An Introduction to
Modern Software Development Tools
Creating A Simple GUI-Based Tool
Apple's XCode Version 6.4

Charles J. Ammon / Penn State
August, 2011, 2015

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-to-ten 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 (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 or Swift (or both) languages. Objective C is an object-oriented flavor of C, as is C++, for example. Swift is a new object-oriented language that looks similar to C at time. Right now you will not find objective-c or Swift 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-like constructs, the only thing really tied to Apple is the GUI code. I cannot teach you swift or 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 that manage buttons, text fields, sliders, graphic views, web views, most everything you see in a standard Apple 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 (or their Swift counterparts): 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.

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 Application object that runs the application. Fortunately, the Application object has a helper called a delegate that is easy to customize. In the Cocoa template application, this delegate is called the AppDelegate. 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 and it is par is part of Xcode. 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 your program's 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 (AnyObject), and executes a particular action in your code.

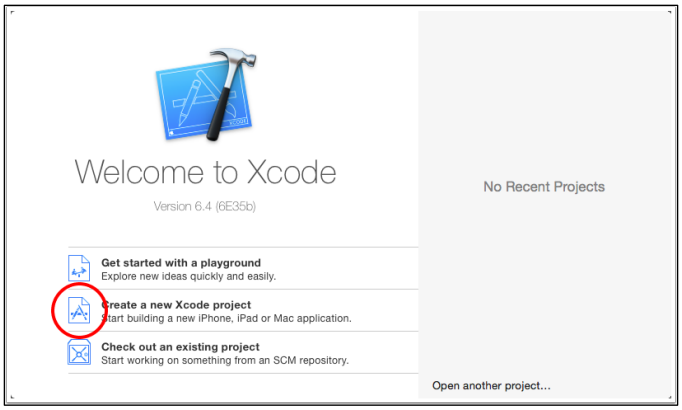
Creating a Simple GUI Calculator

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

1. Create a Project in Xcode.
2. Create a SeisCalcWindowController object to "control" our tool.
3. Delete the default window used in the application template.
4. Set up our SeisCalcWindowController to provide the application window.
5. Lay out the user interface in XCode.
6. Create two Swift IBOutlets, one for Mw and one for the seismic moment.
7. Create on IBAction Swift function that computes moment from Mw.
8. Connect the user-interface elements to the Swift IBOutlets and IBAction.
9. Compile, link, and run the application.
10. Save a snapshot of the code (as in git).

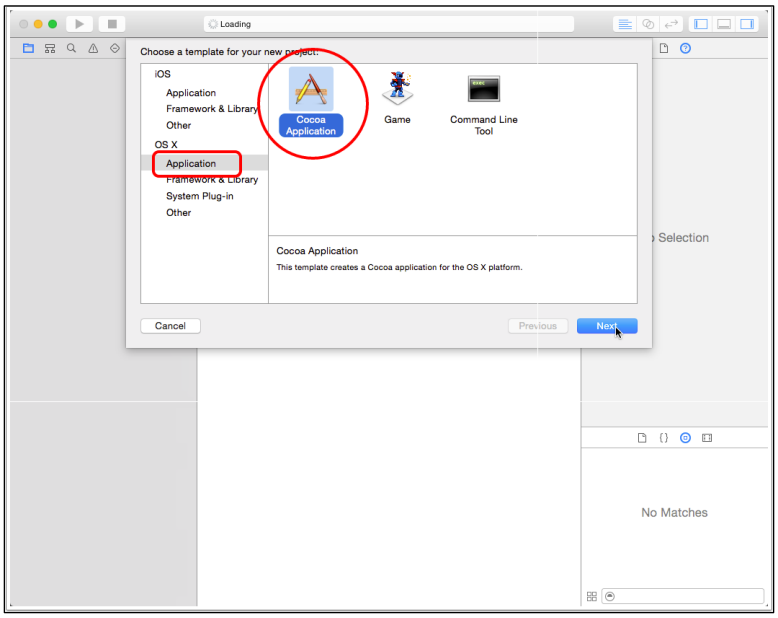
Creating a New Project in Xcode

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



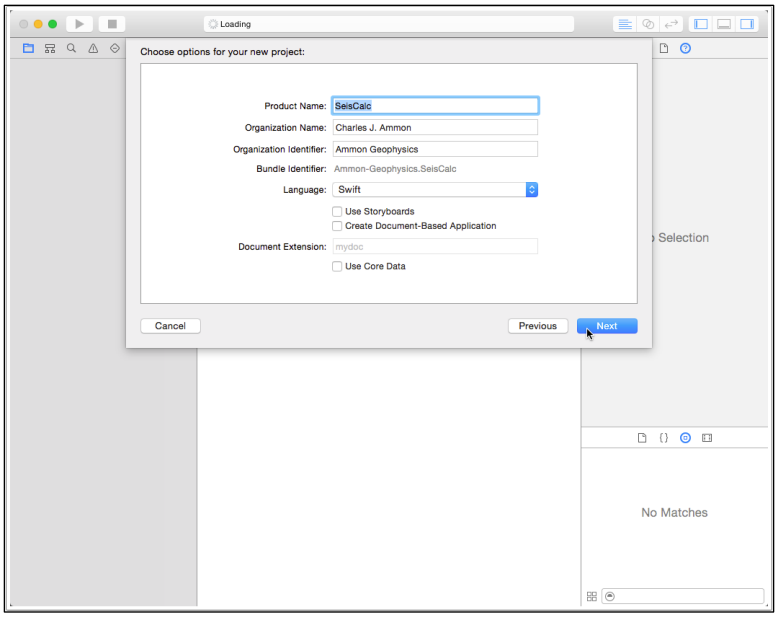
92

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

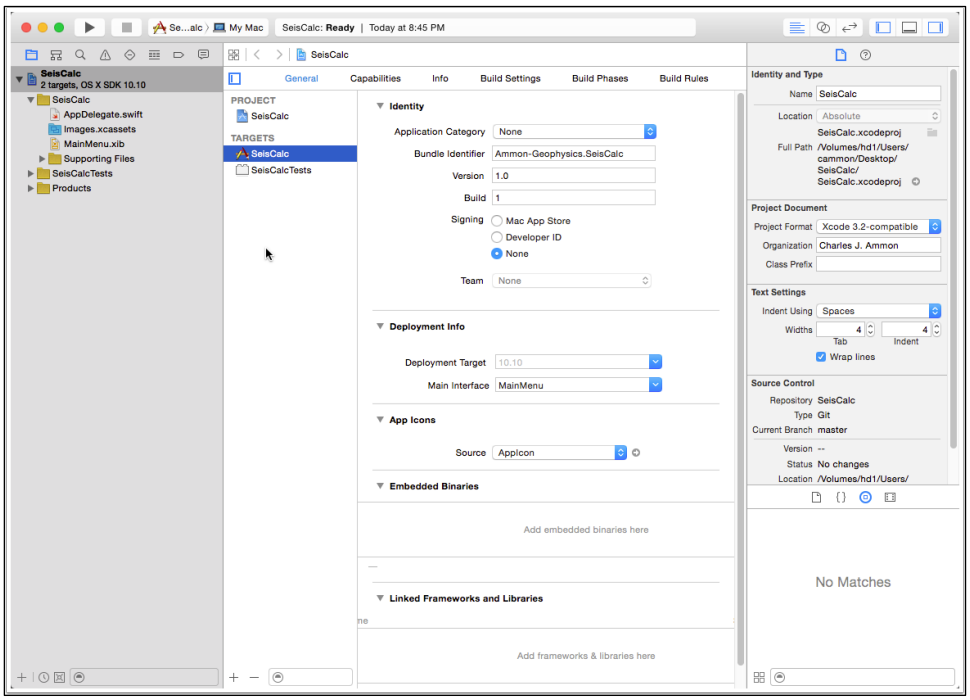


96

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

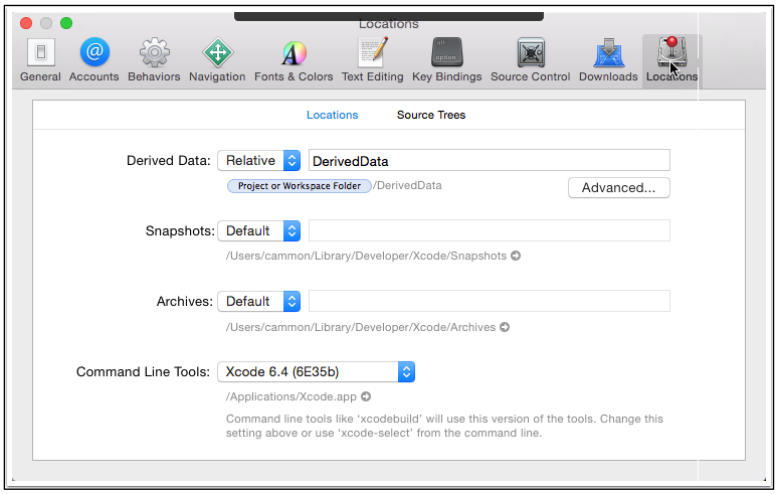


When you are finished creating the new project, you should see something like



If you don't see all the files on the left, click the reveal triangles next to the folder icons in the left column.

Before continuing, it will be easier if we set one default on XCode – where it stores the compiled files. Open the Preferences window and select the "Locations" tab. Set the Derived-Data popup menu to "relative" as shown below. Then close the preferences and continue.

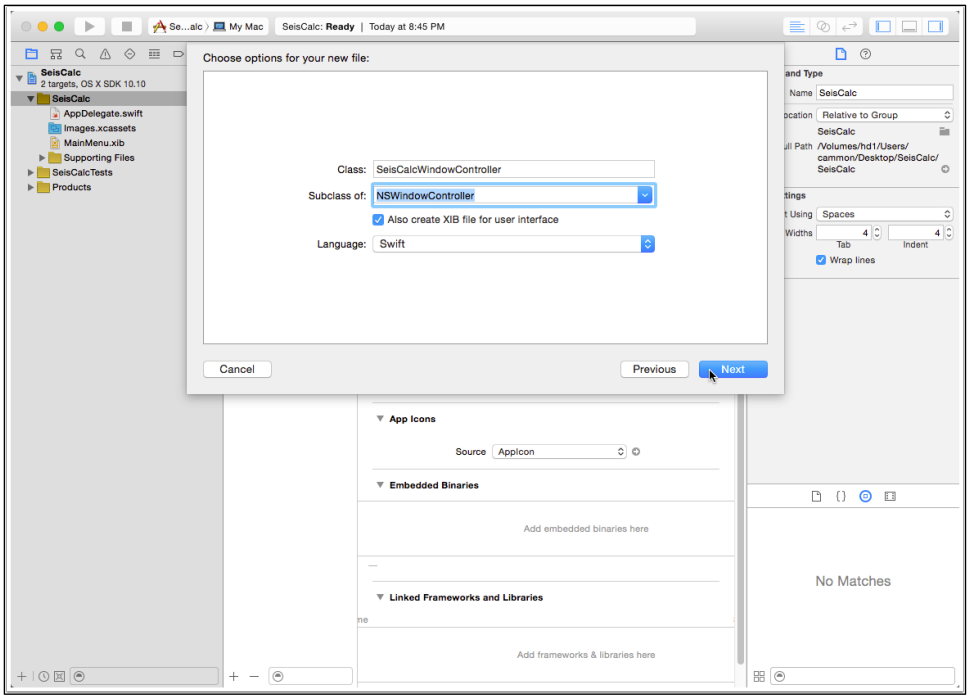


We need only to work with four files, the

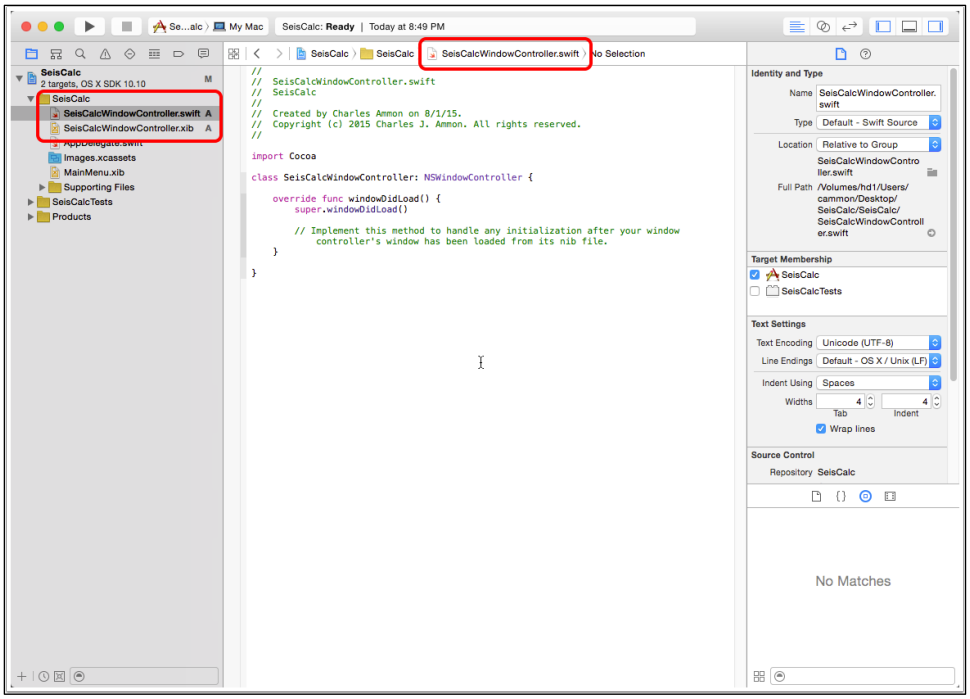
- AppDelegate.swift
- MainMenu.xib
- SeisCalcWindowController.swift
- SeisCalcWindowController.xib

We have to create the second and third of files to get started.

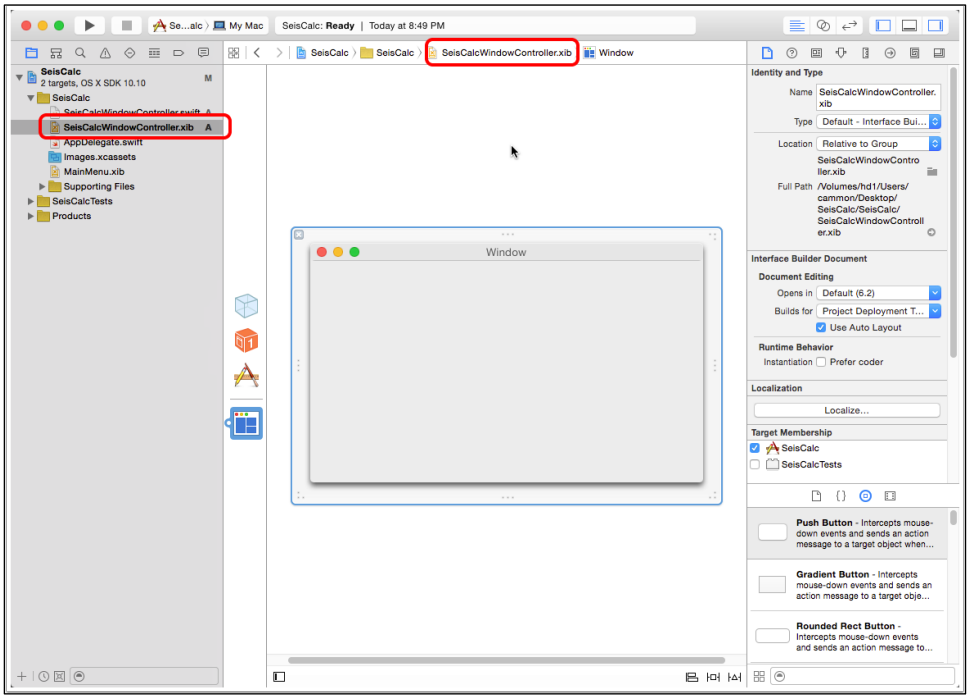
Select New File from the File menu and select the option that creates a file that is a subclass of NSWindowController. Make sure check the box to also create an "xib" file. The "zip" or "nib" files store elements from our graphical user interface. The next figure shows what to set up before you create the new file. We'll call our class SeisCalcWindowController. Guess what it does? It controls manages our SeisCal window…



The SeisCalcWindowController files will appear in the file list on the left of XCode. Single click on SeisCalcWindowController.swift and you will see:



Then single-click on the SeisCalcWindowController.xib file. This is the file that holds the initial graphical user interface elements. Initially, it's empty. We have to add our interface elements to this file.



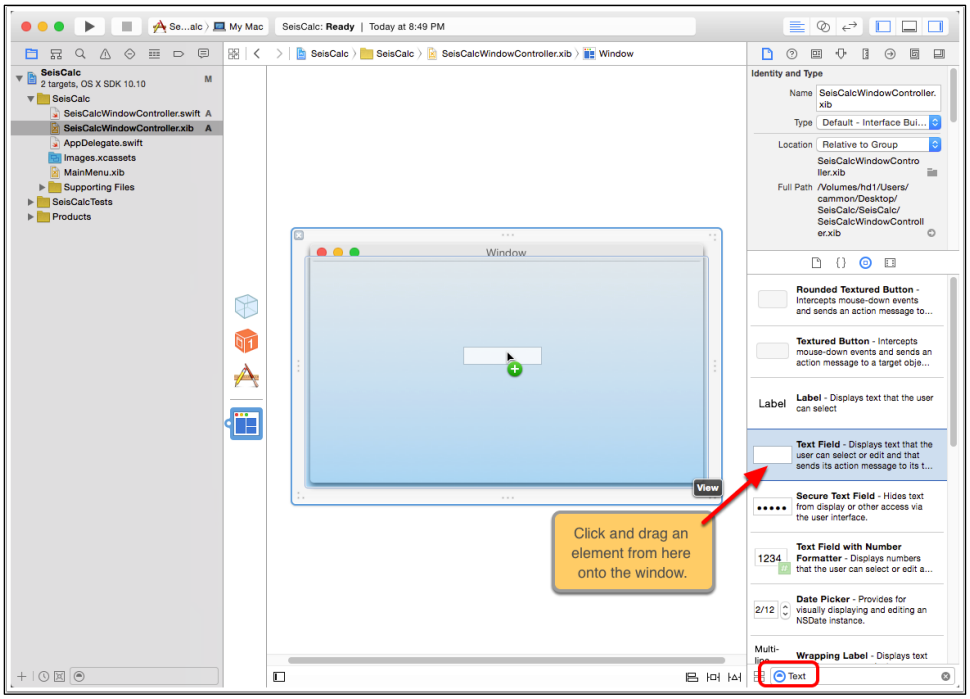
To add interface elements we use the list of objects in the lower right of XCode. You can filter the interface elements by typing into the text field at lowermost right of XCode – see below. We'll search for text-related elements.

Once you have limited the elements to those related to text items, drag a "Text Field" element onto the window and drop it. You can reposition and resize the element once it's on the window.

The text field is an input interface element – you can type into it (or copy text from it). Apple developed the Text Field class and it contains all the familiar text editing features. We get them with little effort on our part.

We'll also want to label our text field, so drag and drop a "Text Label" onto the window and position them as well. Double-click the label and change it to Mw.

Note that as you move items around the window, guides will appear to indicate positions that satisfy Apple's user interface guidelines. You should choose positions consistent with the guides whenever you can.

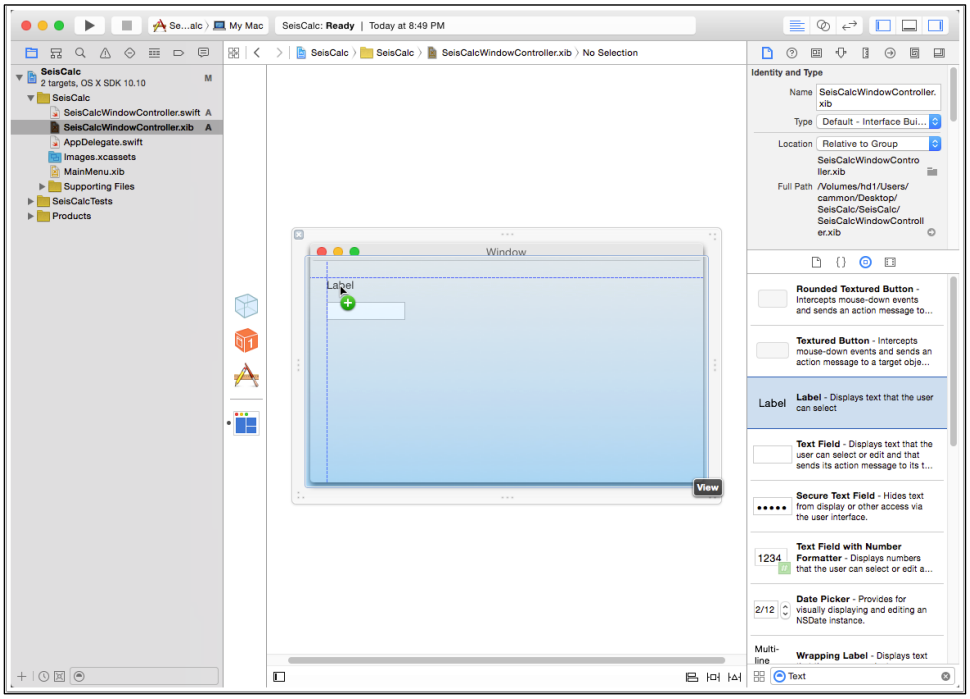


153

154

155

156

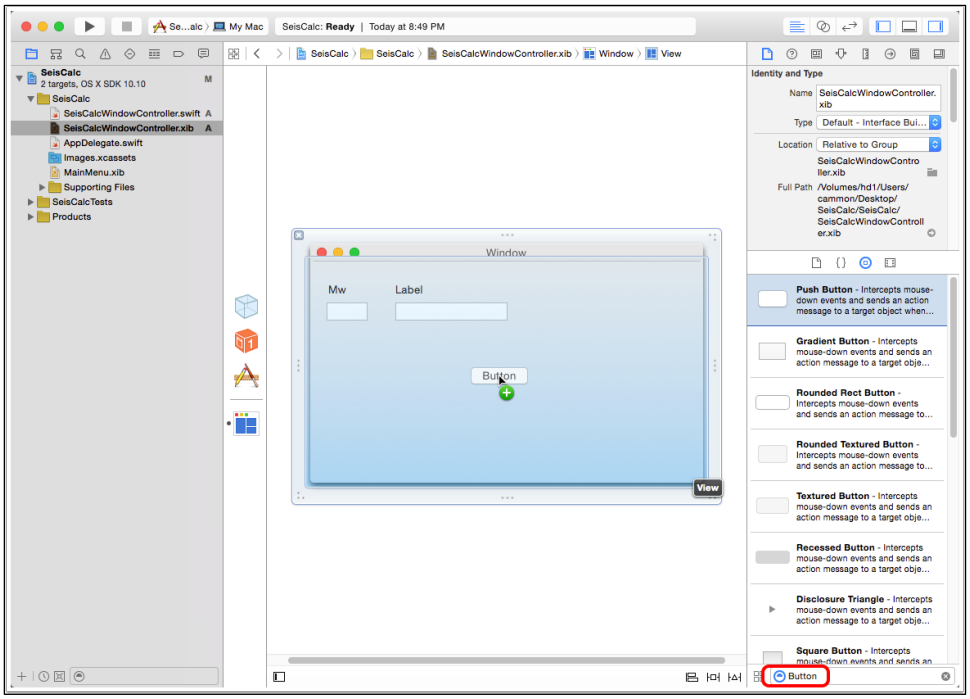


157

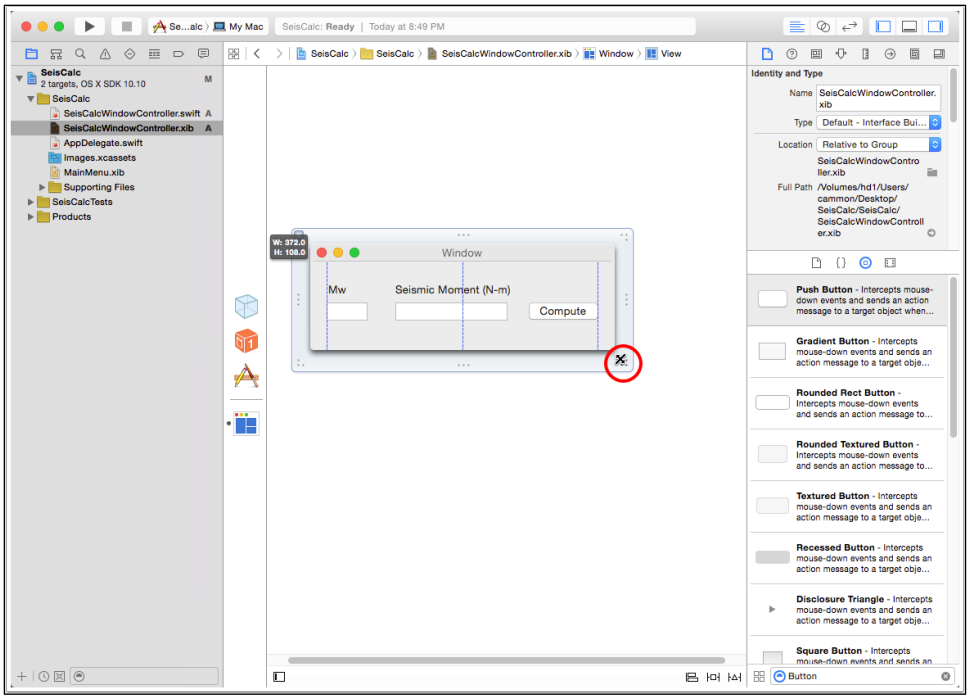
158

159

Next add another Text Field and another label, this time for the seismic moment. Then change you interface element search string (in the lower right) from “text” to “button”. Drag a “Push Button” onto the window, double-click it to rename it to “compute”.



Align the interface elements and resize the window to eliminate the blank spaces along the sides and the bottom.

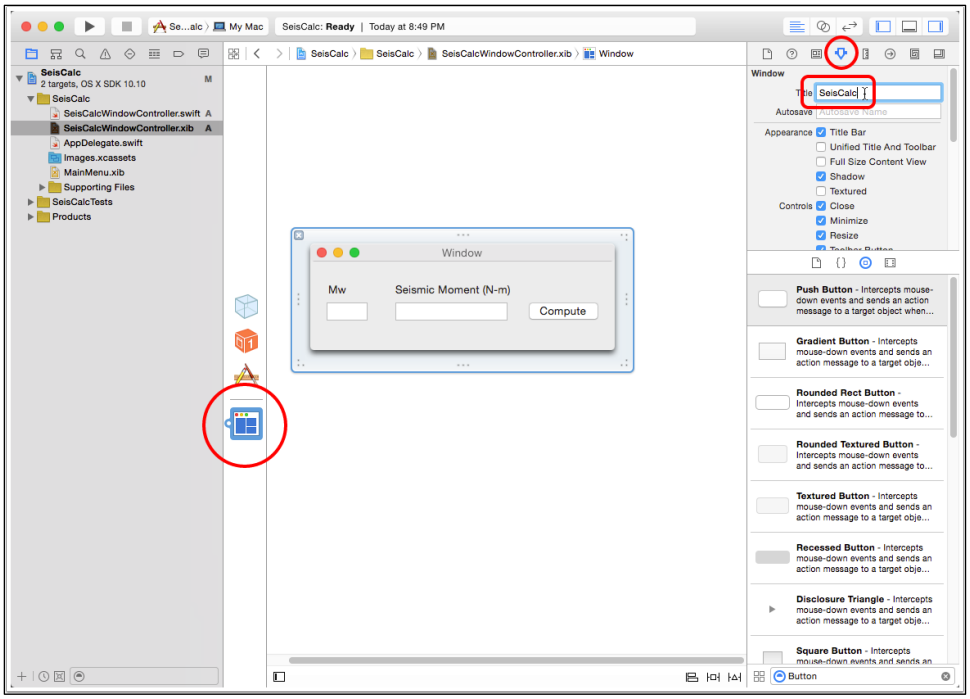


169

170

171 One last thing, change the window title to SeisCalc by selecting the window and
172 editing the Title field in the object properties form (see below):

173

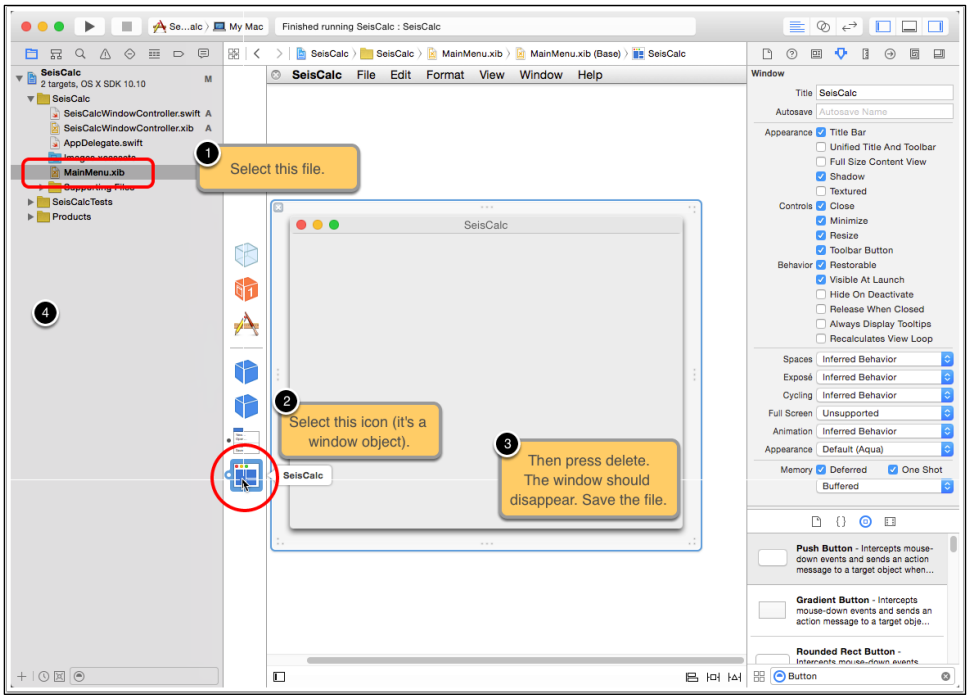


174

175

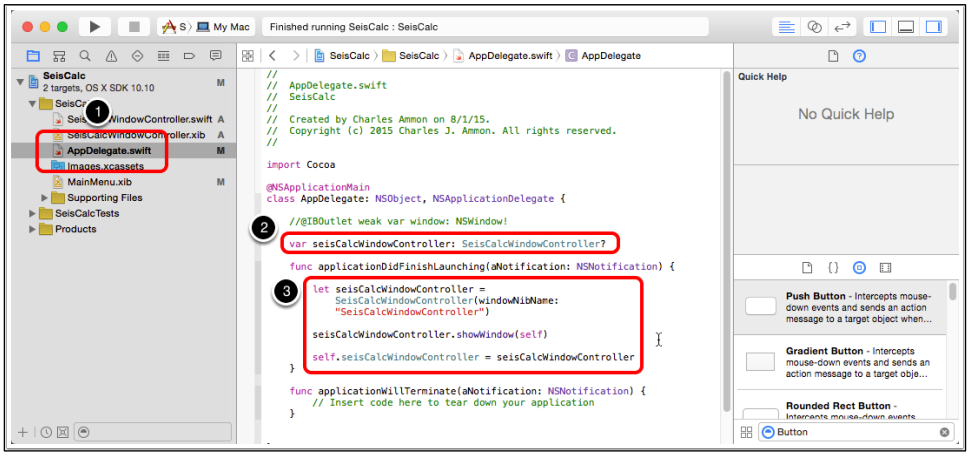
176

By default, the XCode application template is connected to the window object in the MainMenu.xib file. To break that "connection", single-click on the MainMenu.xib item in the file list, then select the window object in that xib file, and delete it (press delete), see below.



Now that we've disconnected the default window, we have to connect the AppDelegate to our window, which is in the SeisCalcWindow.xib file. We do that in the AppDelegate.swift code.

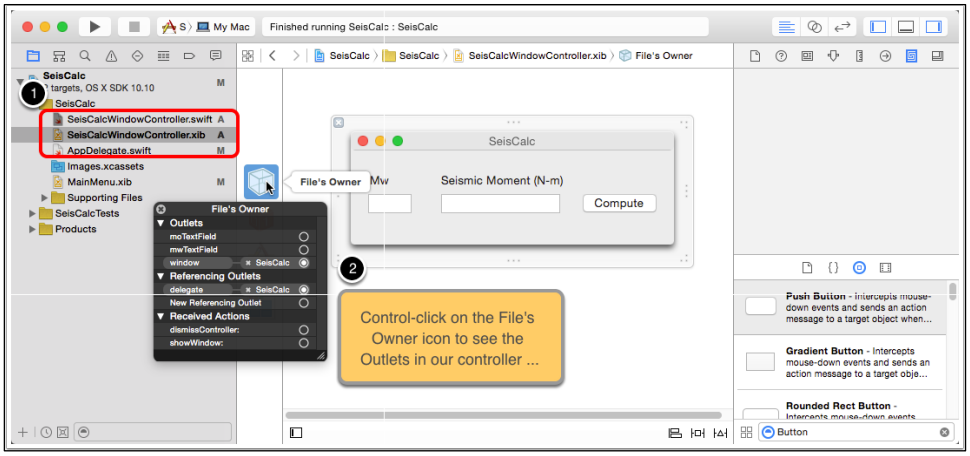
Select the AppDelegate.swift file and edit to code to look like that below. In four lines of swift, we create a seisCalcWindowController object & have it load our xib.



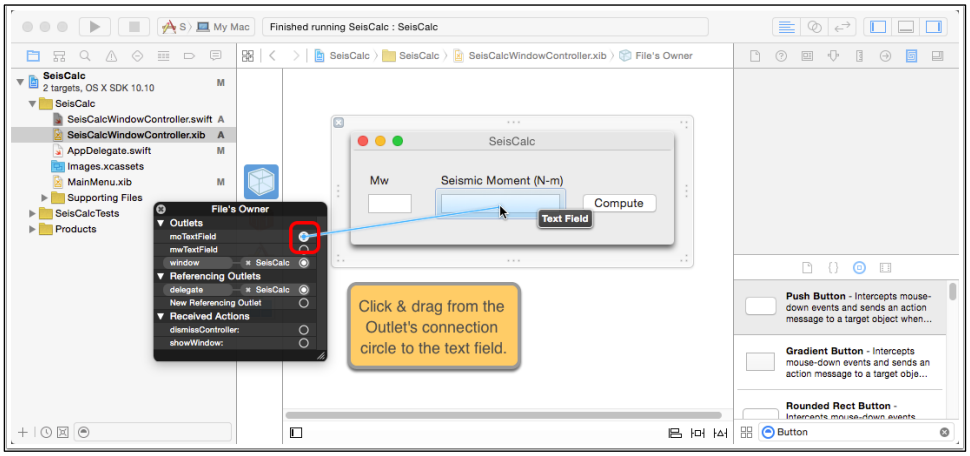
You can build and run the code and you should see our window appear. However, none of our interface elements are connected to the swift code. To produce the working tool, we create variables to hold the values of the text fields and a function to be executed when the button is pushed. Then we connect the code with the interface using XCode's interface builder tools.

We'll start by connecting the text fields. Select SeisCalcWindowController.swift file and add the two lines of code highlighted below. These lines create variables to point at the two text fields we created in our interface to store Mw and Seismic Moment.

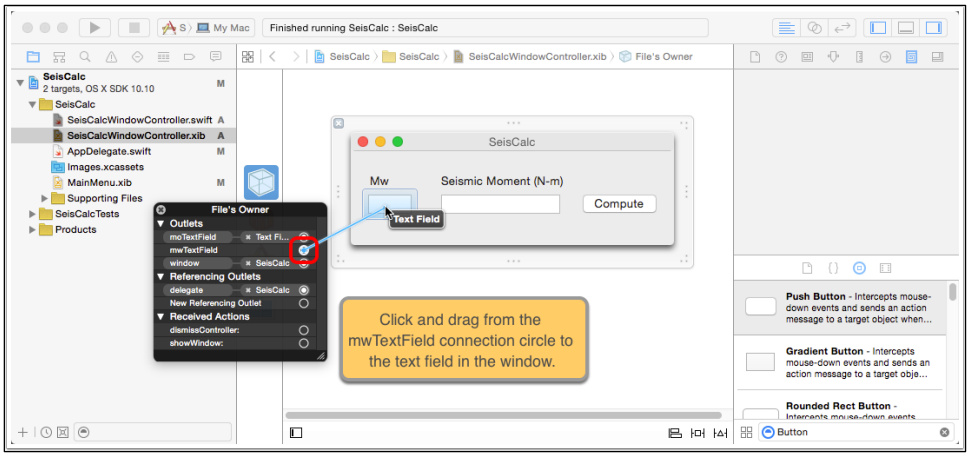
Now we make the connections in the xib file interactively using XCode. Select SeisCalcWindowController.xib and then ctrl-click on the File's Owner icon. The File's Owner represents our SeisCalcWindowController (it owns this xib file). You should see the two IBOutlets that we just implemented in the swift code. Each outlet needs to be connected to the corresponding text field in the interface.



To make a connection, click and drag from the circle next to outlet in the listing of the File's Owner outlets (see below) to the corresponding text field. When the text field is highlighted release the mouse to make the connection.

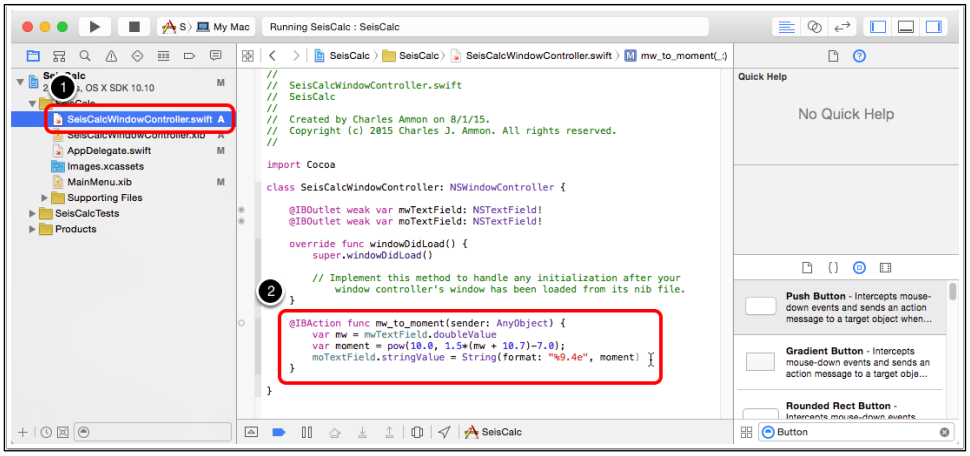


Repeat the procedure for the other TextField.



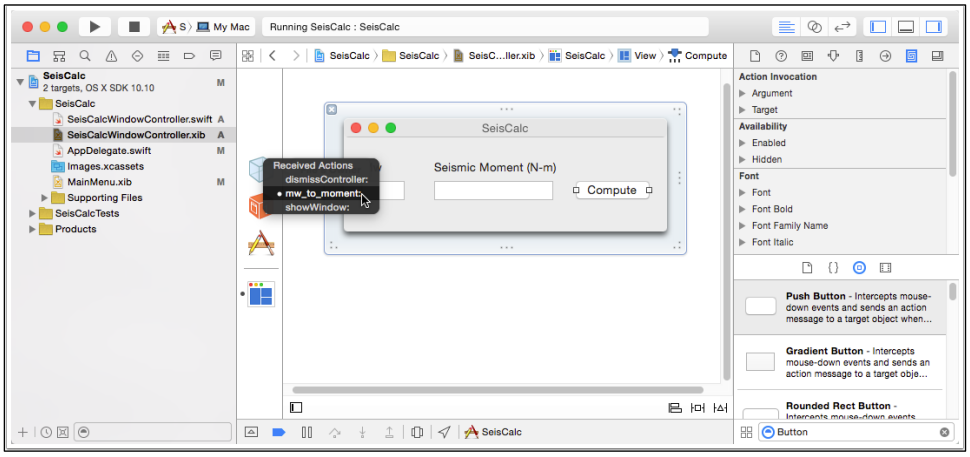
Now we can read and write strings from and to the text fields. The last thing we need to set up is a function that computes the seismic moment from the Mw value when the button is clicked by the user. We do that in the Swift source code.

Go back to the SeisCalcWindowController.swift source code and add the function as shown below.



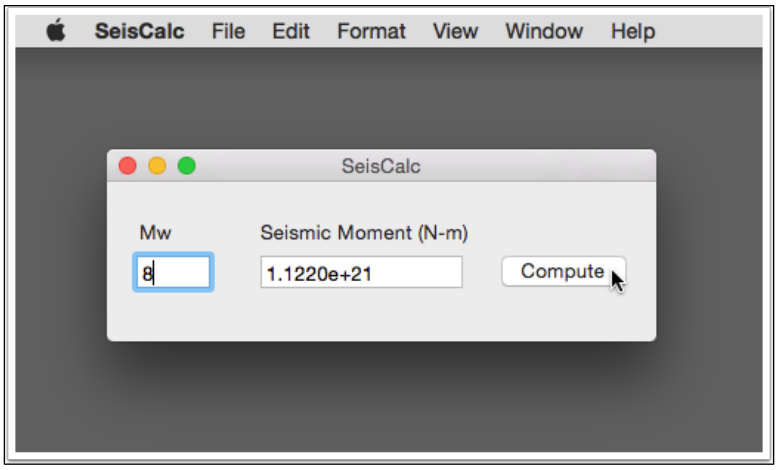
You just created an action that extracts the value of the text field called mwTextField, and then uses that value (Mw) to compute the seismic moment. Then the code sets the string in the momentTextField equal to the result of the calculation.

Save the Swift file and return to the SeisCalcWindowController.xib file. Ctrl-click and drag from the “Compute” button to the “File’s Owner”, icon which represents the SeisCalcWindowController object. A list of IBActions should popup. Select the function we just added to the Swift file (mw_to_moment).



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.).



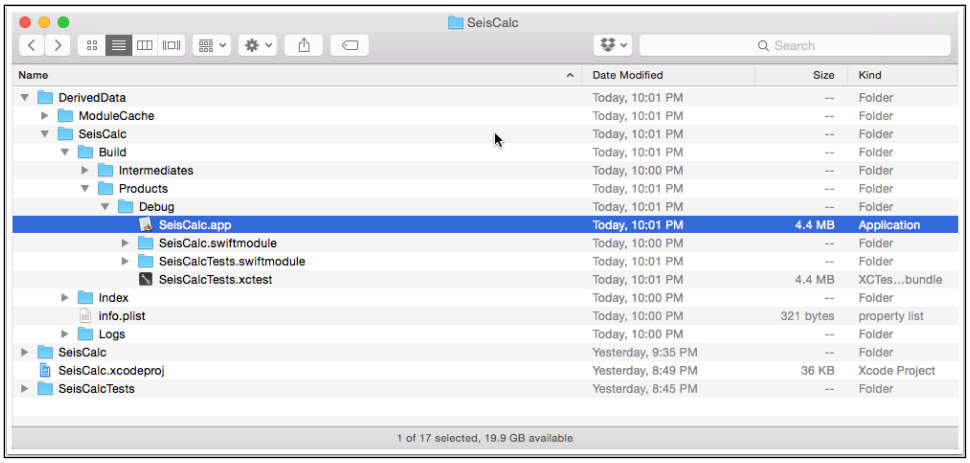
Test the code. If the application did not launch, check the error and warning messages and fix any typos that may be causing problems. If the application compiles and launches but is not functioning, check you connections in xib file.

Make a Snapshot

Once you have a working app, you can use the built-in source version control to save a snapshot of the code (the equivalent of a "commit" in git). Select "Create a Snapshot" from the file menu, provide a snapshot title and description, and click OK. Now you can experiment with the code, without worry. You can always revert to this point in time using that snapshot.

Where is the Application?

One of the most frustrating things for new XCode users is figuring out where it put the results of compilation. When you complete your application, you can find the app in the build subfolder of the project folder (this is why we set the preference to "Relative" at the beginning of the activity).



Learning More

We wrote eleven 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. This example is a modification of an earlier one that I created for a previous workshop. It closely resembles the first exercise in the book “Cocoa Programming for OS X, The Big Nerd Ranch Guide” by Hilegass, Preble, and Chandler: