An Introduction to Modern Software Development Tools Creating A Simple GUI-Based Tool

Apple's XCode Version 3.2.6

Charles J. Ammon / Penn State
August, 2011

In this exercise I will show you how to create a very simple GUI (Graphical-User-Interface) tool to perform a simple calculation. You will write about five lines of code to accomplish this, but see how if you write some more, you could make a functional, specialized utility to perform simple, routine calculations. We'll perform a very simple calculation converting a moment magnitude to a seismic moment. The equation that we'll use is

$$M_o = 10^{\left[\frac{3}{2}(M_w+10.7)-7.0\right]},$$

where Mo is the seismic moment in N-m, and Mw is the moment magnitude. In the C programming language, this equation looks like

```
moment = pow(10.0, 1.5*(Mw + 10.7)-7.0);
```

This is an easy calculation, we don't want to get bogged down in complicated computations, our focus is on the tool again (which is the opposite of what you should do when doing your science).

GUI or No GUI

The first question you should ask your self before developing a tool with a GUI is whether it is worth the time. Most of the time when you are just writing a script of tool to process some data, you might be creating a one-off tool that you will use a few times. Don't waste time with a GUI in that case. Even if you think you may run something many times, a GUI may not be the right choice. But if a GUI can make a process more efficient, enable you to view or process more data faster and more reliably, then it's worth some investment of time.

Our goals in this exercise is to show you how to implement a GUI simply so you know how these things are built. We won't have many features and it will be relatively painless. But keep in mind that the more elements in your GUI, the more tedious it can become, and the more you should think before you implement the GUI. Always remember that science is your business, not programming.

Mac (and iOS Development)

Serious Apple development is done using Xcode and the Objective-C language. Objective C is an object-oriented flavor of C, as is C++, for example. You will not find objective-c on other platforms so if you write in it, you are wedding your code to an Apple product. If you perform your calculations in mostly C constructs, the only thing really tied to Apple is the GUI code. I cannot teach you objective-c in an hour session of an information-dense short course, but the basic idea is that you create objects and send them messages.

For the GUI, Apple provides dozens of objects such that manage buttons, text fields, sliders, graphic views, web views, most everything you see in a standard Apple gadget application. Apple has invested a tremendous amount of effort to do so, and one of the problems for a beginner is finding what you want in all those options. The documentation of all these objects and their capabilities is also vast, so you have to use the online, searchable, documents and rely on command-line completion help to find the exact object and the exact message that you want to send to that object.

We're going to use just a few of these objects. If you plan to do more advanced development along these lines, learn these classes well: NSString, NSArray, NSMutableArray, NSDictionary, NSMutableDictionary, NSData and NSMutableData. The NS stands for NextStep, which is where Apple bought these foundation classes. These are just a few of the classes, but they are extremely powerful for manipulating data and parameters.

Object Addresses (Pointers) & Messages

When objects are created, they are stored in some part of memory and they are addressed by specifying what part of memory holds them. The "address" of a memory is stored in a "pointer". So, when you create an object, you get the address of the object, and using that address, you can pass messages to the object or pass the address of that object to other objects so they can interact. For example, if you had a seismogram object, you could send it a message asking for its maximum amplitude, tell it to remove the mean value from all the amplitude values, taper its endpoints, etc.

Suppose you had an object that responds to a function to return the mean value of a set of numbers contained in the object. The syntax for sending a message is to an object to get the mean value and store it in the variable theValue is

```
theValue = [myObject meanValue];
```

That's a contrived example, let's assume that our object is an NSString (one of Apple's foundation classes) and that the string contains a number, such as "4.35". To get the numerical value from the string, you can use

```
theValue = [myString floatValue];
```

So when you see square brackets in objective-c, generally you are seeing a message being sent to an object.

Application Delegates

One of the most important objects in our application will be what's called an application delegate. When an application launches, it creates an NSApplication object that runs the application. Fortunately, the NSApplication object has a helper called a delegate that is easy to customize. In the Cocoa template application, this delegate is called the applicationDelegate. That's the part of the source code that we'll modify to create a simple calculator.

IBOutlets and IBActions

Interface Builder is Apple's graphical tool for building user interfaces. In the most recent versions of Xcode, Interface Builder is part of Xcode. In the version that we are using, it is a separate, but complementary application. We will lay out our user interface in Interface Builder (IB) and then connect the interface to our program using IBOutlets and IBActions.

An IBOutlet is a variable in you program source code that is ready to be connected to an object in your user interface. An IBAction is a function in your source code that takes one argument, the address of the sender, and executes a particular action in your source code.

Creating a Simple GUI Calculator

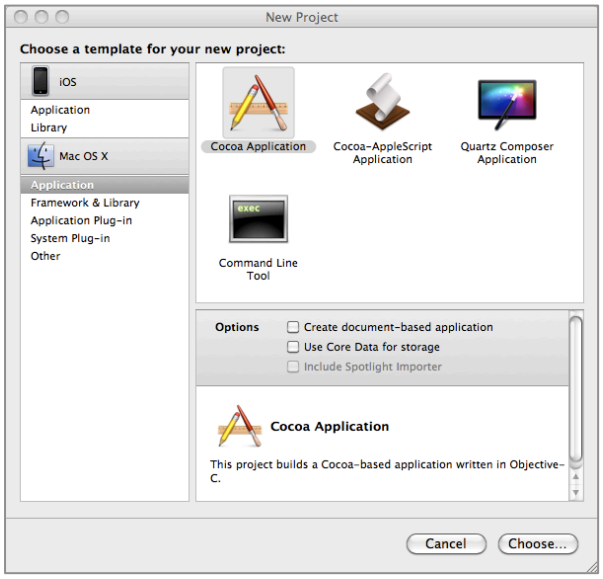
Here's the way we'll develop a simple moment calculator:

1. Create a Project in Xcode
2. Create two IBOutlets, one for Mw and one for the seismic moment.
3. Create on IBAction function in XCode that computes moment from Mw.
4. Lay out the user interface in Interface Builder
5. Connect the user-interface elements to the IBOutlets and IBAction
6. Compile, link, and run the application.

Creating a New Project in Xcode

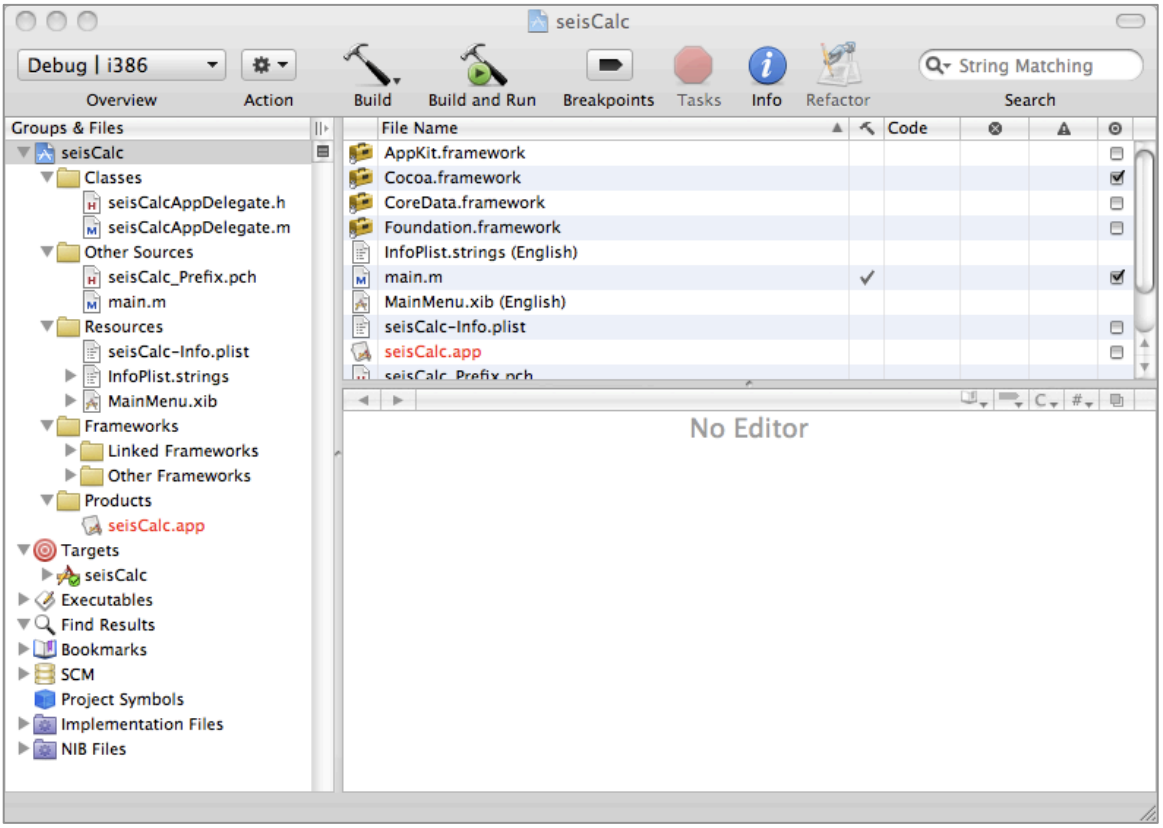
Launch Xcode and select Create a new Xcode project in the options on the left of the Welcome to Xcode Window.

Then select Application under the Mac OS X subtitle in the left column of the window. Then choose Cocoa Application and click the choose button at the bottom of the window.



To follow along with the notes, you should name your project seisCalc and save it in a new folder on your Desktop.

When you are finished creating the new project, you should see something like



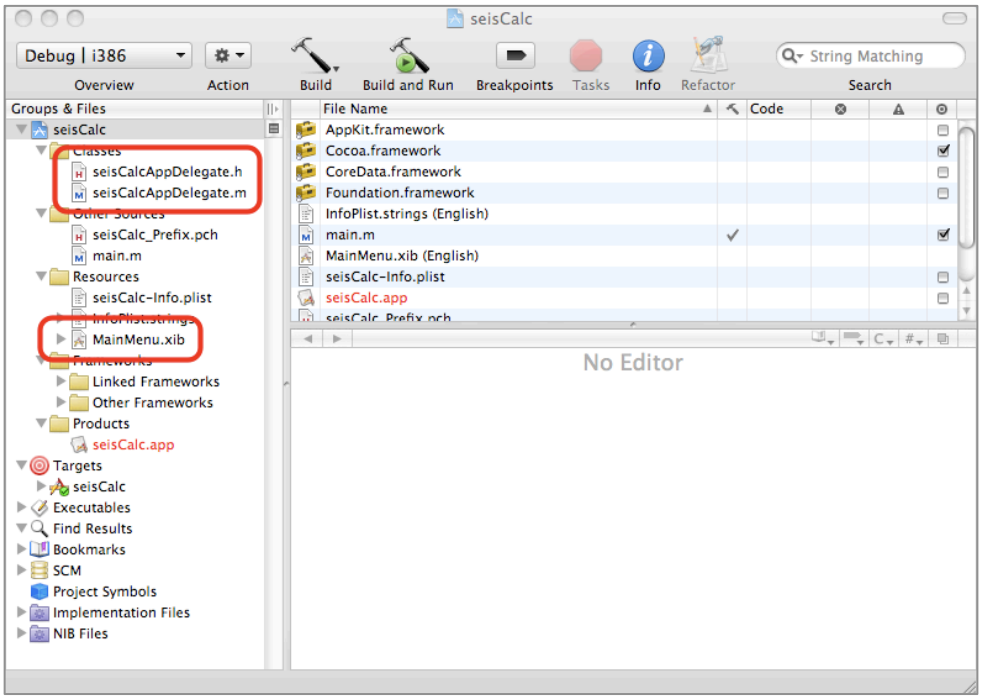
111

112 If you don't see all the files on the left, click the reveal triangles next to the folder
113 icons in the left column. We need only to work with three files in the list, the

114 • seisCalcApplicationDelegate.h

115 • seisCalcApplicationDelegate.m

116 • MainMenu.xib



The first is a header file (.h), the second is an objective-c source code file (.m), and the third is an Interface Builder file (.xib). Note that Xcode has created these files for us, they are part of the Cocoa application template. We just need to modify them.

Click once on the applicationDelegate.h file and edit it to look like this (lines that you change are in bold):

```
//
//  seisCalcAppDelegate.h
//  seisCalc
//
//  Created by Charles Ammon on 8/12/11.
//  Copyright 2011 Penn State. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface seisCalcAppDelegate : NSObject <NSApplicationDelegate>
{
      NSWindow *window;

      IBOutlet NSTextField *mwTextField;
      IBOutlet NSTextField *momentTextField;
}

@property (assign) IBOutlet NSWindow *window;

- (IBAction) mwToMoment:(id) sender;

@end
```

You added two IBOutlets and declared one IBAction. The syntax requires the dash in front of (IBAction) in the name, and an IBAction must have one argument of type (id) and you should always call the argument sender. When the action is called, the sender will contain the address of the object that sends the message mwToMoment to our applicationDelegate. We won't need to use that value (but sometimes you can query the sender for information), our action proceeds in the same manner when called by any sender.

Now click once on the applicationDelegate.m file and edit it to look like this (changes to the file are shown with a bold font style):

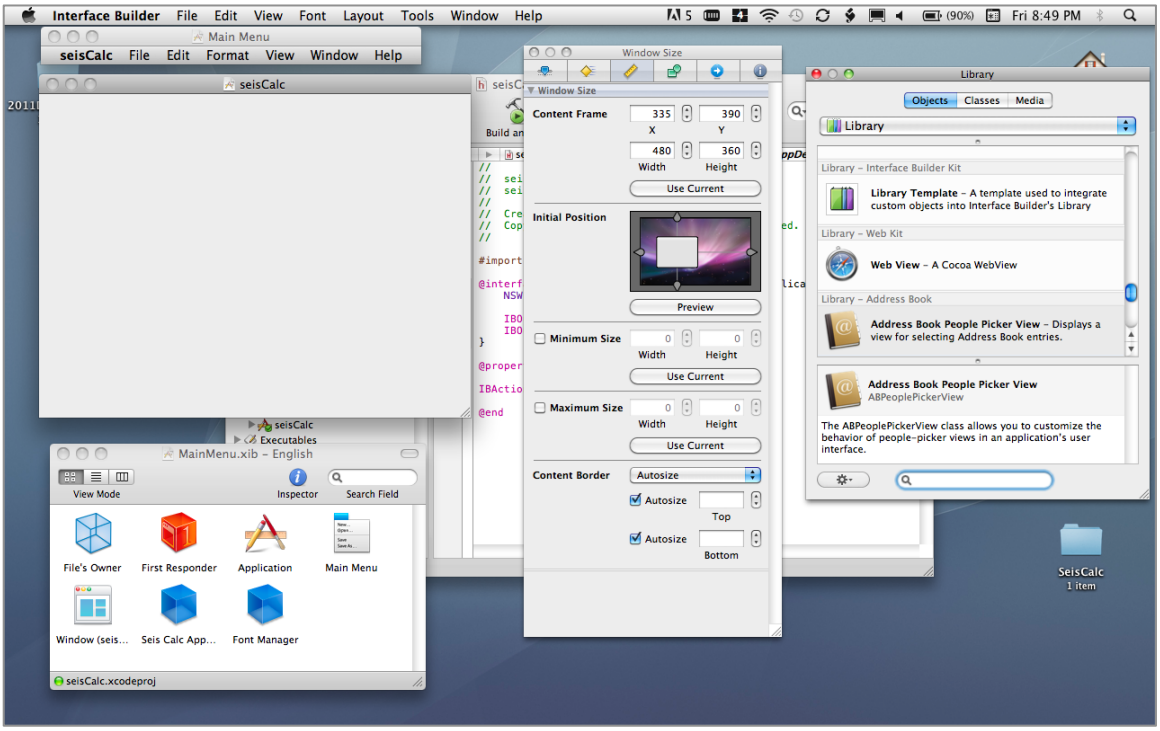
```
//
//  seisCalcAppDelegate.m
//  seisCalc
//
//  Created by Charles Ammon on 8/12/11.
//  Copyright 2011 Penn State. All rights reserved.
//

#import "seisCalcAppDelegate.h"

@implementation seisCalcAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(NSNotification
*)aNotification {
        // Insert code here to initialize your application
}

- (IBAction) mwToMoment:(id) sender;
{
        double moment;

        double Mw = [mwTextField doubleValue];

        moment = pow(10.0, 1.5*(Mw + 10.7)-7.0);

        // This is one line of source code
        [momentTextField setStringValue:
        [NSString stringWithFormat:@"%9.3e",moment]];

}

@end
```

You just created an action that extracts the value of the text field called mwTextField, and then uses the value (Mw) to complete the seismic moment. Then you set the string in the momentTextField equal to the result of the calculation.

To summarize our source code changes, we created addresses or pointers to hold two instances of NSTextField's and one action that will convert the value of the string in the mwTextField into a seismic moment and show the result in the momentTextField. We wrote eight lines of objective-c.

Interface Builder

Interface Builder (IB) is the graphical tool for creating user interfaces. IB talks to Xcode when both are running, so changes that you make in one are transmitted to the other. To launch IB, double-click on the MainMenu.xib file in the left column of the Xcode window. You should see something like this (you may have to open the Library palette shown on the far right). The small Main Menu window contains the menus that will be shown in our application. The window just below that is our application's window and is by default titled seisCalc. This window is set to open automatically when our application is launched.

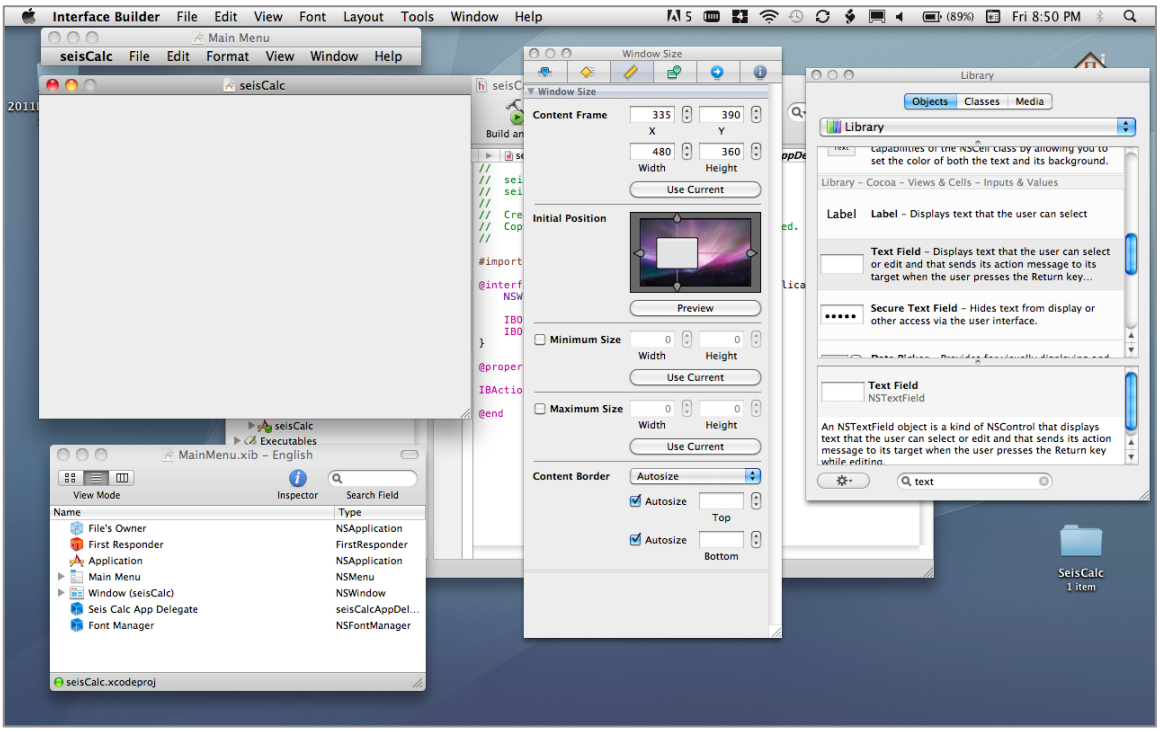


The window on the lower left shows the objects that are contained in the MainMenu.xib file. The window in the middle is the Inspector, which allows you to customize objects that you add to our interface. The Inspector shows information about our application window because I clicked on the seisCalc window.

You can see in the MainMenu.xib window (lower left) that the template created an instance of our application delegate – the first blue cube icon. When the application is launched, the object corresponding to that icon is linked to our application delegate object. That allows us to add a button that sends a message to our application delegate, and to create text fields that correspond to our application delegate's IBOutlet's.

To add items to our application user interface, we drag them from the library into our window (we'll get to that in a short while). Select the objects tab in the Library palette and browse the objects that we have to work with. We're going to add three labels, two text fields and one push button.

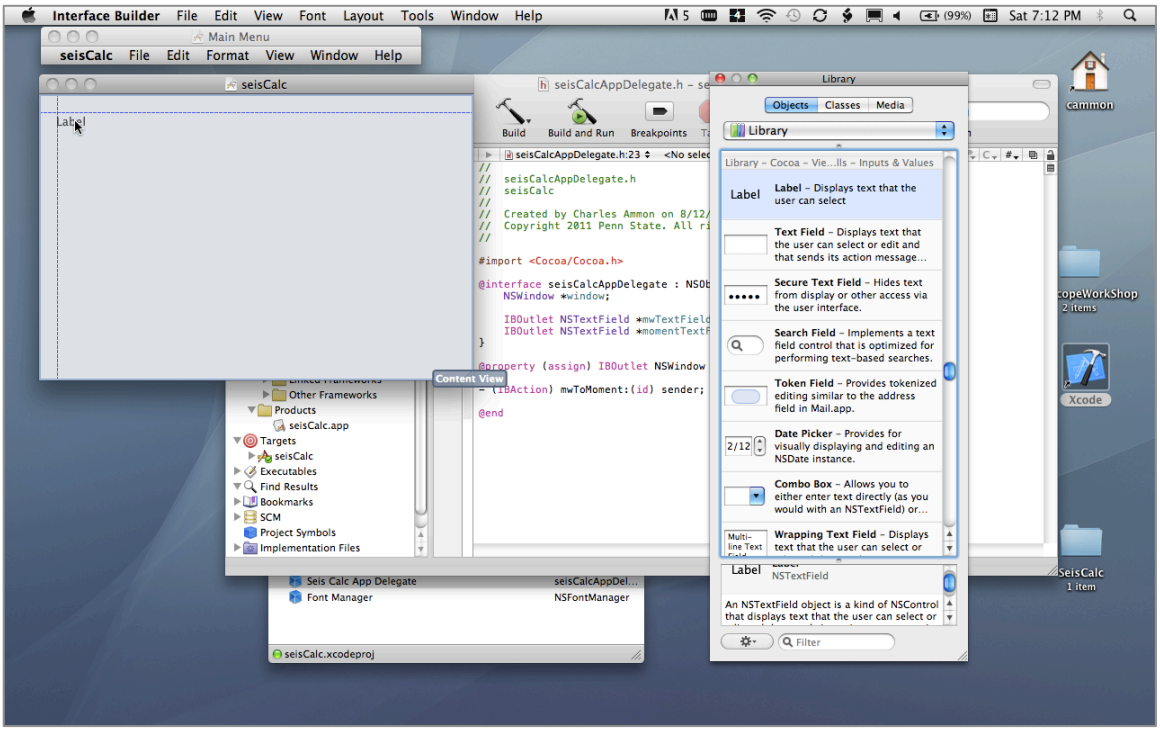
You can filter the objects by typing a string in the search box at the bottom. If you enter the work 'text' you see objects related to text interface elements as shown in the next illustration.



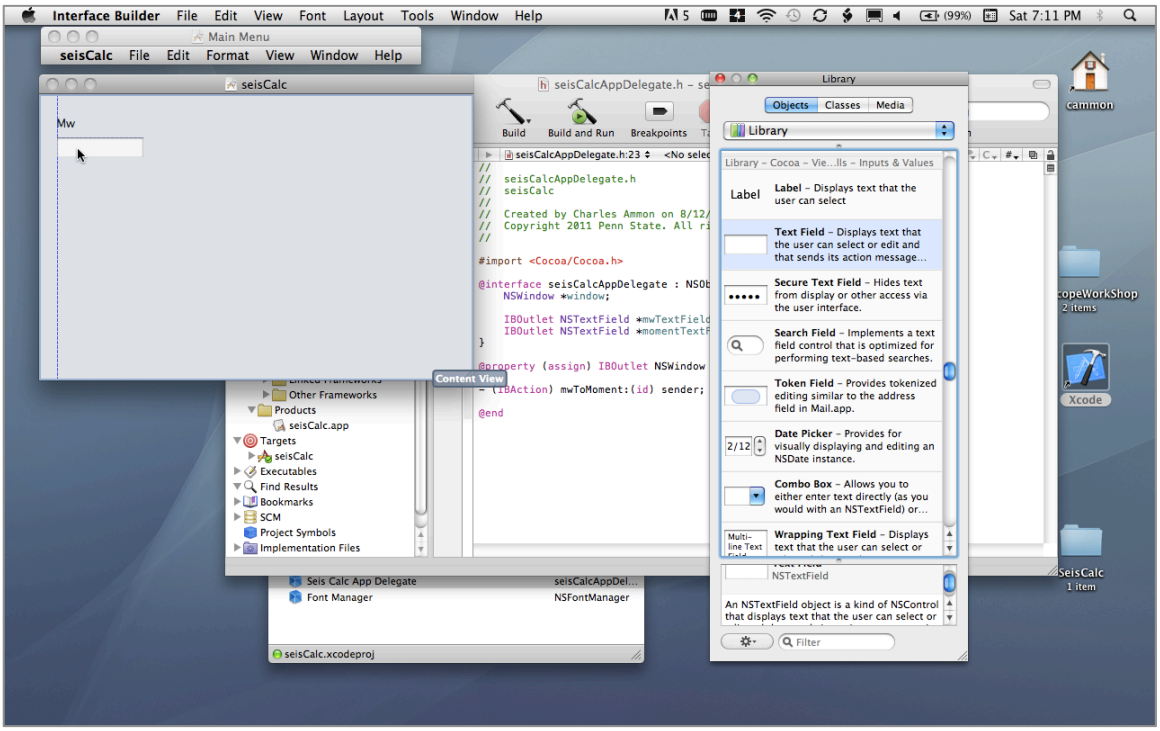
Building Our Interface

Drag a Label item from the Library Window to the seisCalc window. To do that, you click and hold the mouse on the label in the Library palette and drag it across to the seisCalc window. When your mouse enters the seisCalc window IB will begin to show you the size of the label object and if you move near the edges of the window, it will show you guides to help place the object at suggested distance from the window's edge.

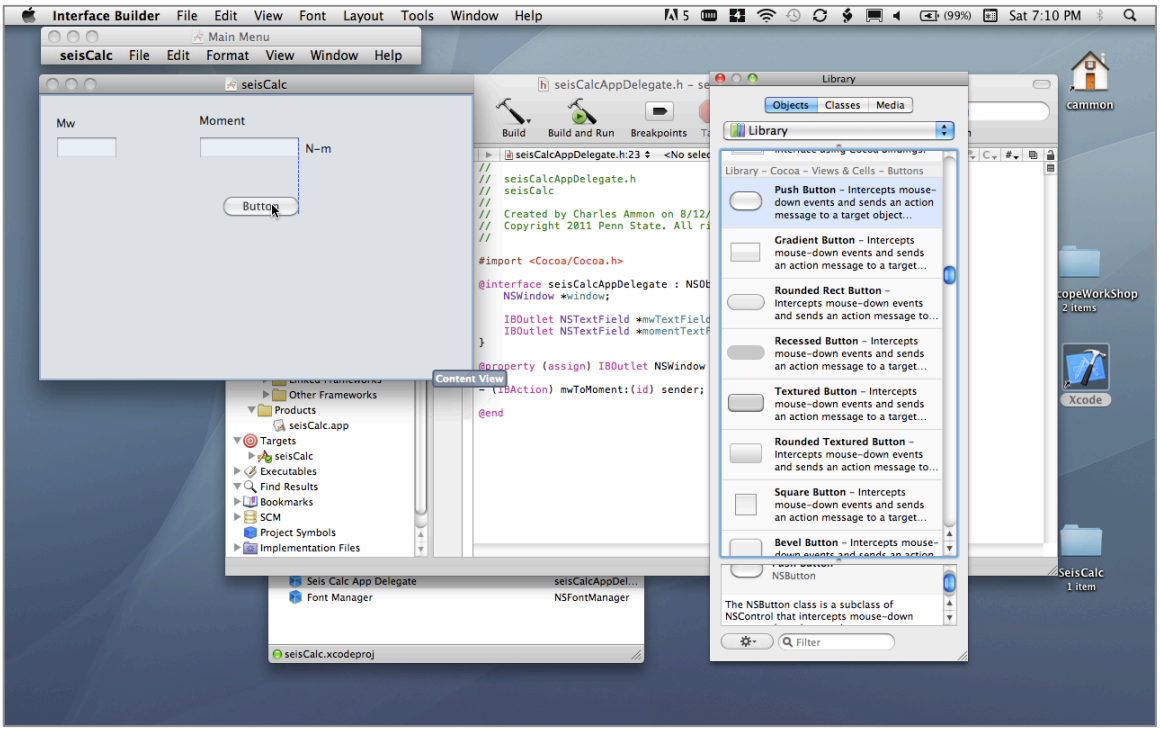
Place the first label in the upper left of the application (seisCalc window). When you have it positioned correctly, release the mouse button and you will have added an element to our application's user interface. You can double-click on the label to change the string to read "Mw". You can adjust the size of the label if you want. Labels are static text objects that provide information to the user, you can change labels from the program if you want, but for our purposes, we don't want to change this, so we're done with that element.



Now drag a Text Field item from the library to the seisCalc window and align it below our label. This text field is where the user will enter the value of moment magnitude (Mw) that we will use to compute the seismic moment.



Repeat the process to create a Label and Text Field for the seismic moment Add a third Label to provide the units for the seismic moment value. Now change the text in the search field of the Library window to “button” (without quotes). Then drag a Push Button object from the Library palette to the application window. Double click the button to rename it “Calculate”.

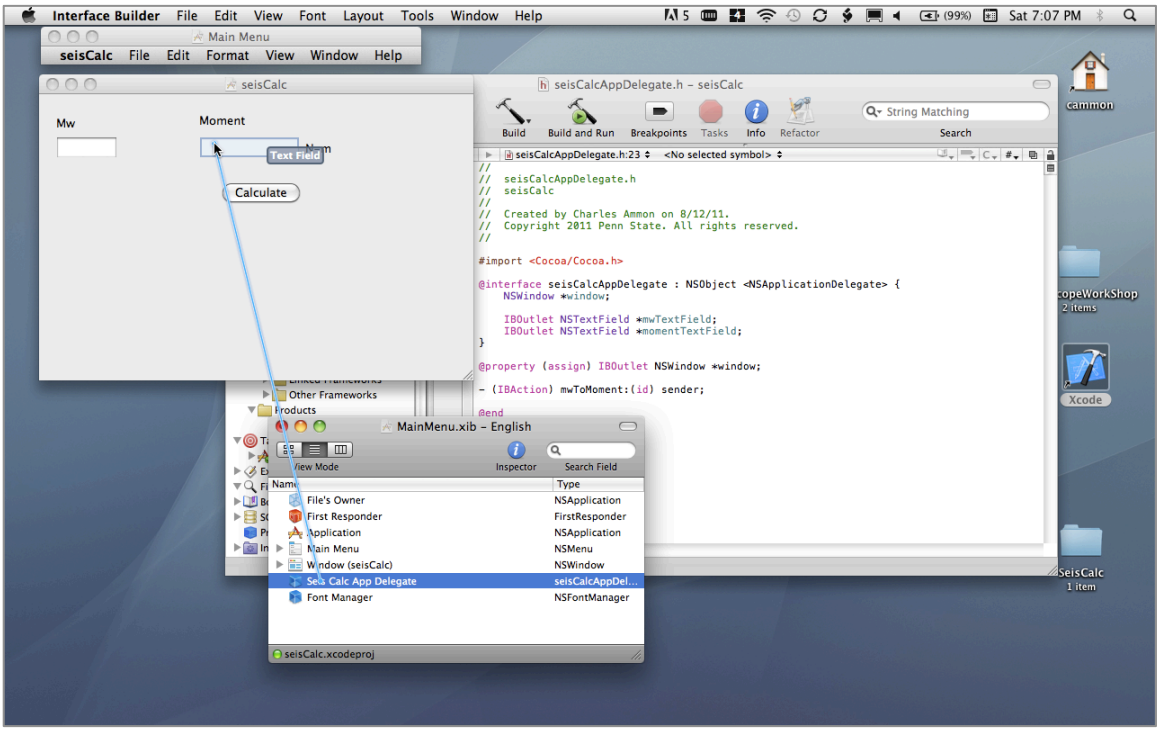


That's our simple user interface – we have all the objects in position. Three labels and three control objects (two text fields and one push button). All that remains is to connect this interface to the object references in our source code. We are going to connect the text fields to our two IBOutlets and set the push button so that is executes our IBAction function to compute the moment from the Mw.

Connecting IBOutlets and IBActions

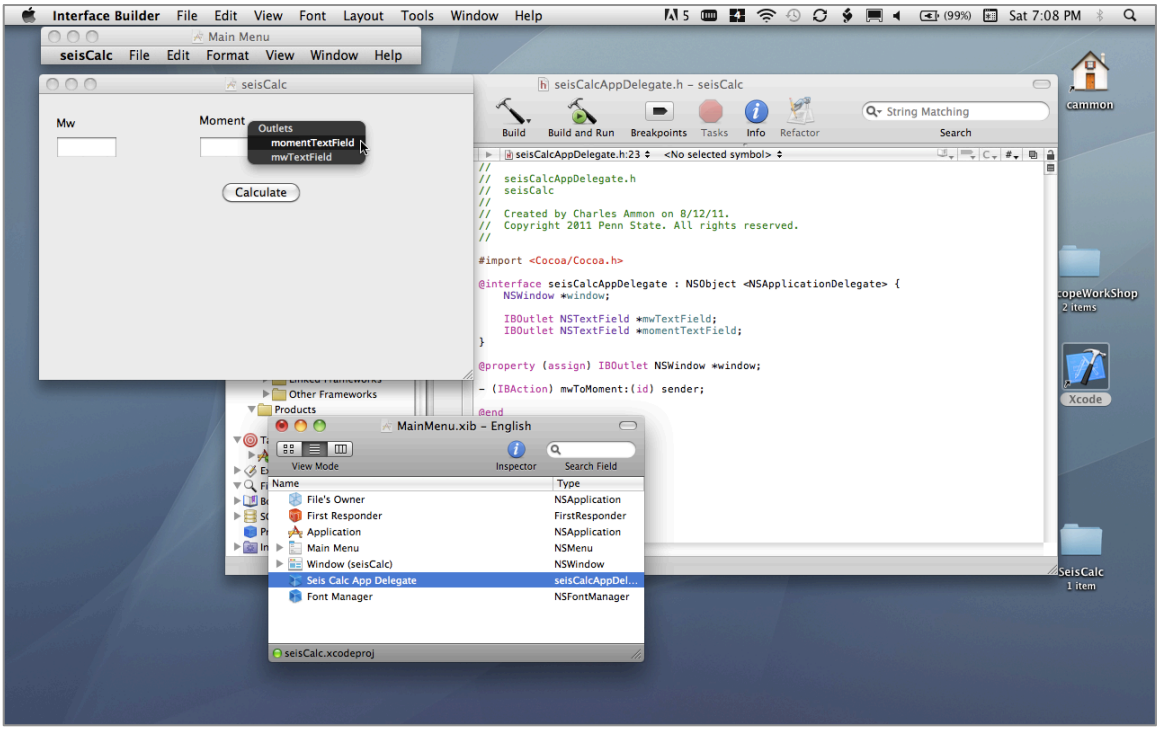
We make these connections using IB and the mouse. Recall that the MainMenu.xib has an instance of our applicationDelegate (the blue cube in the MainMenu.xib window. The applicationDelegate object will be easier to see if you click the view as list icon in the toggle located in the upper left of the MainMenu.xib window.

To connect the IBOutlets in the applicationDelegate to the objects in the user interface we control-drag from the applicationDelegate object to the Text Field. Place the mouse over the applicationDelegate cube in the MainMenu.xib window. Then press the control key, then click and hold the mouse button down. Move the mouse from the applicationDelegate item to the text field that will hold the value of Mw. You should see a blue connection line following the mouse (and you should still have the control key pressed).



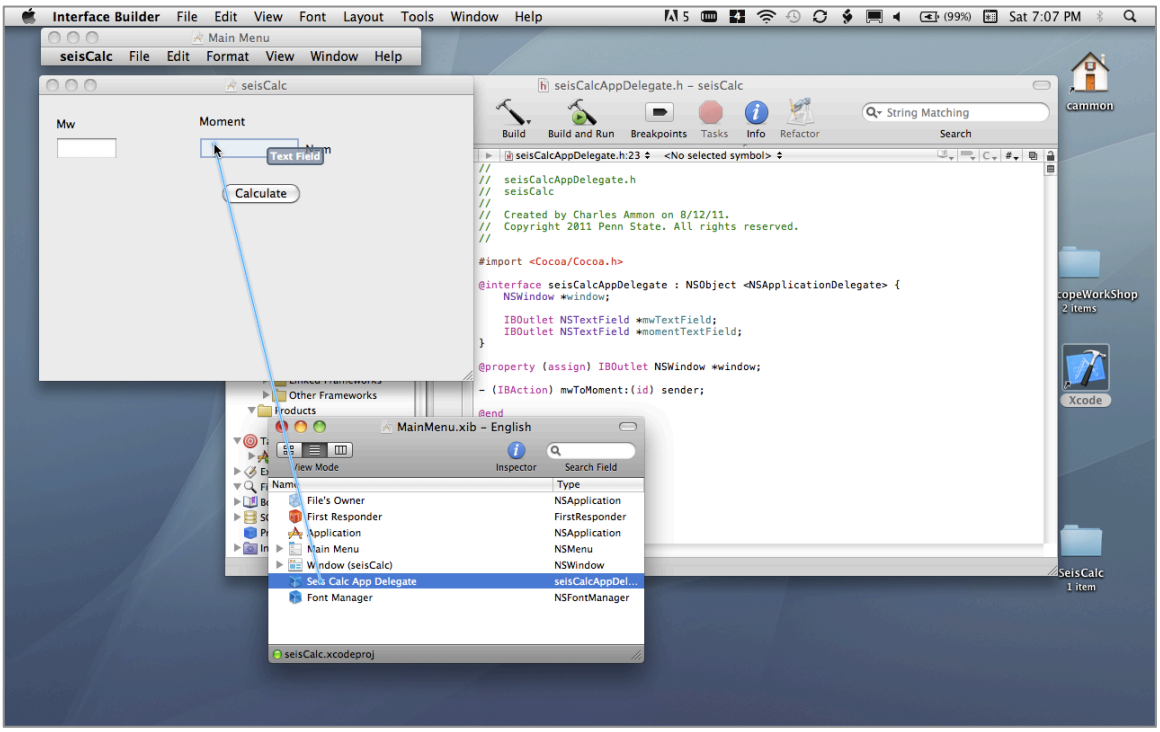
270

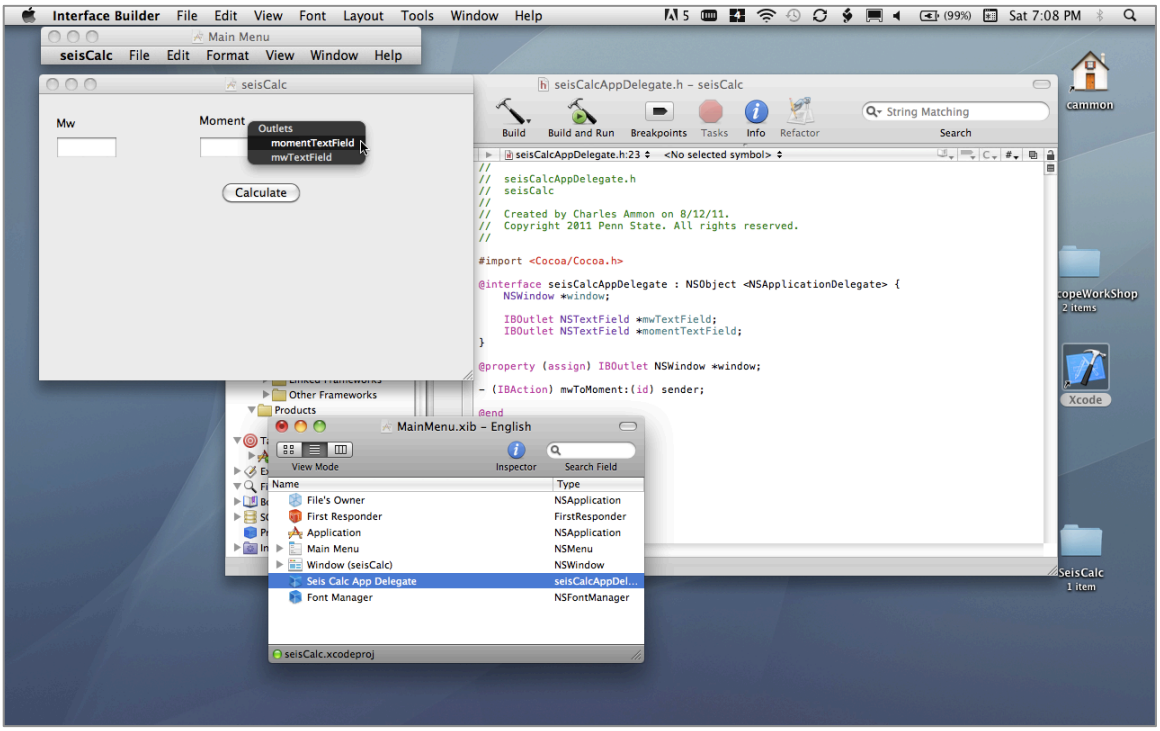
271 When the text field highlights, release the mouse button and small popup list of
272 our IBOutlets should appear.



273

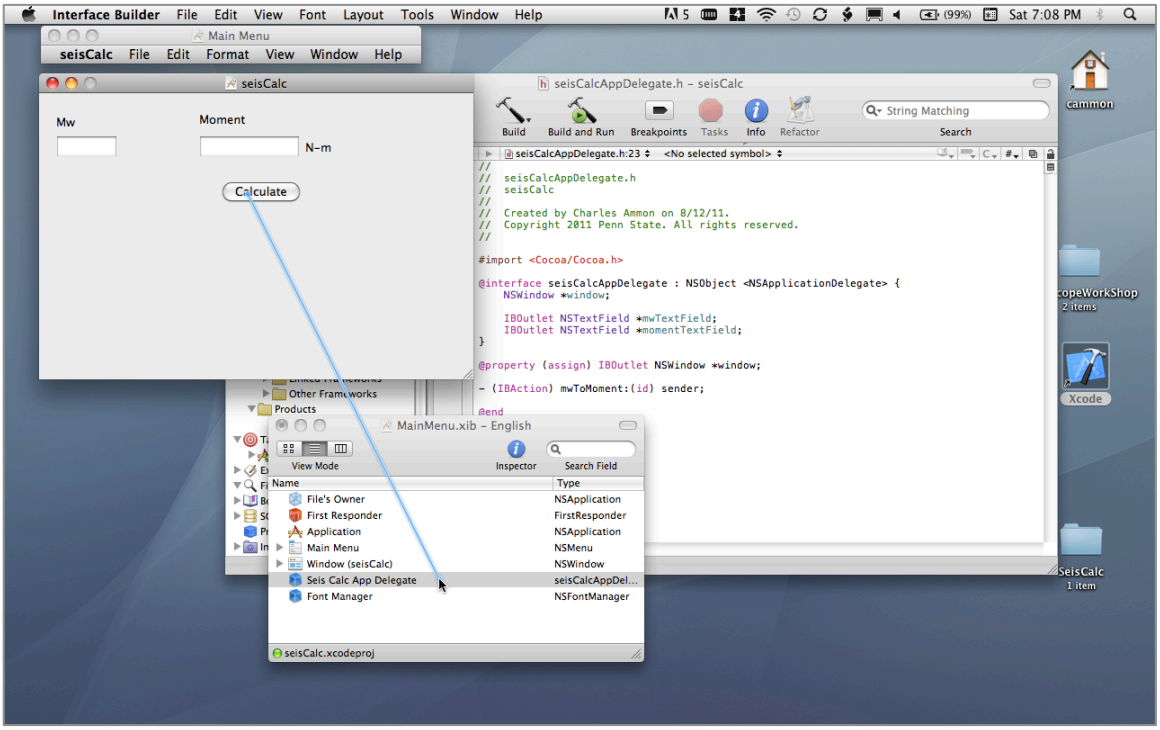
IB gets the names of objects in that list from Xcode. Those are the names of the IBOutlets in our applicationDelegate source code. Select the mwTextField outlet from the list. You just completed your first IB connection. Repeat the procedure and connect the seismic moment text field with the momentTextField IBOutlet.



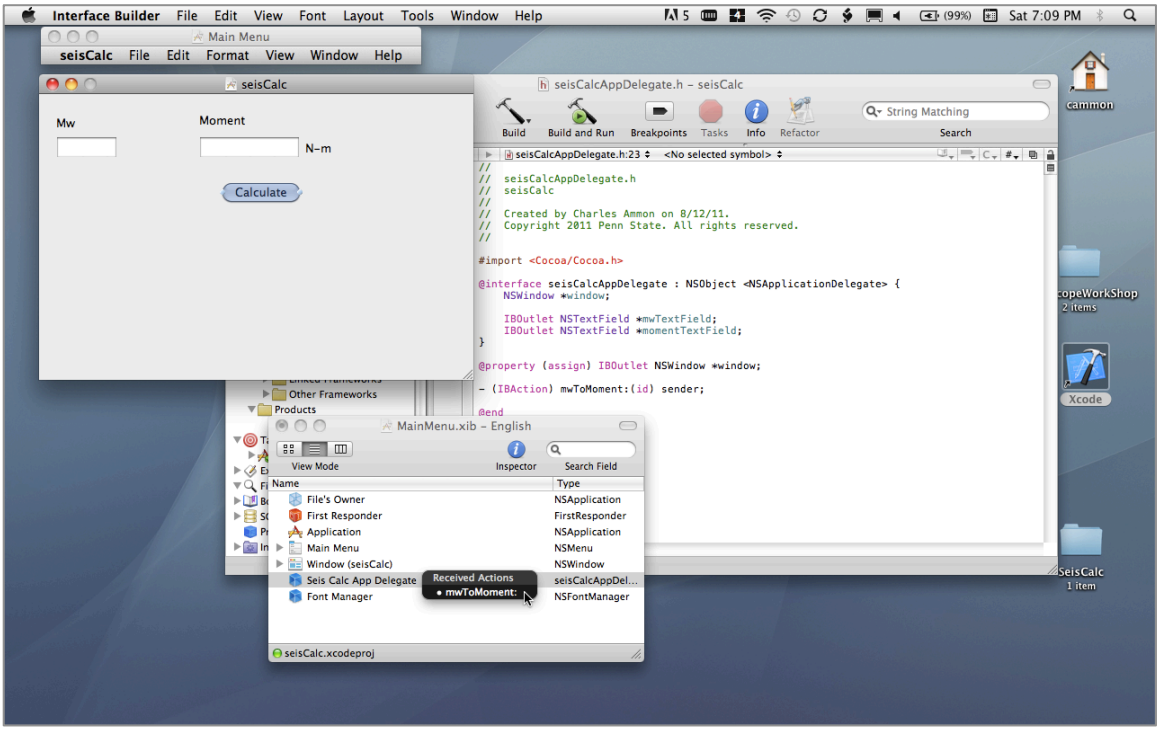


That completes out IBOutlet connections. You can close the Library palette if you want. It wouldn't hurt to save the MainMenu.xib file now. We're not quite done, but we can back up the work we've done so far.

Now we need to connect our push button to our applicationDelegate's IBAction function. The procedure is similar, except now you connect from the push button to the applicationDelegate cube icon. Move the mouse over the button, press and hold the control key, then click the mouse button and move from the push button down to the appicationDelegate cube icon in the MainMenu.xib window. When the applicationDelegate cube icon highlights, release the mouse button and you should see a menu list of the applicationDelegate's IBAction methods. Select the mwToMoment action (it's the only one there in our simple app; see the net two figures.



292

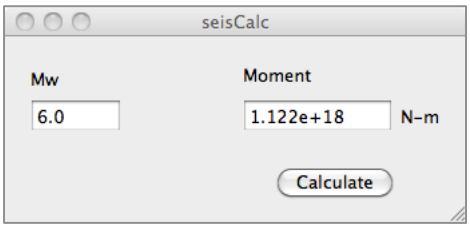


293

294 Save the MainMenu.xib file and return to Xcode (click on one of it's windows).

Build and Run

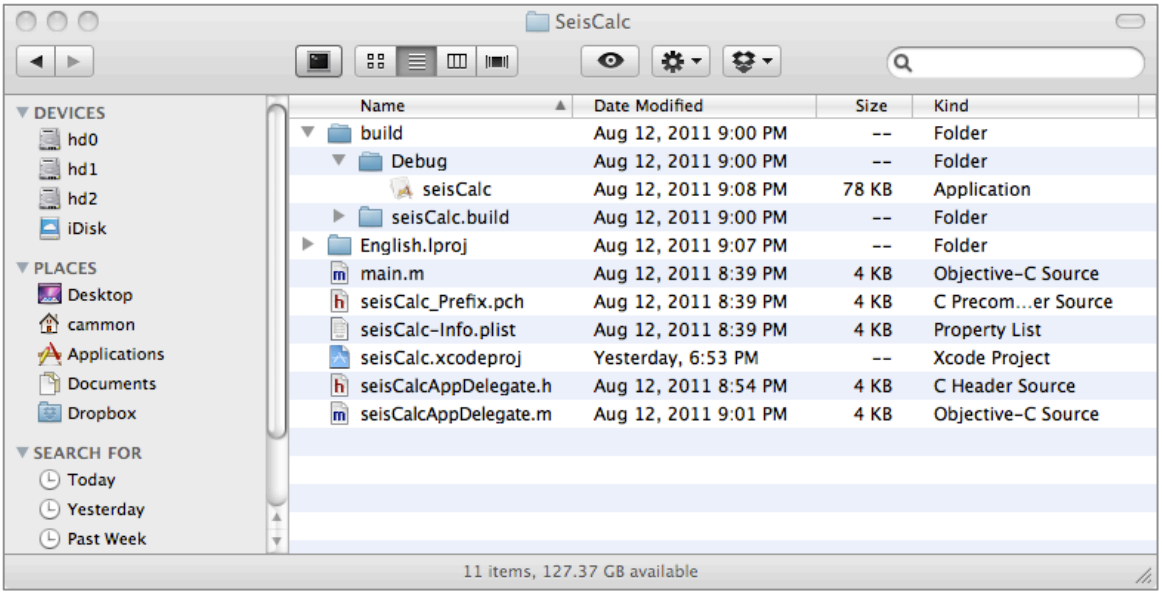
If all went well, then it's time to click the build and run button in Xcode and the code should compile, link, and launch. Enter a value of Mw into the appropriate text field and click the calculation button. You should see the moment appear in the other text field. The standard Text Field objects support all the common text processing (copy, paste, cut, etc.).



If the application did not launch, check the error and warning messages and fix any typos that may be causing problems. If the application compiles and launches but is not functioning, check you connections in IB.

Where is the Application?

When you complete your application, you can find it in the build subfolder of the project folder.



Learning More

We wrote eight lines of code, dragged a few items onto a window and connected them graphically to create a simple utility. You can add more items to you utility by following the same procedures. This ends our little tutorial. If you want to know more, start looking for additional tutorials at developer.apple.com. And search with Google, there are text and video tutorials all over the web. If you decide to pursue this kind of programming further, become familiar with the documentation and how to use it effectively. There are also some reasonably good books on Cocoa (Mac) and iOS programming.