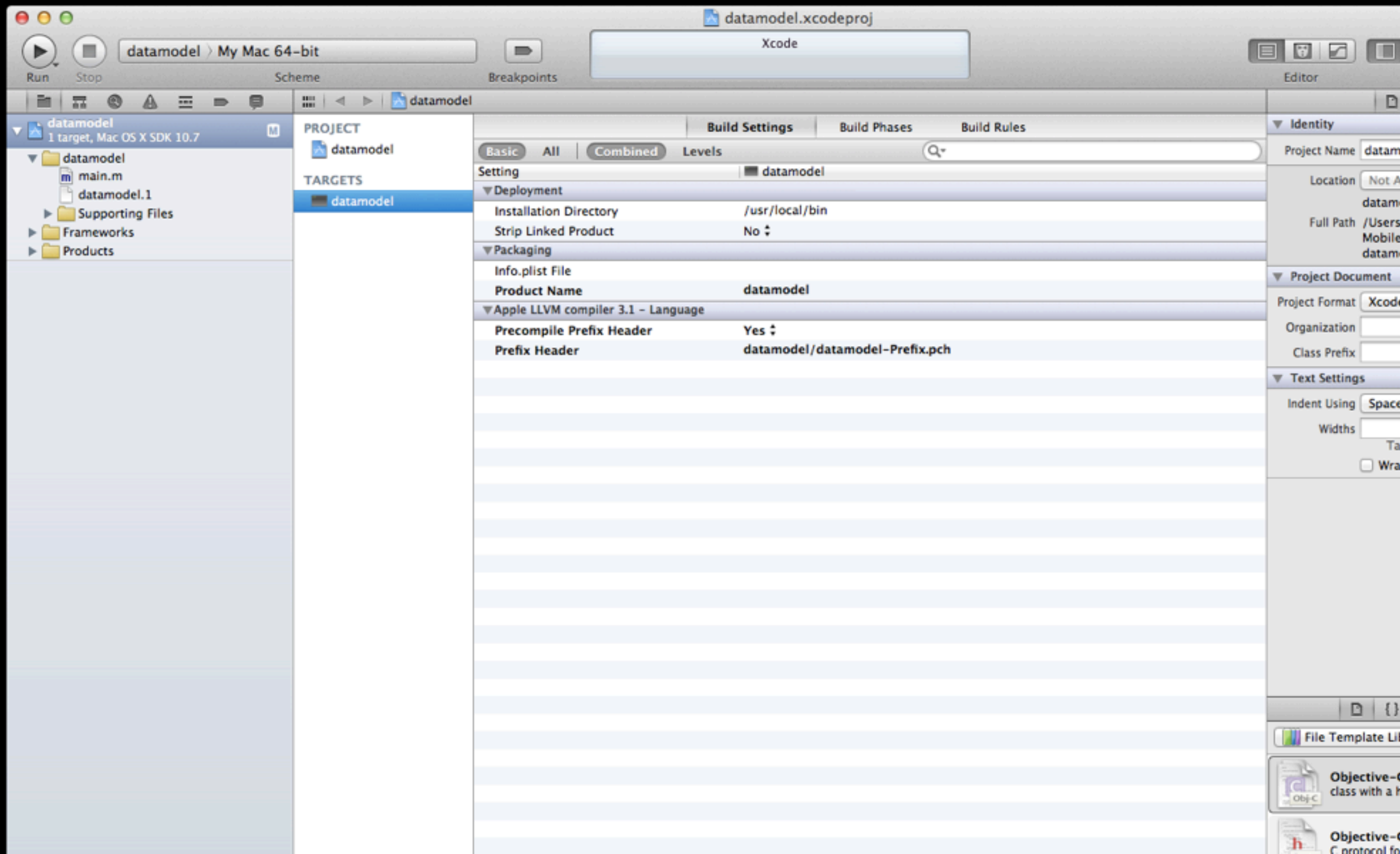
Xcode

New project



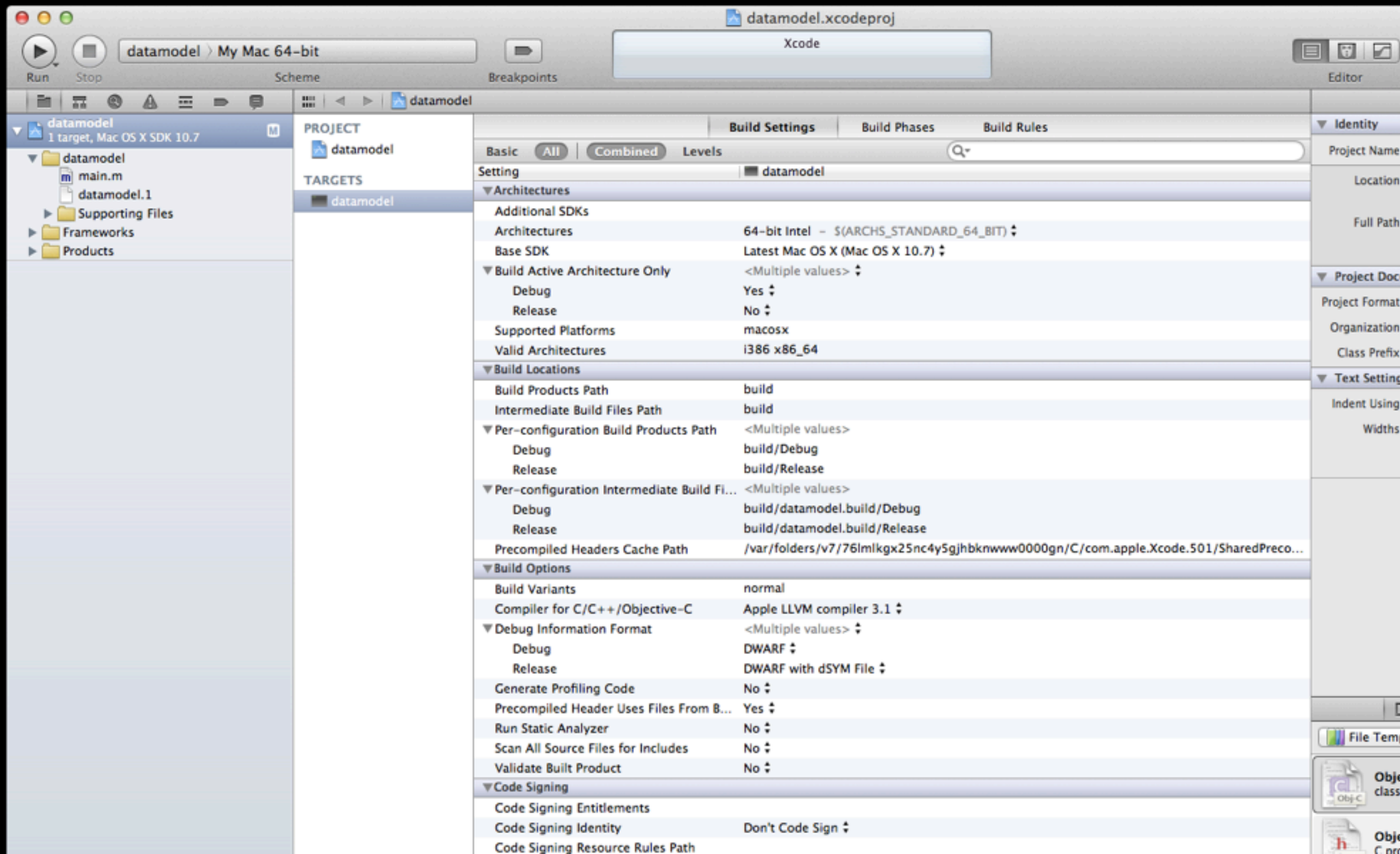
Xcode

Project/Target settings



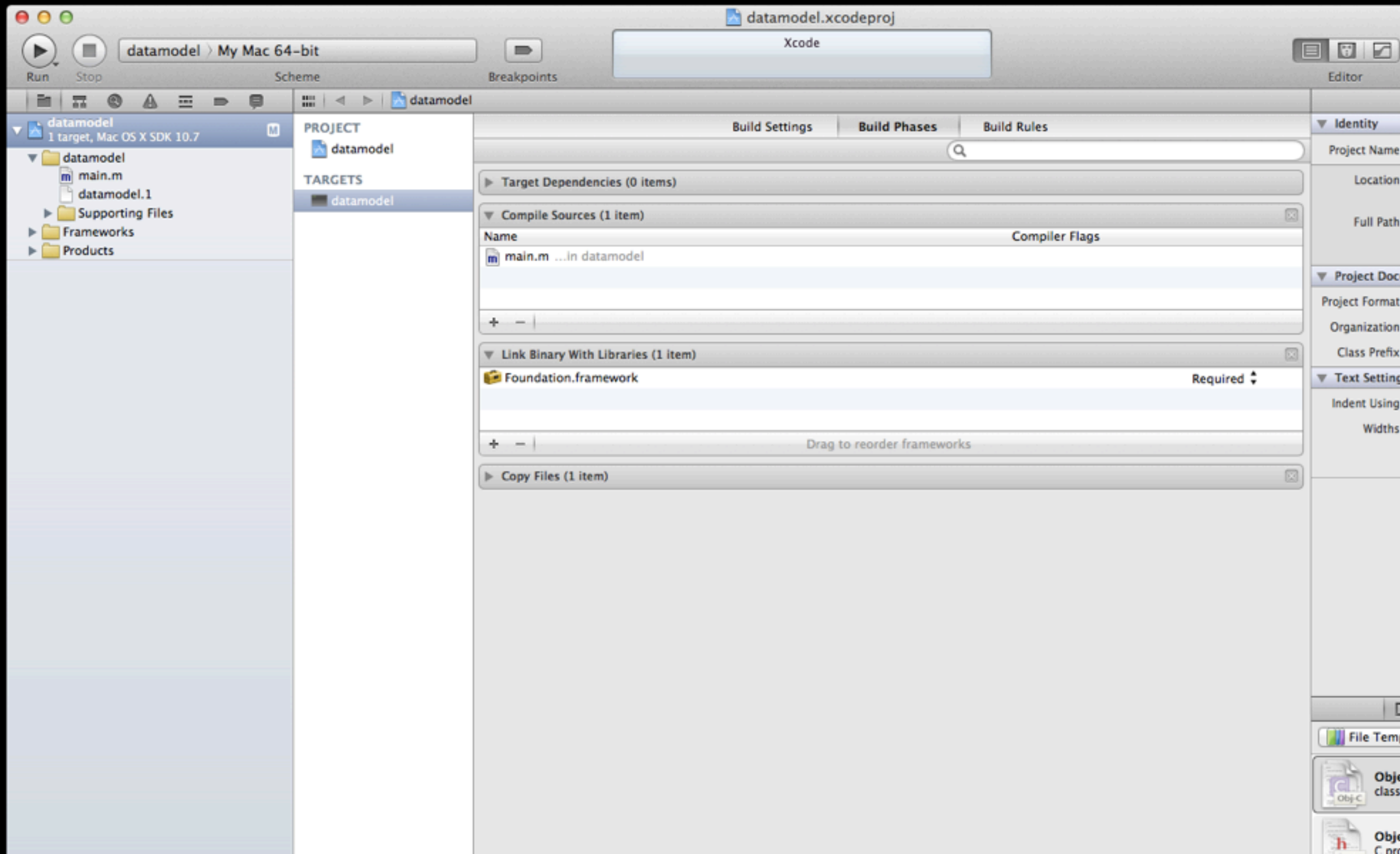
Xcode

Build Settings



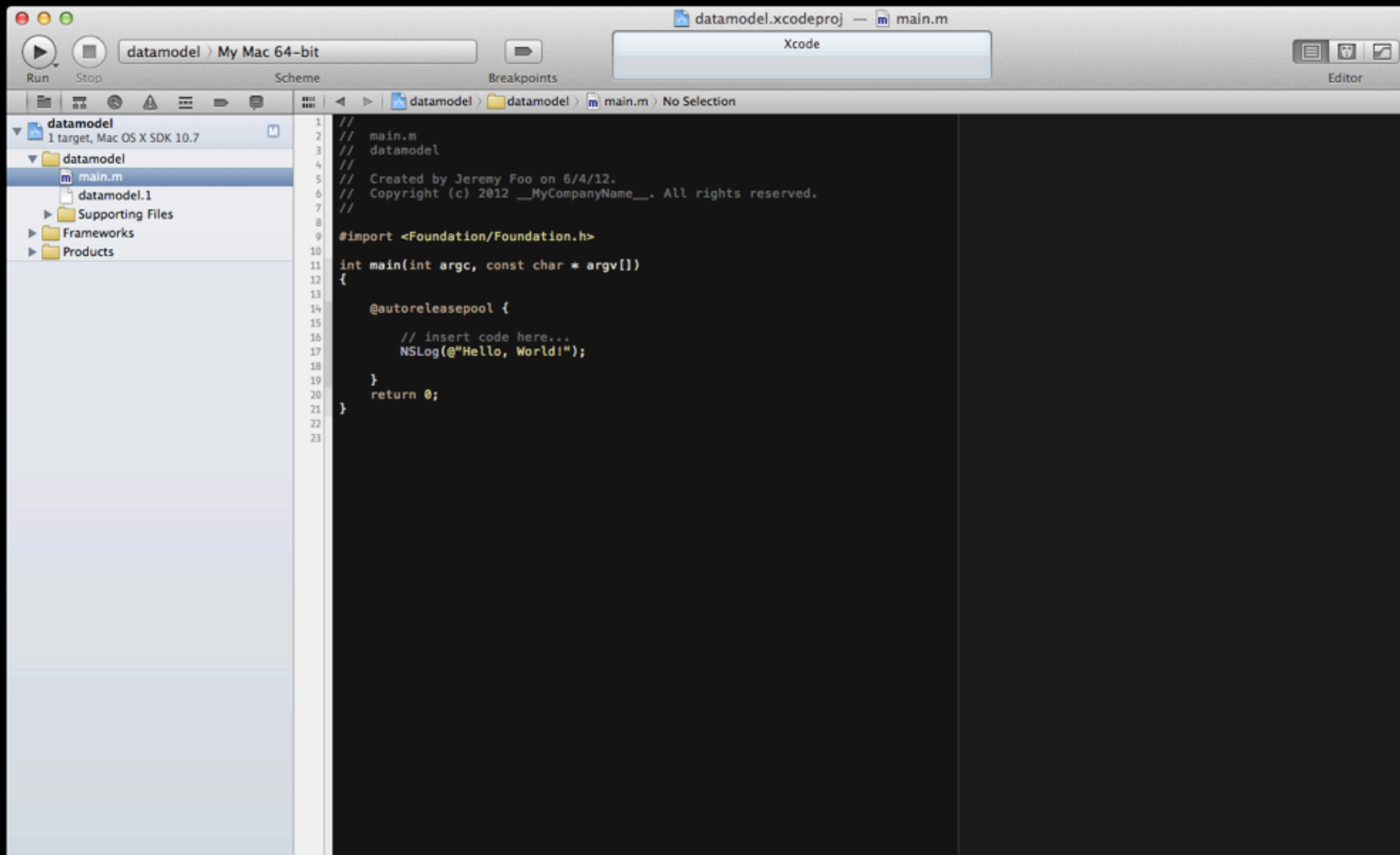
Xcode

Build Phases



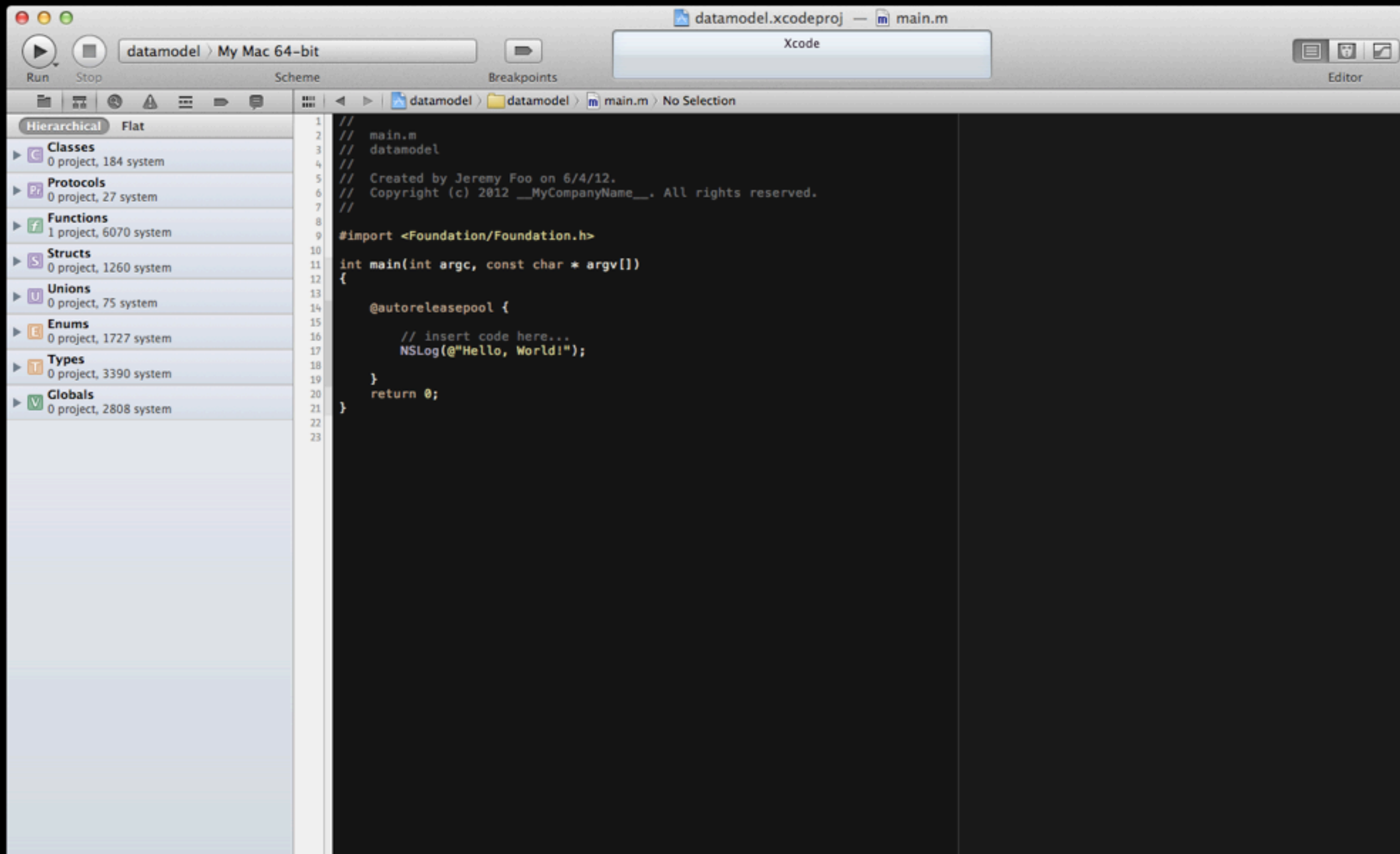
Xcode

Project Navigator



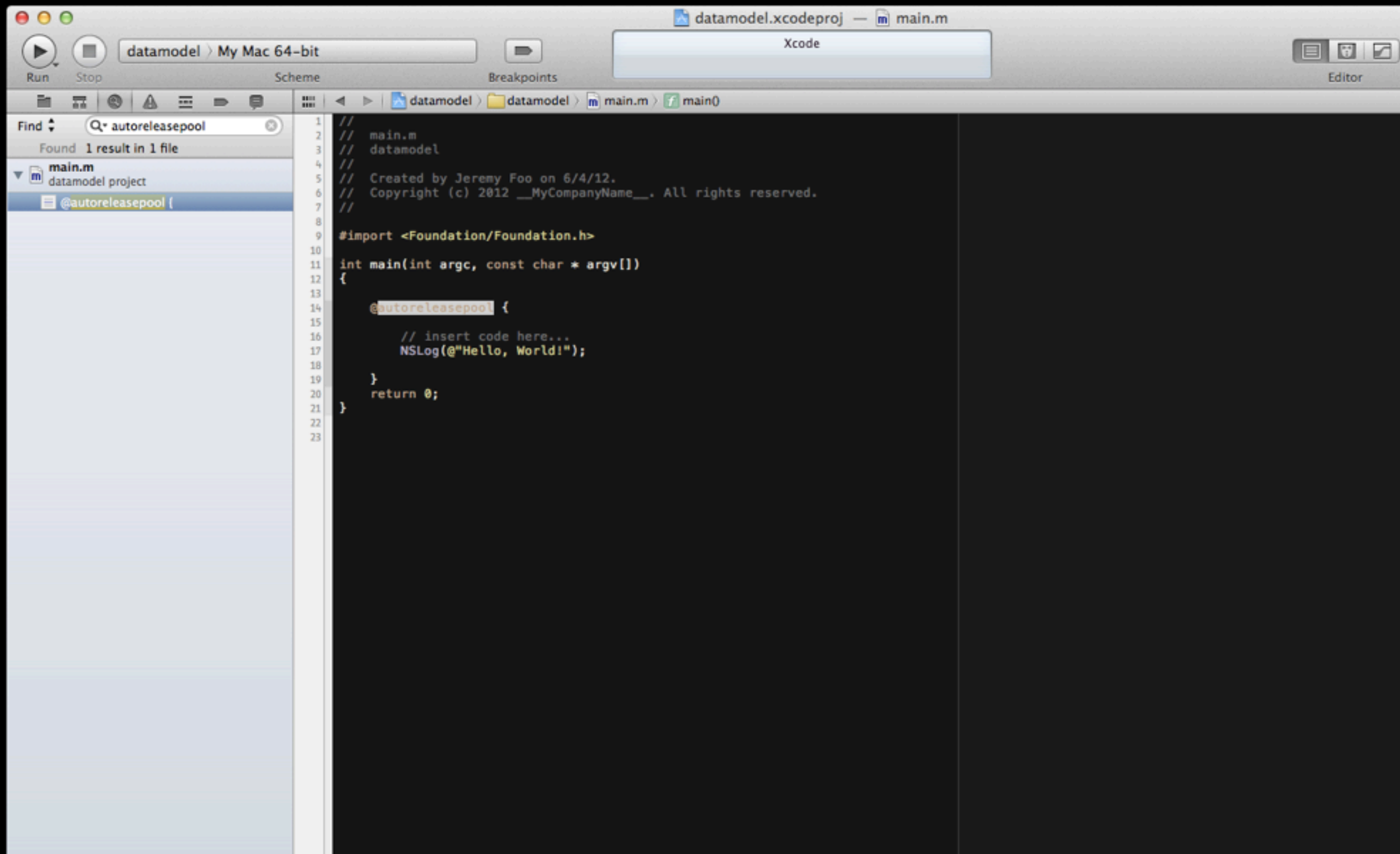
Xcode

Symbol Navigator



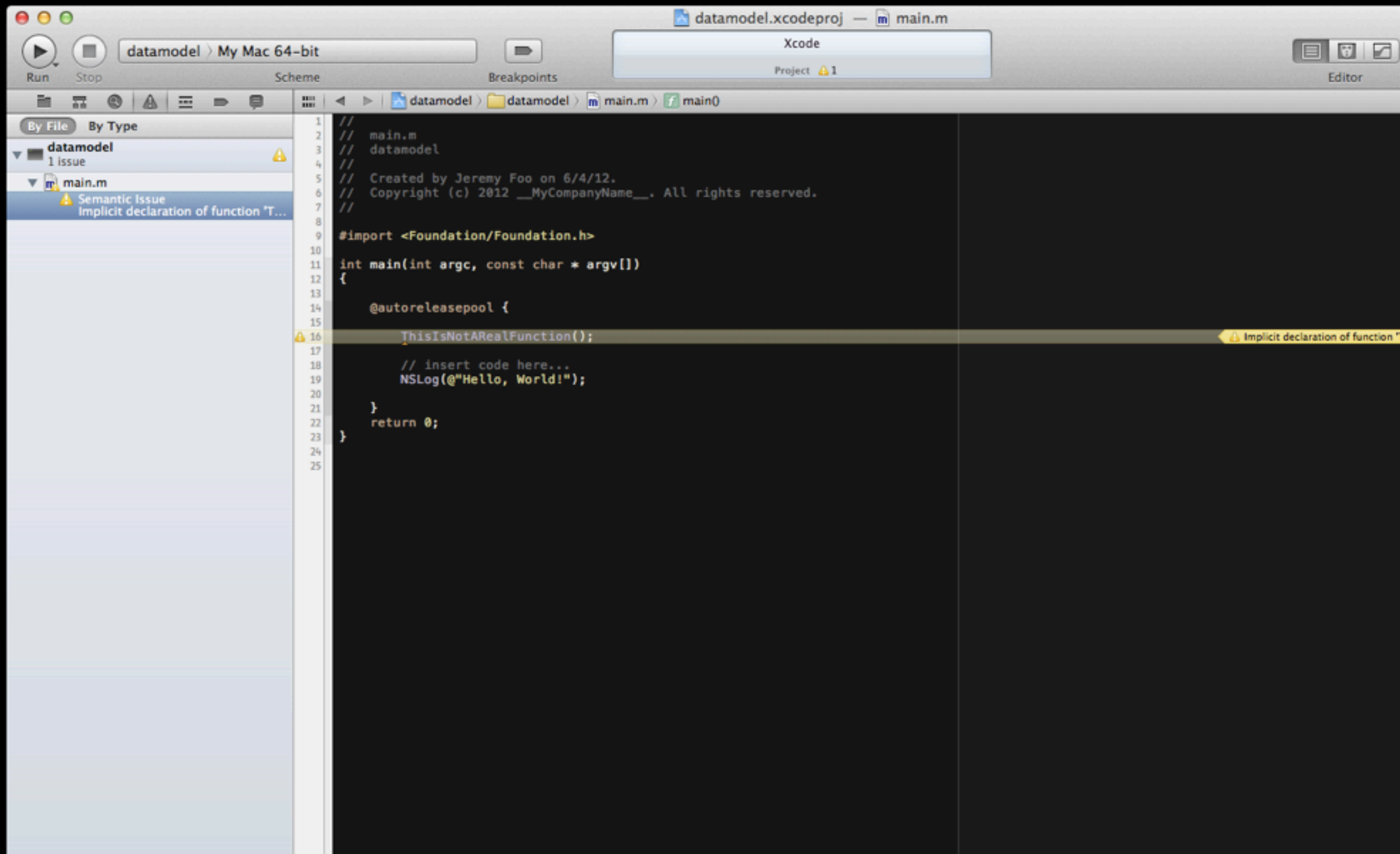
Xcode

Search Navigator



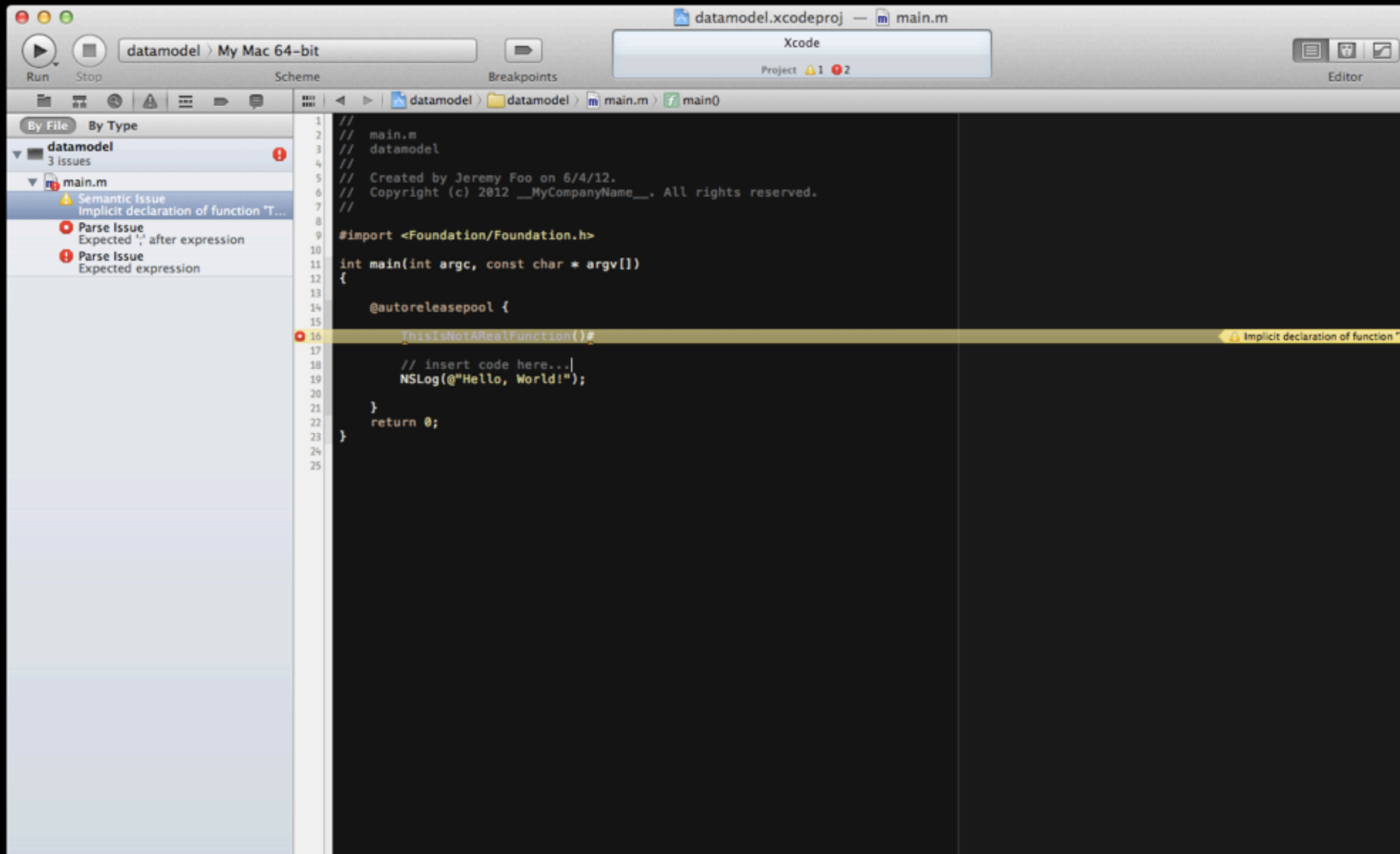
Xcode

Issues Navigator



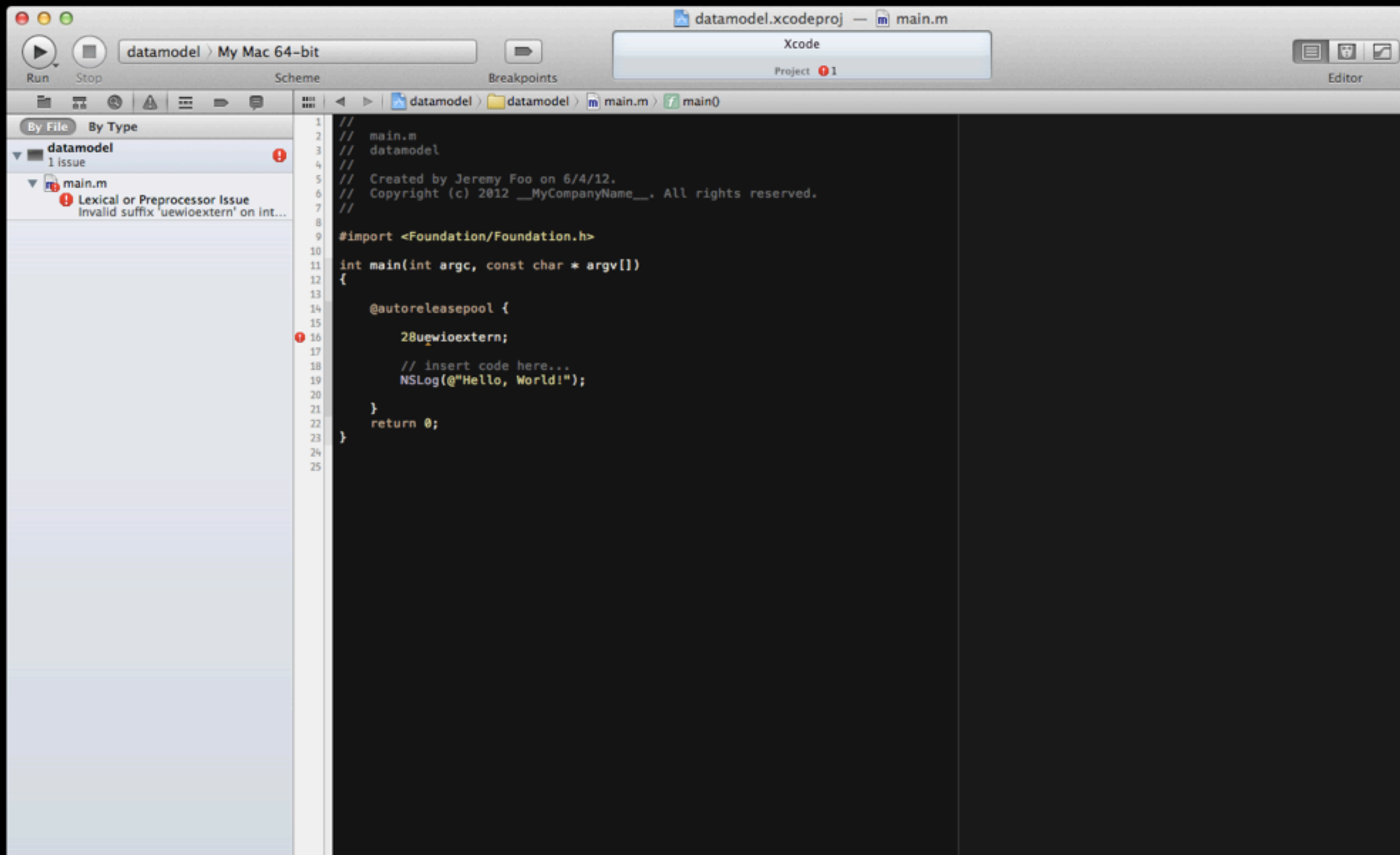
Xcode

Semantic Issues



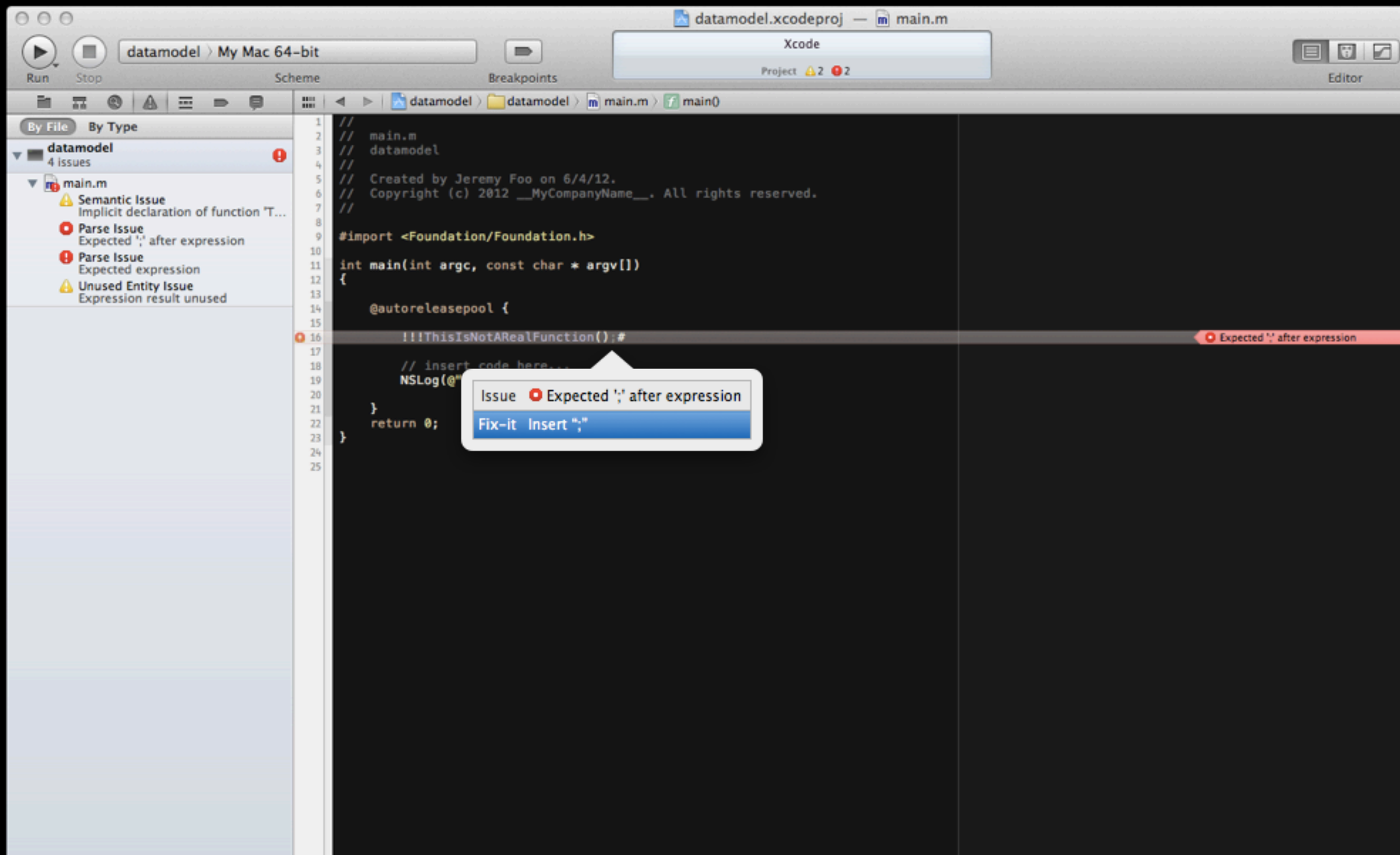
Xcode

Lexical Issues



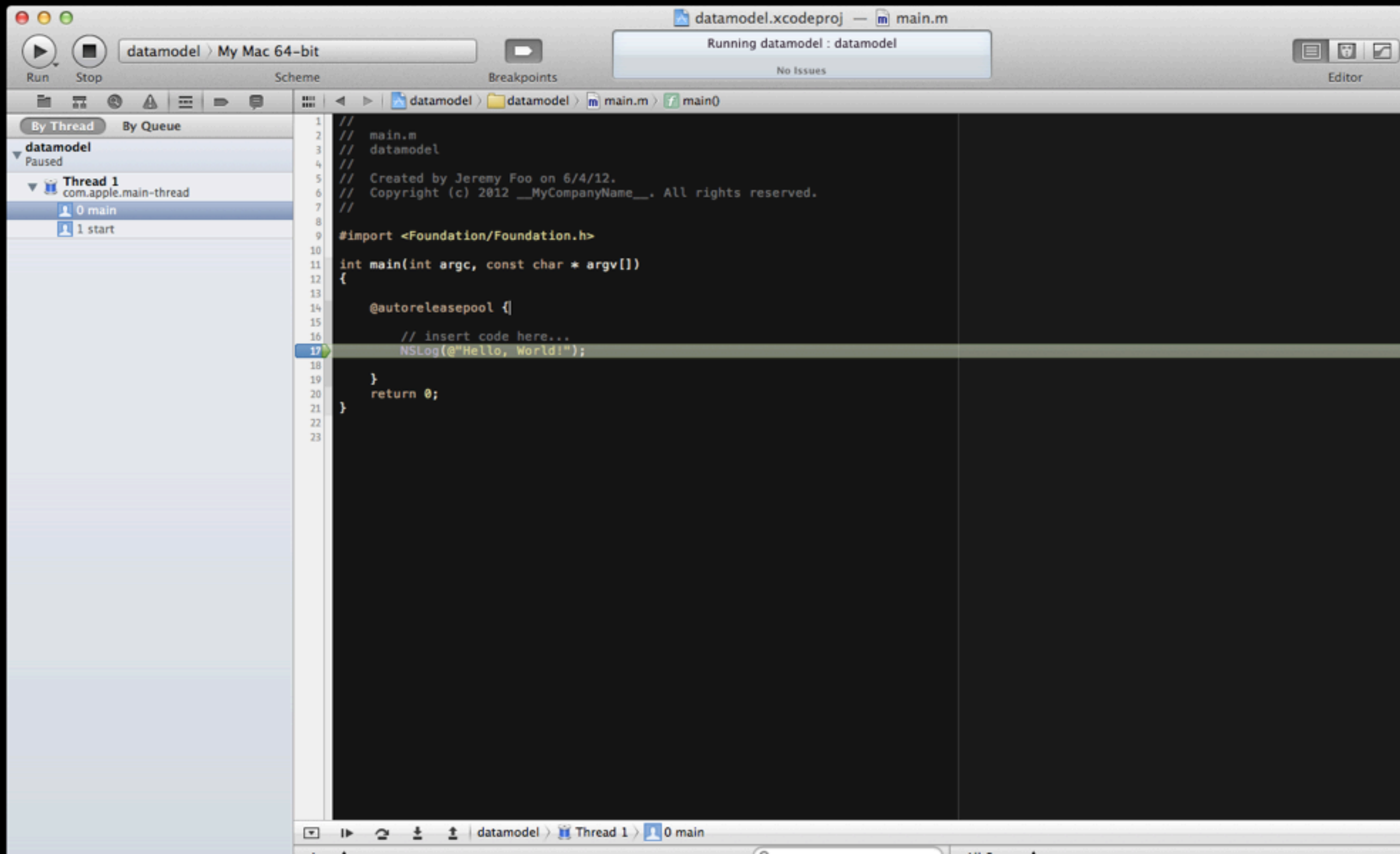
Xcode

Fixit


datamodel.xcodeproj — main.m
Run
Stop
datamodel 〉 My Mac 64-bit
Scheme
Breakpoints
Xcode
Project 2 2
Editor
datamodel 〉 datamodel 〉 main.m 〉 main()
By File
By Type
datamodel
4 issues
main.m
Semantic Issue
Implicit declaration of function 'T...
Parse Issue
Expected ';' after expression
Parse Issue
Expected expression
Unused Entity Issue
Expression result unused
//
// main.m
// datamodel
//
// Created by Jeremy Foo on 6/4/12.
// Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//
#import <Foundation/Foundation.h>
int main(int argc, const char * argv[])
{
@autoreleasepool {
!!!ThisIsNotARealFunction();#
// insert code here...
NSLog(@"
}
return 0;
}
Expected ';' after expression
Issue Expected ';' after expression
Fix-it Insert ";"

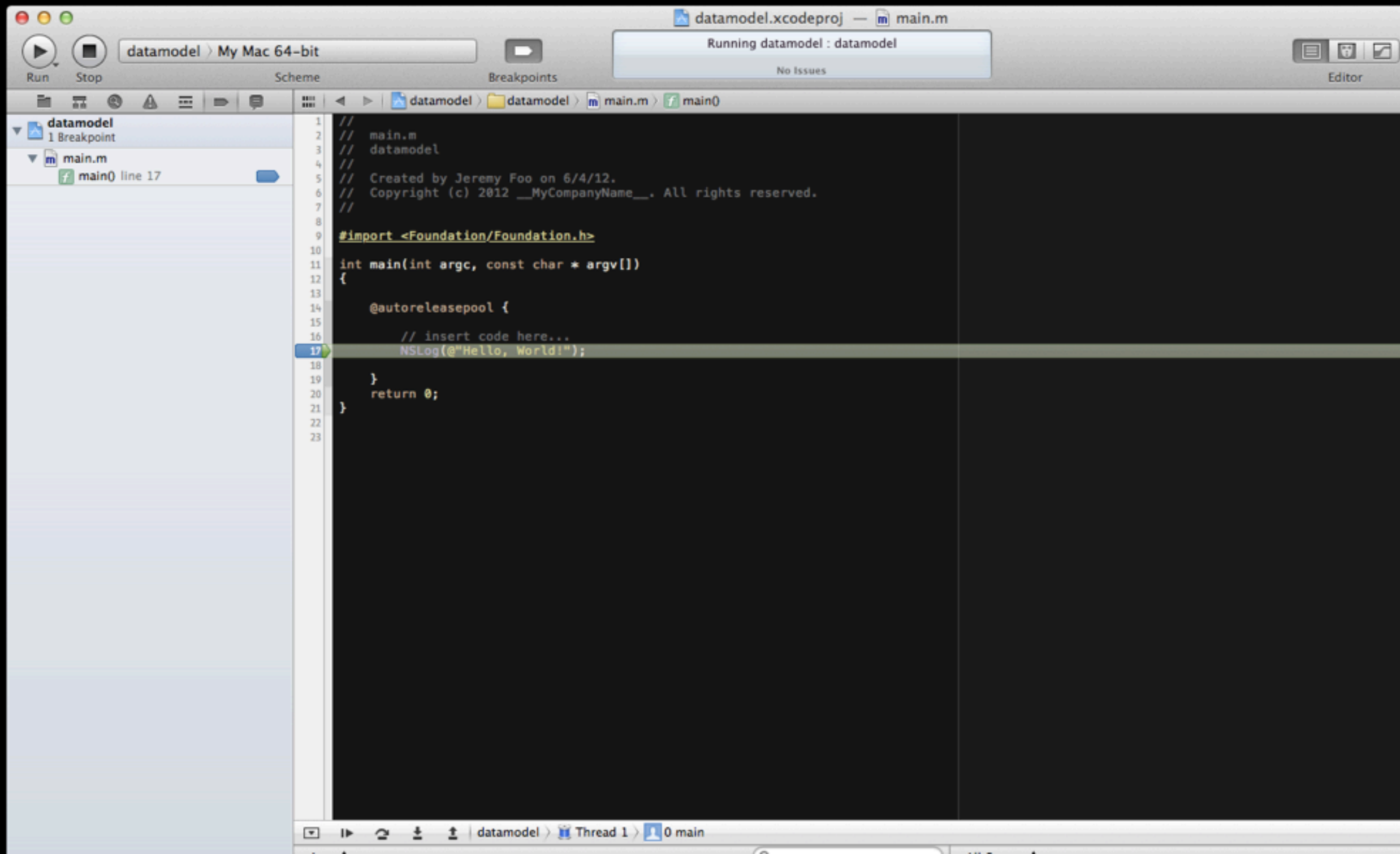
Xcode

Debug Navigator



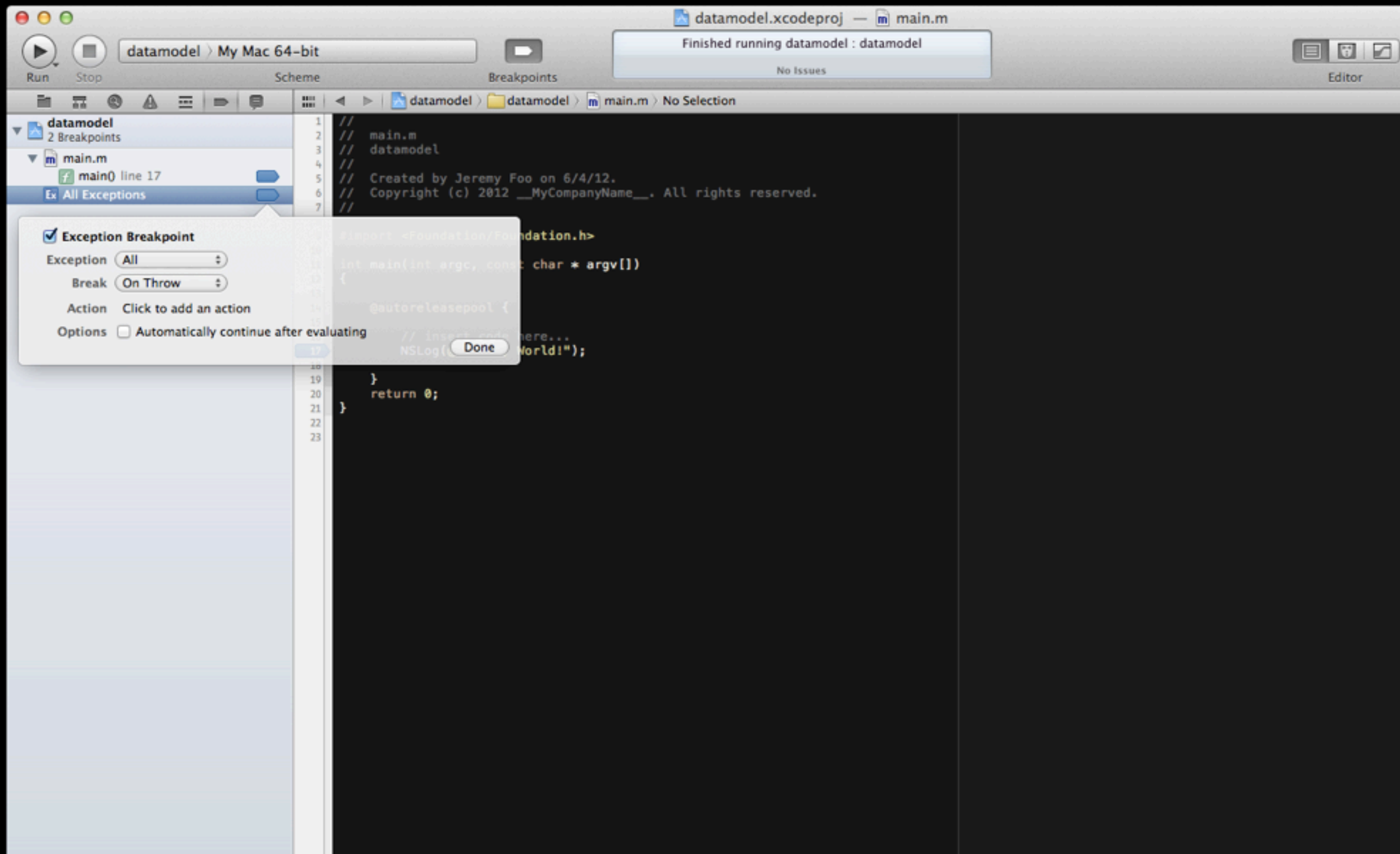
Xcode

Breakpoints



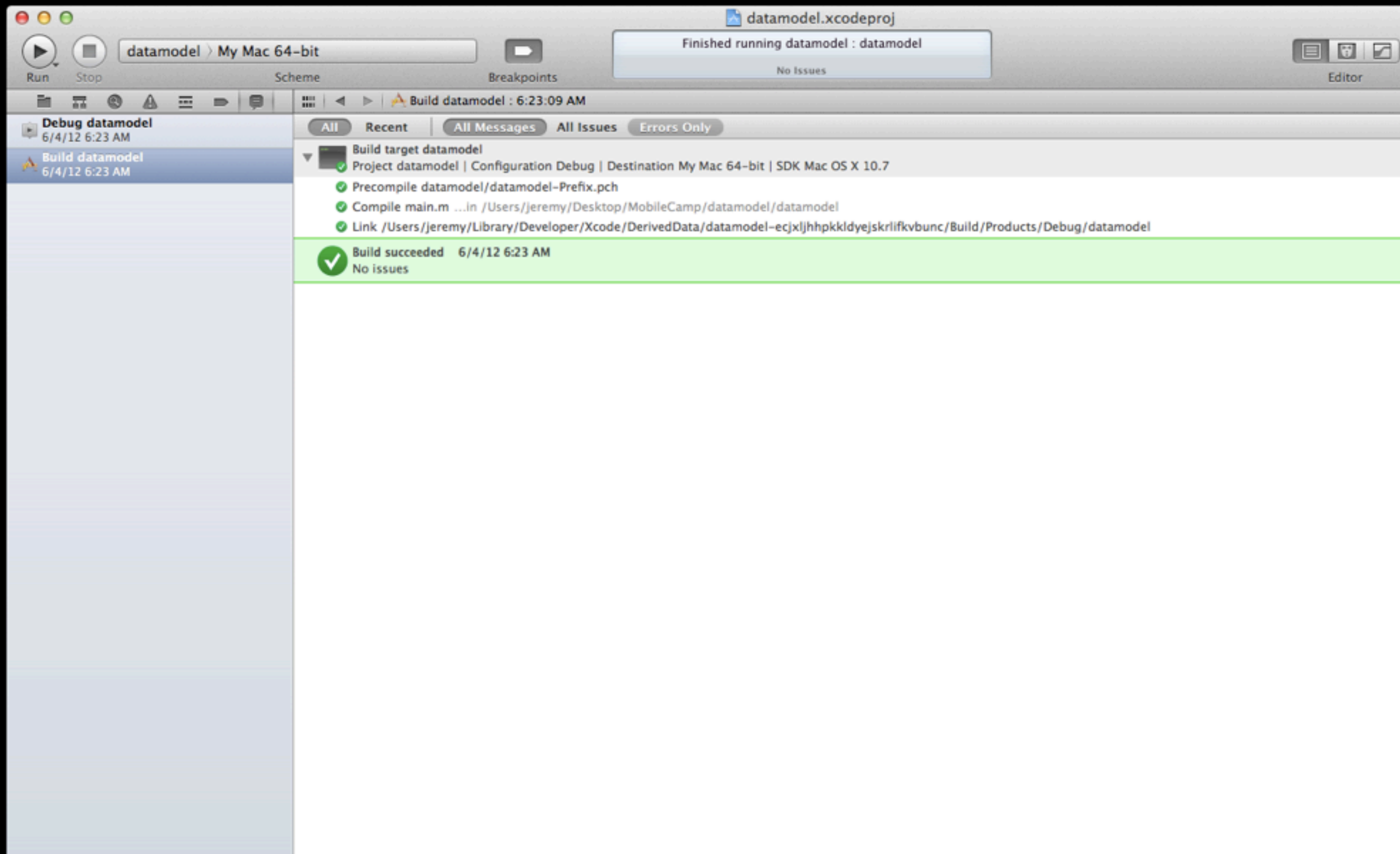
Xcode

Breakpoints



Xcode

Log Navigator



Xcode

Window Shortcuts

New Project Window → ⌘ + ⇧ + 1

Organizer → ⌘ + ⇧ + 2

Xcode

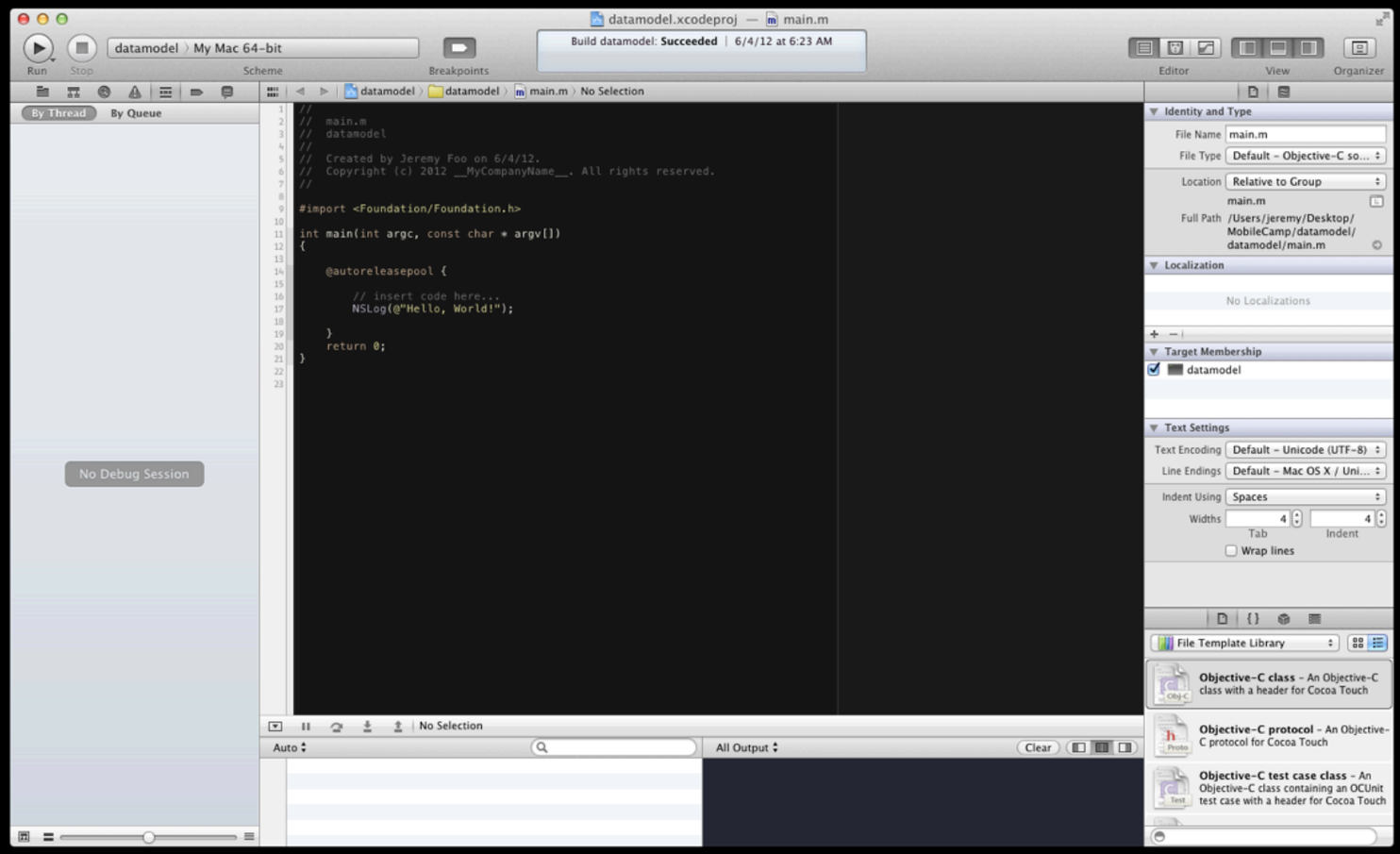
UI Shortcuts

Switch Navigators → ⌘ + (1...7)

Show/Hide Navigators → ⌘ + 0

Show/Hide Utilities → ⌘ + ⌥ + 0

Show Console → ⌘ + ⇧ + Y



Xcode

Compilation Shortcuts

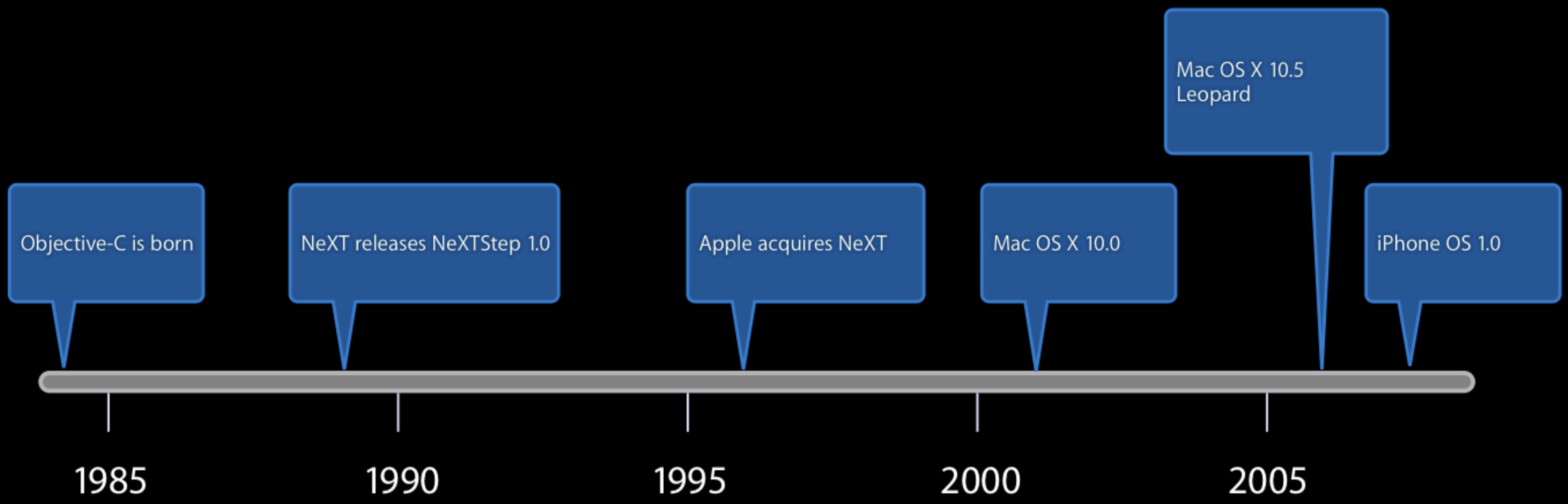
Build → ⌘ + B

Build and Run → ⌘ + R

Analyze → ⌘ + ⇧ + B

Objective-C

Introduction and Basics


Objective-C is born
NeXT releases NeXTStep 1.0
Apple acquires NeXT
Mac OS X 10.0
Mac OS X 10.5
Leopard
iPhone OS 1.0
1985
1990
1995
2000
2005

Objective-C

C

* Objective-C++

Object Orientated C

Dynamic

Runtime

Objective-C

Organization

Prototype + Implementation

Prototype

Implementation

Objective-C

Naming conventions

- No namespace support
- Prefix class names with “namespace”
    - Apple uses “NS” for Foundation classes
    - Stands for NextStep
- Best practice: choose prefixes > 2 characters
- useCamelCase!

Objective-C

Further naming conventions

- Getters are just the name of the property
- Setters are basically `-set<Property Name>`
- Method names provide a summary of
  - Return type
  - Expected parameters

```
- (NSURL *)URLByAppendingPathComponent:(NSString *)pathComponent
```

Return Type

Expected Parameters

Objective-C

@interface

- Defines class structure
- 3 types
  - Class declaration

```
@interface <# class name #> : <# super class #> {
    // instance variables here
}

// method and properties here

@end
```

Objective-C

@interface

- Defines class structure
- 3 types
    - Class declaration
    - Class extension

```
@interface <# class name #> ()

@end
```

Objective-C

@interface

- Defines class structure
- 3 types
    - Class declaration
    - Class extension
    - Category

```
@interface <# class name #> (<# category name #>)

@end
```

Objective-C

@property/@synthesized

- Apple's effort to reduce written code
- Represents a property of an object
- Companion to `@synthesized` keyword
- Declared within class declaration block
- Accessible by dot notation
    - object.property

Objective-C

@property/@synthesized

- **Special attributes for the property**
  - `non-atomic/atomic`
  - `retain/copy`
  - `assign`
  - `readonly/readwrite`
  - `getter=/setter=`

```
@property (nonatomic, retain) NSString *myString;
```

Objective-C

@property/@synthesized

- Synthesize properties in implementation

```
@implementation DCListItem
@synthesize myString = _myString;
@end
```

Objective-C

@property/@synthesized

- Synthesize properties in implementation

```
@implementation DCListItem
@synthesize myString;
@end
```

Objective-C

Key-Value Coding

- All properties have a key
- They can be accessed via `–valueForKey`
- And set via `–setValueForKey`
- Better Reflection than most

Objective-C

@protocol

- Interface specifications that classes conform to
- Keywords to control conformance
  - `@required`
  - `@optional`
- Used extensively for the delegate pattern
- Can conform to another protocol at the same time

```
@protocol aProtocol <NSObject>

@end
```

Objective-C

@implementation

- **Implementation of** `@interface`
- **Block for**
  - `@synthesized`
  - **Selectors**

```
+(void)thisIsAStaticMethod {

}

-(void)thisIsAnInstanceMethod:(id)firstVar secondVar:(id)secondVar {

}
```

Objective-C

Variable Types

- Supports all C scalar types
- Objective-C native object types
  - `NSString`
  - `NSArray`
  - `NSNumber`
  - `NSDictionary`
  - `NSSet`
  - `...`

Objective-C

Native object types

- Everything inherits from `NSObject`
- Relatively flat object hierarchy
- Lots of design pattern use to reduce complexity

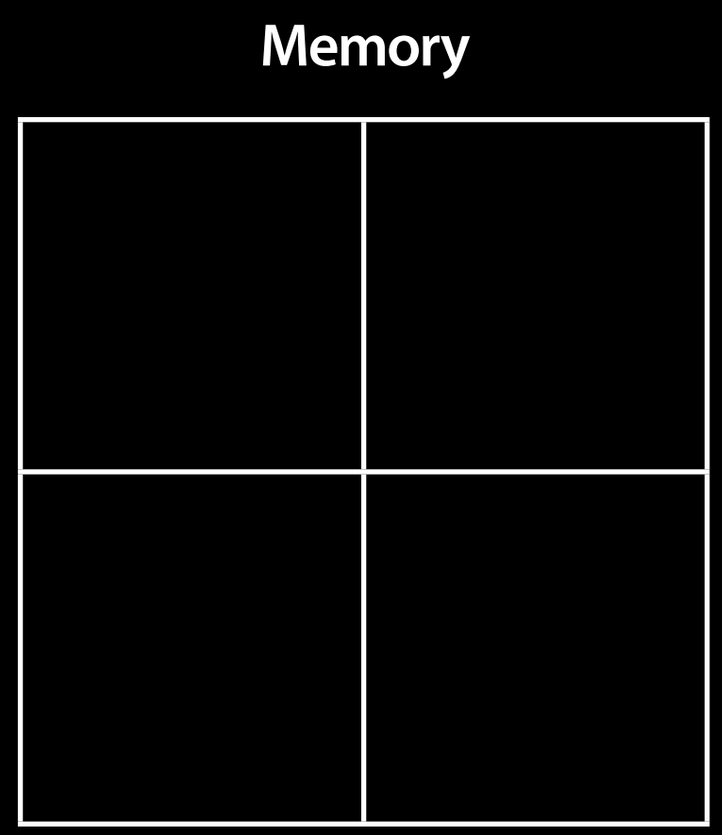
Objective-C

Native object types

- Struct in memory with `isa` pointer
- `isa` points to an object's class
- All object references are pointers to the actual object

`myString`

`myString`

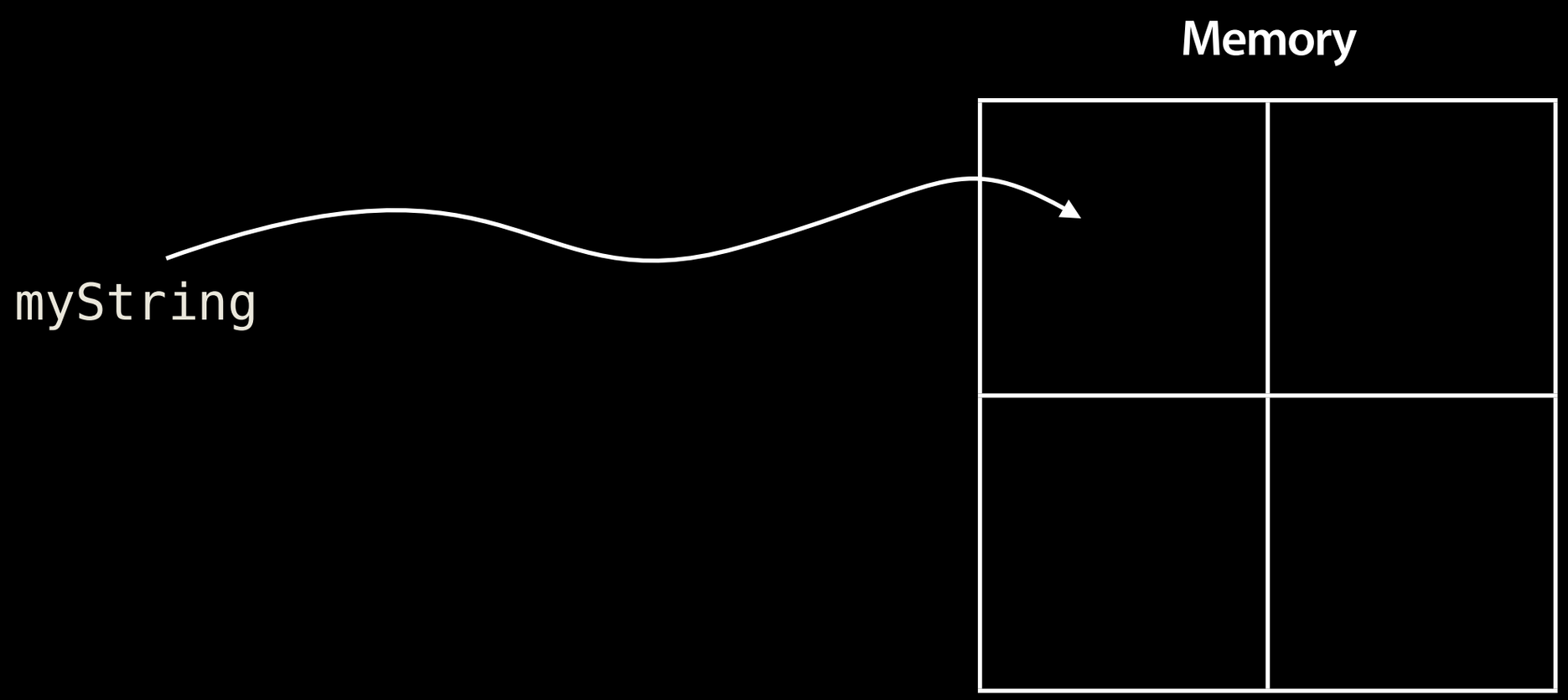


Objective-C

Native object types

- Struct in memory with `isa` pointer
- `isa` points to an object's class
- All object references are pointers to the actual object

```
NSString *myString
```

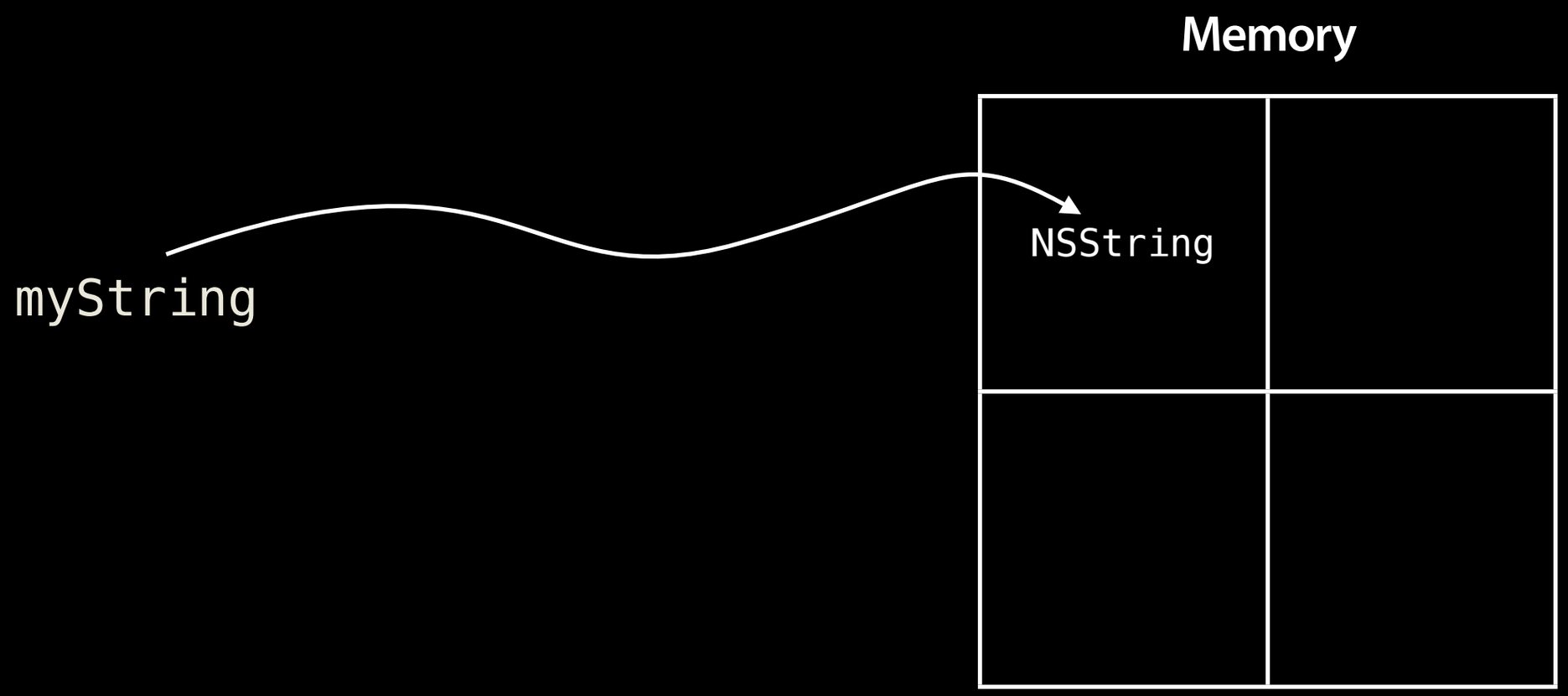


Objective-C

Native object types

- Struct in memory with `isa` pointer
- `isa` points to an object's class
- All object references are pointers to the actual object

```
NSString *myString = [[NSString alloc] init];
```



Objective-C

Messaging

- Dynamically typed language
- Concept of messaging
  - Accept/Forward
- Uses SEL to represent message

```
SEL mySelector = @selector(uppercaseString);

[myString performSelector:mySelector];

[myString performSelector:@selector(uppercaseString)];
```

Objective-C

Messaging

- Sending a message uses the [ ] braces
- A message can have multiple parameters

```
[myObject firstParameter:YES secondParameter:YES];
 |_____|  |_____________________________________|
Receiver   Message with parameters (@selector)
```

- Receiver can also be a class
  - Static methods (+)
  - Classes are also objects

Objective-C

Messaging

- Not function calls!
- Resolved at runtime
- No errors on compile if object doesn't respond to selector
- Will assert exception during runtime

Memory Management

ARC, Retain, Release

Memory Management

Overview

- No garbage collection
- Manual
- Reference counting
- Automatic Reference Counting (>= iOS 5.0)
- 3 easy steps to remember

Memory Management

Object creation

- Object creation returns an object with a ref count of +1
- Copying also returns a ref count of +1

```
NSString *test = [[NSString alloc] init];
NSString *test2 = [test copy];
```

Memory Management

Retain/Release

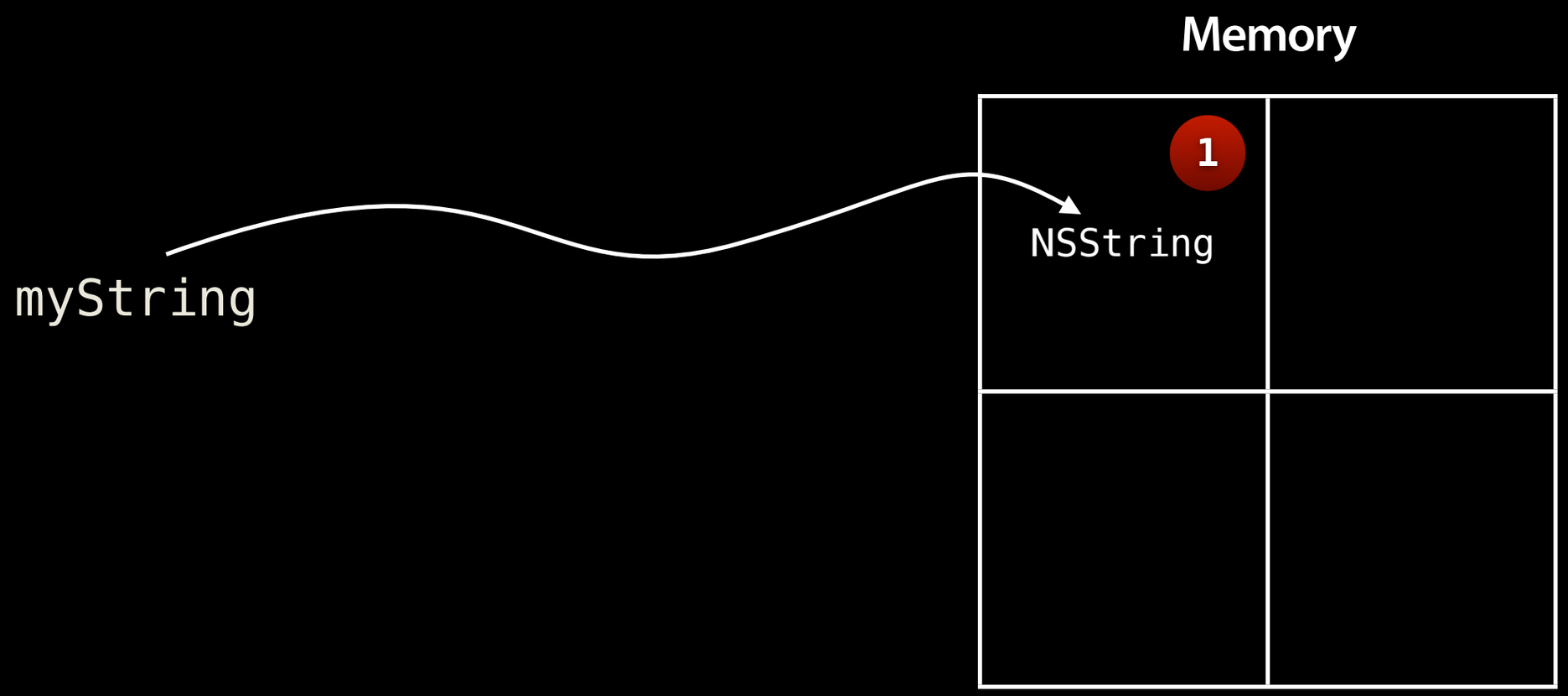
- Retain when you want to use a resource
- Release when you are done with it
- Autorelease returned objects
- Static selectors return autoreleased objects
- Cleanup can be done in `-dealloc`

```
[myString retain];
[myString release];
[myString autorelease];
```

Memory Management

Retain/Release

```
NSString *myString = [[NSString alloc] init];
```

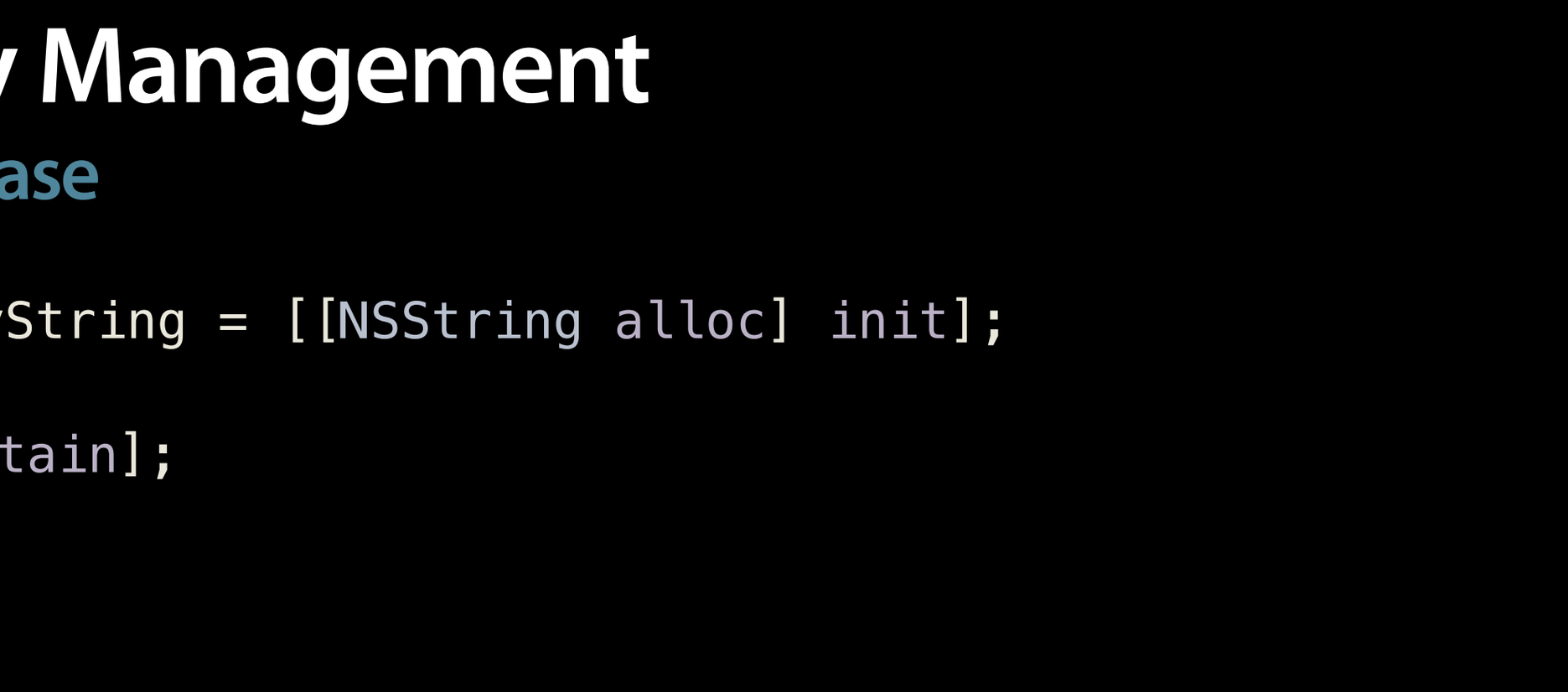


Memory Management

Retain/Release

```
NSString *myString = [[NSString alloc] init];

[myString retain];
```

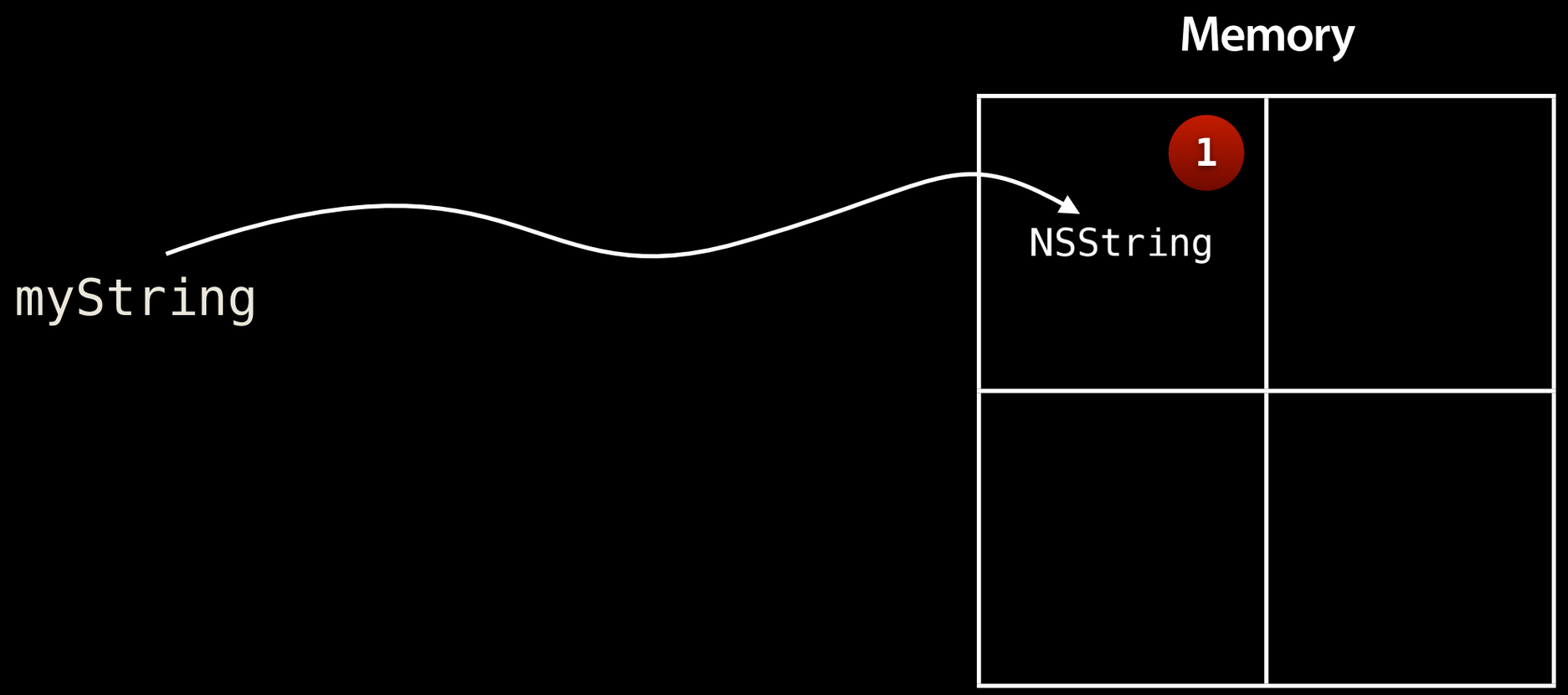


Memory Management

Retain/Release

```
NSString *myString = [[NSString alloc] init];

[myString retain];
[myString release];
```

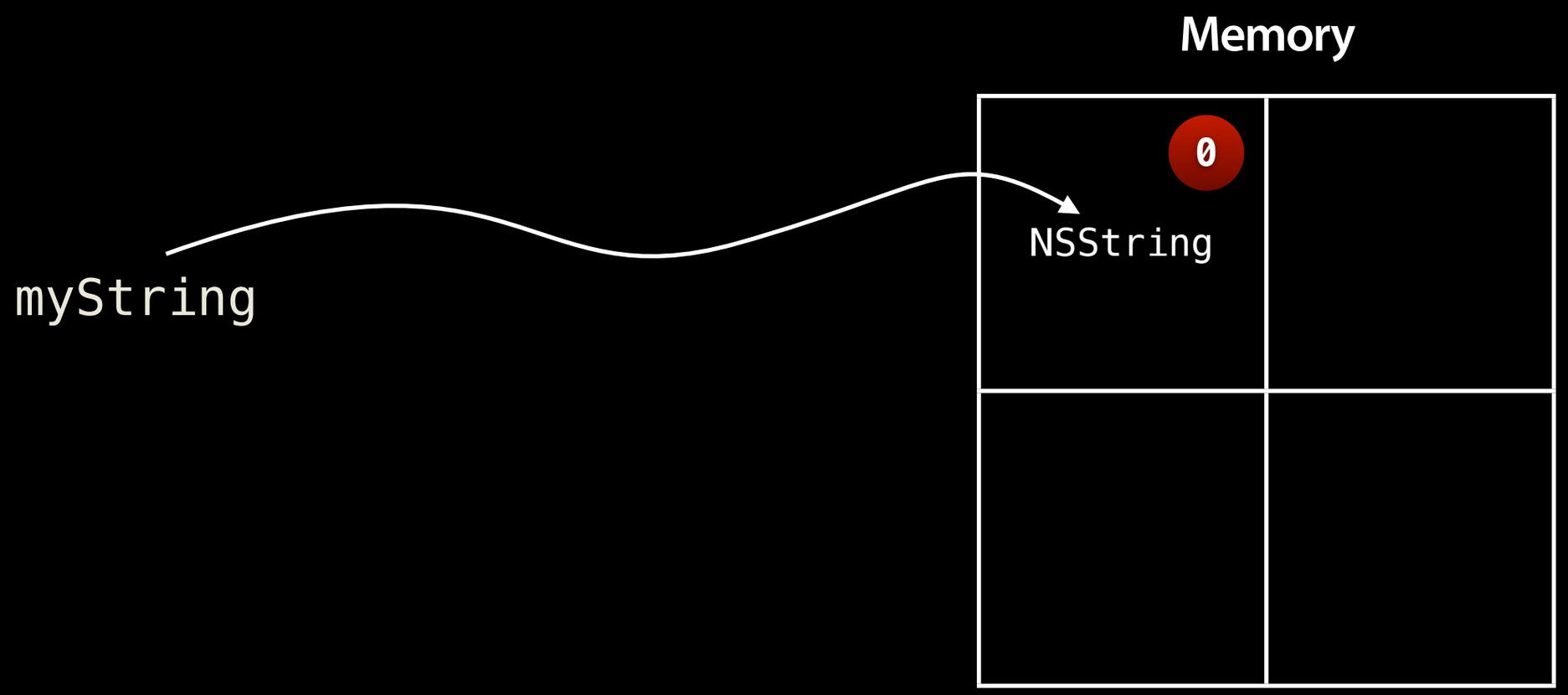


Memory Management

Retain/Release

```
NSString *myString = [[NSString alloc] init];

[myString retain];
[myString release];
[myString release];
```

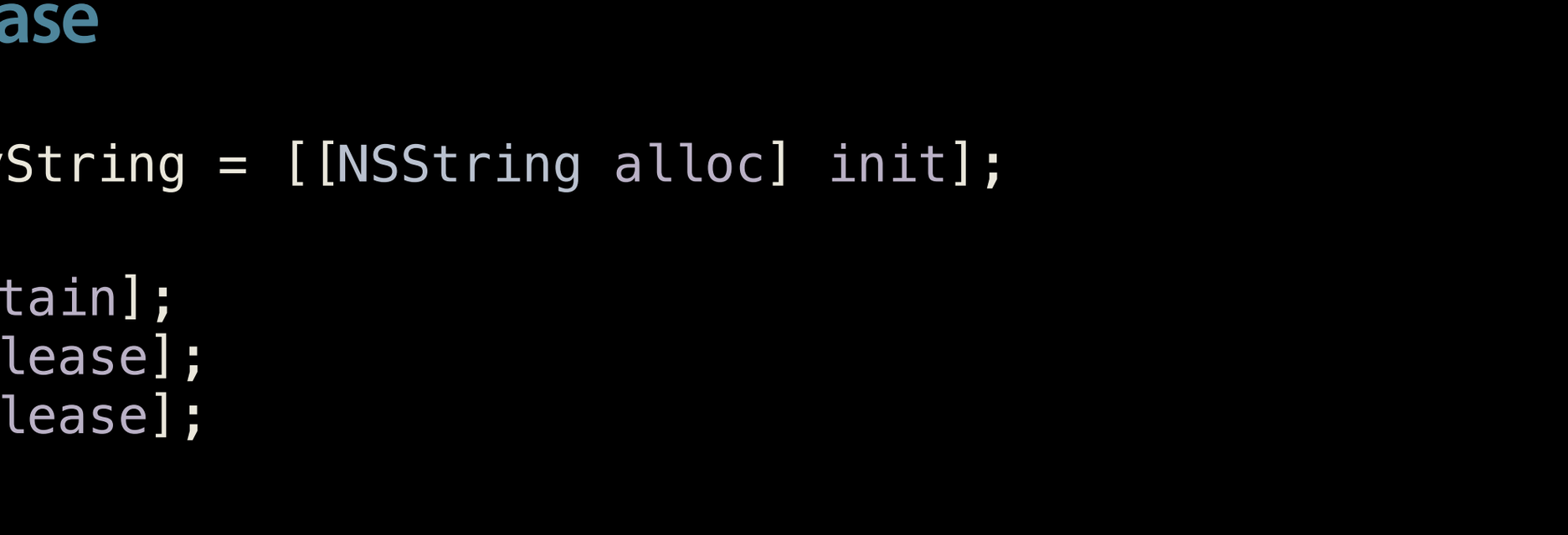


Memory Management

Retain/Release

```
NSString *myString = [[NSString alloc] init];

[myString retain];
[myString release];
[myString release];
```



Memory Management

Autorelease

- Stack based object pool
- Possible to nest pools
- When `–drain` is called, reduces all objects by 1 ref count
- Objects added to pool by calling `–autorelease`

```
@autoreleasepool {
    @autoreleasepool {

    }
}
```

Memory Management

Object Properties

- `@property (retain)`
- `@property (copy)`
- `@property (assign)`

Memory Management

Automatic Reference Counting

- No more retain/release
- No more dealloc!
- Weak/Strong references
- Requires strict adherence to naming convention

Memory Management

Weak Reference

- Object that gets deallocated will immediately be set to `nil`
- HOMG!
- Mutually exclusive to strong/retain/copy/assign

Memory Management

Rules Recap

1. Retain when you need a resource, release when done
2. Creating and Copying gives a +1 retain count
3. Returned objects are always/should be autoreleased
4. NEVER RELY ON THE RETAIN COUNT EXPLICITLY

Blocks

Closures, Lambdas and *all* that

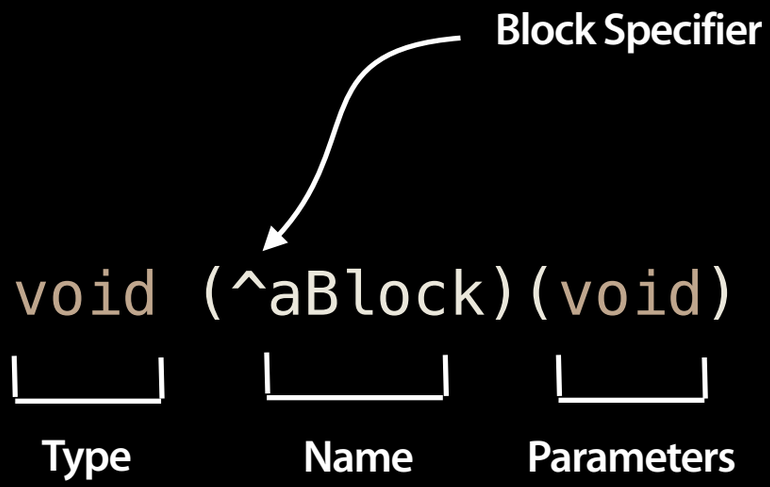
Blocks

What on earth are they?

- Functions as variables
- Specified as a caret or ^
- Has a return value as its type
- Has C styled parameters
- Called closures or lambdas

Blocks

What on earth are they?



Blocks

What on earth are they?

```
void (^aBlock)(void)
int    anInt;
```

Blocks

What on earth are they?

```
void (^aBlock)(void) = ^{
    NSLog(@"This is a block called \"Hello\"");
};
```

Blocks

What on earth are they?

```
void (^aBlock)(void) = ^{
    NSLog(@"This is a block called \"Hello\"");
};

aBlock(); // will print "This is a block called "Hello""
```

Blocks

Return a value and accepts parameters

```
int (^aBlock)(int x, int y) = ^(int x, int y) {
    return x + y;
};

int test = aBlock(1, 2); // test = 3
```

Blocks

External Variables

Value Captured

```
int outsideInt = 10;

int (^aBlock)(int x, int y) = ^(int x, int y) {
    return outsideInt + x + y;
};

int test = aBlock(1, 2); // test = 13
```

Blocks

__block Variables

```
__block int outsideInt = 10;

int (^aBlock)(int x, int y) = ^(int x, int y) {
    outsideInt = outsideInt + x + y;
    return outsideInt;
};

int test = Hello(1, 2);  // test = 13
                         // outside = 13

int test2 = Hello(1, 2); // test = 16
                         // outside = 16
```

Blocks

An easier way to define them

```
typedef void (^theBlock)(void);

theBlock aBlock = ^{
    NSLog(@"This is the block");
};
```

Blocks

So what are they?

- NSObject type
- Stack based variables that needs to be copied to heap
- Makes code more readable and concise
- Same memory rules

Blocks

Usage

- **Code to execution on completion of async operation**

```
[UIView animateWithDuration:0.33
                 animations:^{
                     // animations here

                 }
                 completion:^(BOOL finished) {
                     // what to do when animations are done

                 }
];
```

Blocks

Usage

- Code to execution on completion of async operation
- Lightweight thread operation using GCD

```
// send work to a background queue
// and return immediately

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
    // work
});
```

Blocks

Usage

- Code to execution on completion of async operation
- Lightweight thread operation using GCD
- Function for sorting

```
[array sortedArrayUsingComparator:^NSComparisonResult(id obj1, id obj2) {
    if (obj1 > obj2)
        return NSOrderedAscending;

    if (obj1 == obj2)
        return NSOrderedSame;

    if (obj1 < obj2)
        return NSOrderedDescending;
}];
```

Blocks

Usage

- Code to execution on completion of async operation
- Lightweight thread operation using GCD
- Function for sorting
- In short, very widely used

Project Exercise

Design Data Model, Play with TWRequest

Data Model

Some basic properties

- *NSString* Tweet Body
- *enum* Status Flag
- *NSString* Tweet ID
- *ACAccount* Twitter Account

Start iOS Project

Steps to setup

1. New Project
2. Select “Empty Application”
3. Give it a name and optionally;
    1. Company identifier
    2. Class prefix
4. Check “Use Automatic Reference Counting”
5. Give it a place to exist and enable version control

Start iOS Project

Import your data model

1. Locate your model's interface and implementation files
2. Drag and drop it into your iOS project's navigator
3. Ensure it copies items to destination folders

TWRequest

Things to know

- Needs system Twitter Account
- Handles all networking, just set the parameters
- Makes extensive use of Blocks
- https://dev.twitter.com/docs/api/1/post/statuses/update