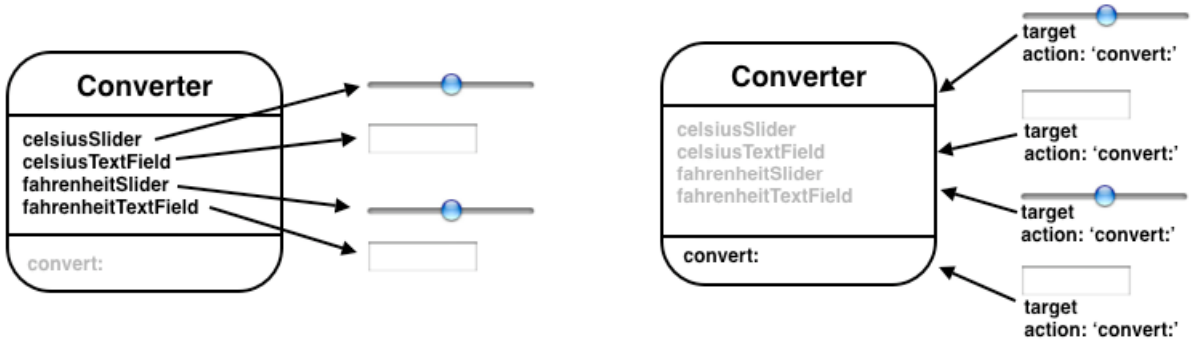
CS193E: Temperature Converter Walkthrough

The goal of this 'walkthrough' is to give you a fairly step by step path through building a simple Cocoa application. You are encouraged to follow the walkthrough, as it should help you successfully complete the assignment, help you quickly become familiar with the tools and basic Cocoa development workflow.

Also keep in mind, that although you will be navigating a series of steps in tools, what you are building is the connected group of objects diagrammed below. You will define a converter object with instance variables (*outlets*) that refer to the controls. The converter object implements behavior, an *action method*, that does the conversion. You will connect each control so that the target of the control is the converter, and the action that will be sent is 'convert:'

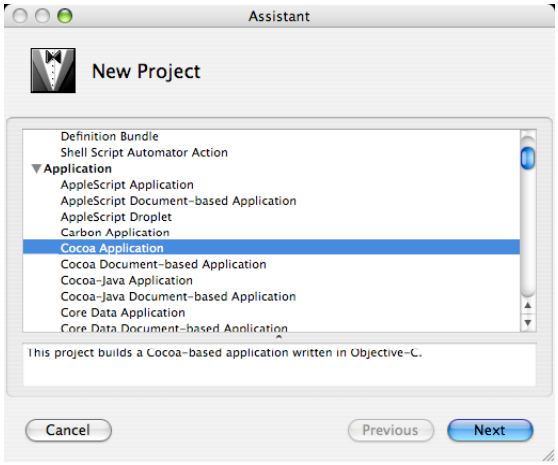


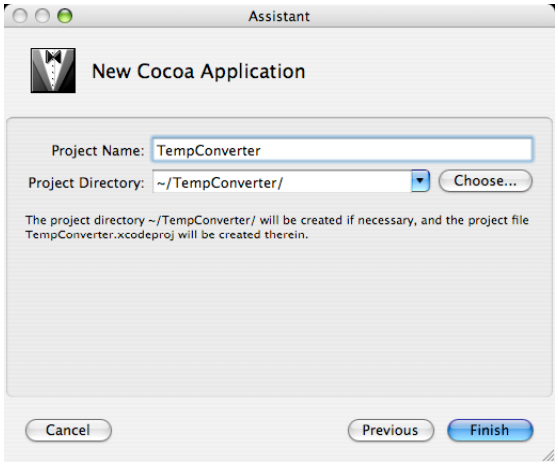
Launching

Launch Xcode. It is located in /Developer/Applications.
On first launch of Xcode, you will be asked a series of questions. The default settings are all fine.

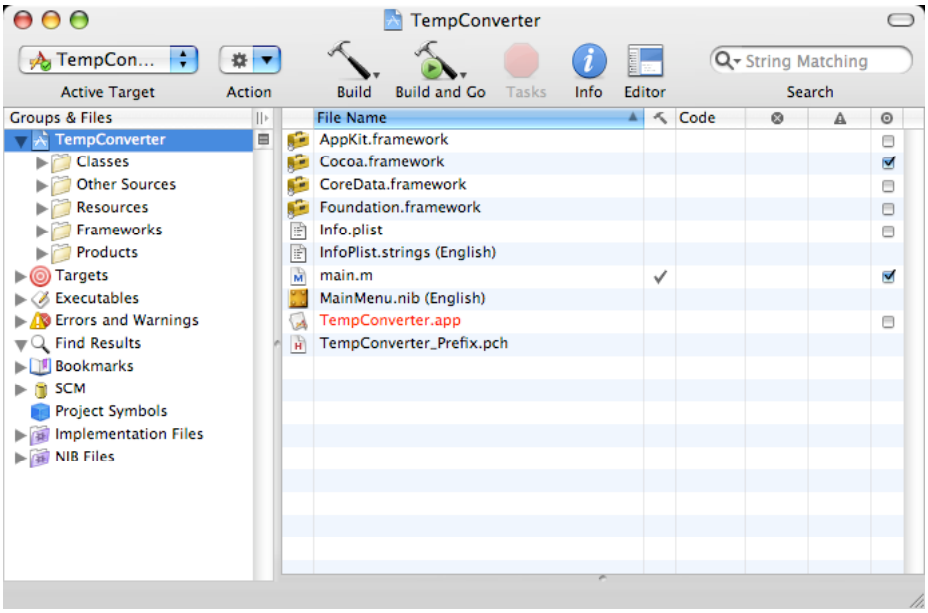
Create a new Cocoa Application project:

1. Select menu **File > New Project…**
2. Select **Cocoa Application**
3. Name the project 'TempConverter'





Xcode is the native Integrated Development Environment for Mac OS X. Each project contains source files, resources such as Interface Builder nib files and images, as well as settings that define how your project is built.



Checkpoint: Build and Test
The project is already set up with the skeleton of a Cocoa application. Convince yourself and double check by clicking the "Build and Go" button in the toolbar. This will build the application and then run it.

Note that a default set of menus and an empty window is already present. When you are done poking around, note that the Quit menu item and keyboard shortcut is already functional.

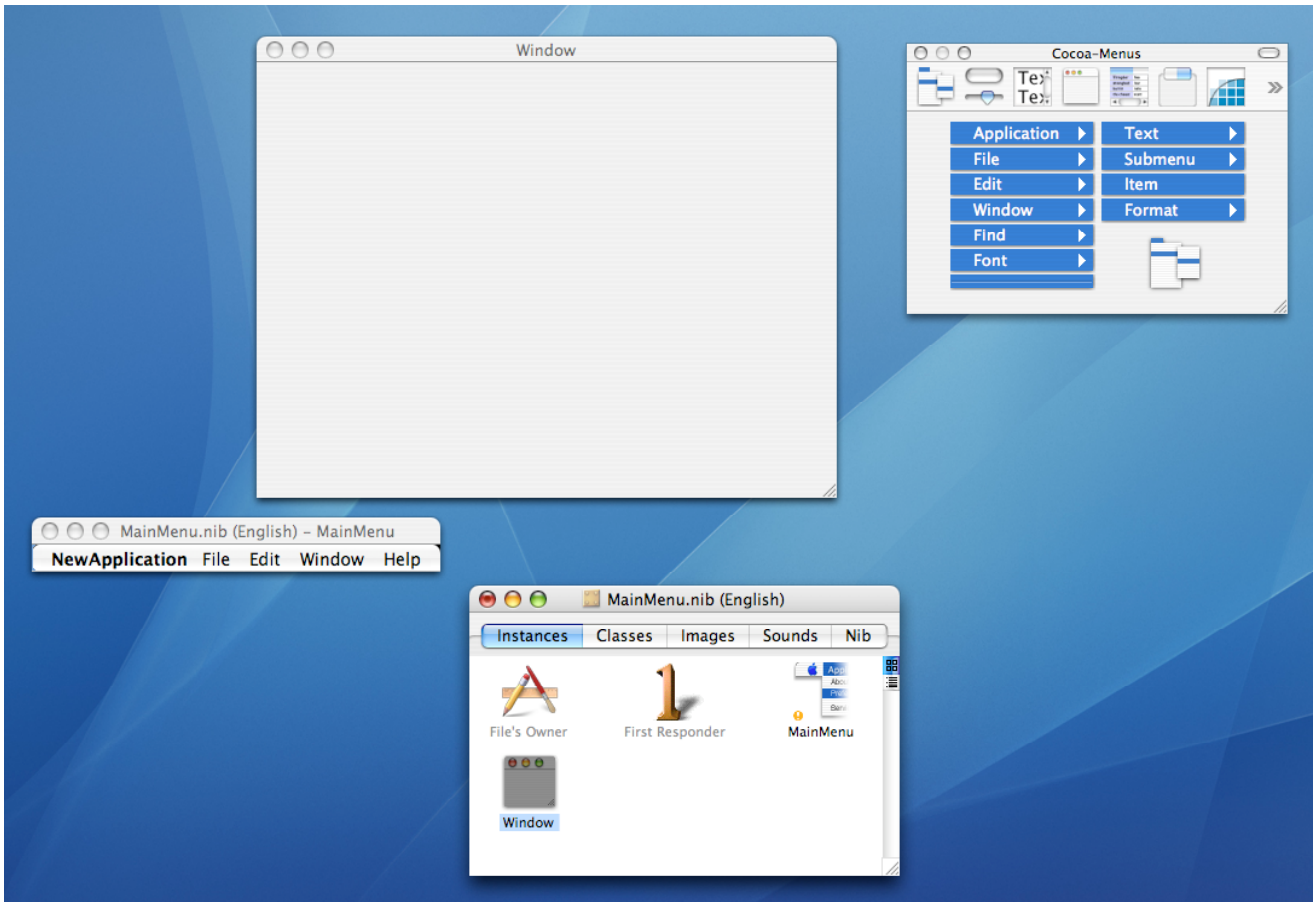
Troubleshooting: If you do not reach this point, and your project does not look like the project picture above, it's likely you accidentally chose the wrong project type – just start again.

Working in Interface Builder

The menus and window that you saw when you built and ran the bare-bones Cocoa application are defined in the MainMenu.nib file. A nib file contains archived objects that you configure and connect in Interface Builder. By convention, a Cocoa application loads a nib named "MainMenu.nib" at launch time. This nib file contains the main menu of the application and potentially other user interface element. When a nib is loaded, all the objects defined within it are unarchived into live object instances.

Open the file MainMenu.nib by double clicking. This will launch Interface Builder. (Interface Builder is located in /Developer/Applications).

Tip: You can select Hide Others from the Interface Builder menu to focus on just Interface Builder.

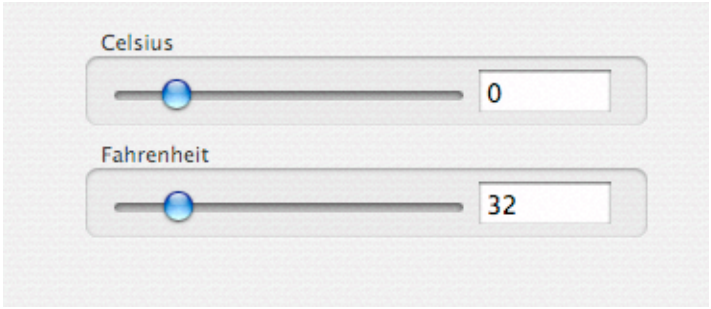


In Interface Builder we will be doing four main tasks:

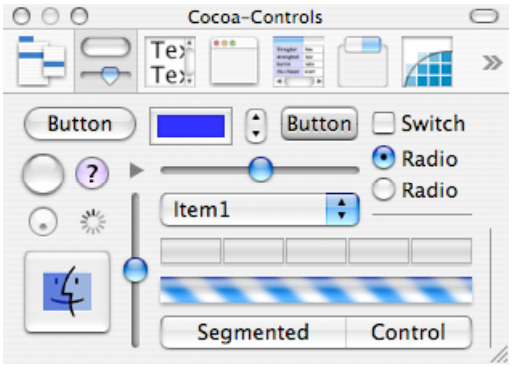
1. Layout and configure the user interface
2. Define a controller class and create an instance of that class
3. Connect the user interface and controller
4. Generate source files for the new class

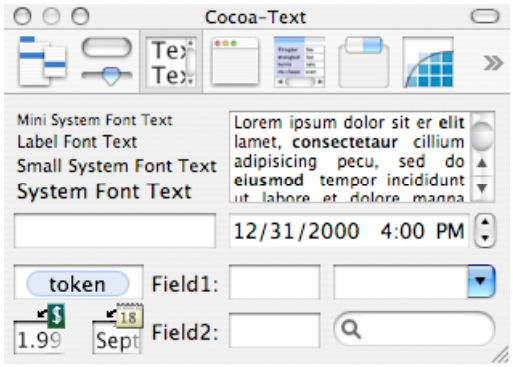
Layout and configure the user interface

Following the steps below should give you a user interface that looks similar to:



The palette window contains different palettes of user interface objects you can drag to the window.



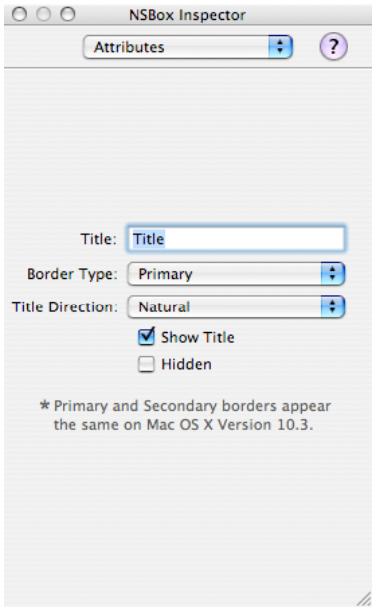


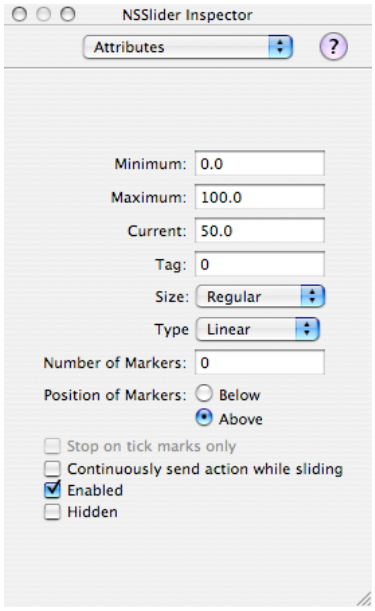
1. Drag a slider from the Cocoa-Controls palette to the window
2. Switch to the Cocoa-Text palette
3. Drag a text field to the window

To enclose both items in a box:
1. Select both items
2. From the menu choose **Layout > Make subviews of… > Box**

To make a copy of the box and its controls for the other temperature:
1. Option-drag the box, or select and use copy/paste to make a second copy

The Inspector
Show the Inspector panel by selecting Tools > Show Inspector. You use the inspector to view and edit properties of the selected item.





You can edit the title of the two boxes by changing the title in the Inspector, or by double clicking directly on the Title.

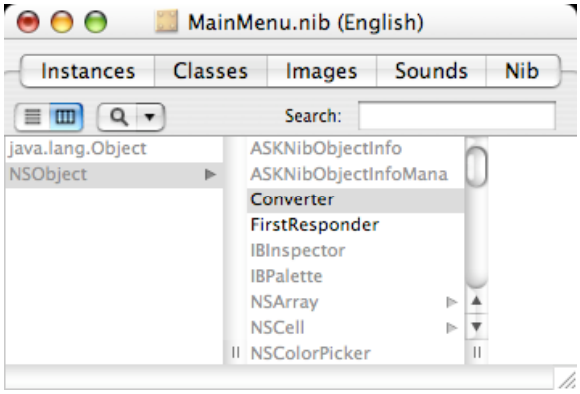
Set the minimum and maximum values of the sliders:
1. From the menu select Tools > Show Inspector if it is not already showing.
2. Select a slider
3. Use the text fields to set the minimum and maximum values.

Celsius should range between at least 0 and 100, Fahrenheit should range from between at least 32 and 212.

Save your changes.

Defining and instantiating a class in Interface Builder
The interface for the application is in place. We are now going to define a class called Converter that has instance variables that will reference the two sliders and the two text fields. Each reference is known as an *outlet*. The class will also define a method that will be called by the controls known as an *action*.

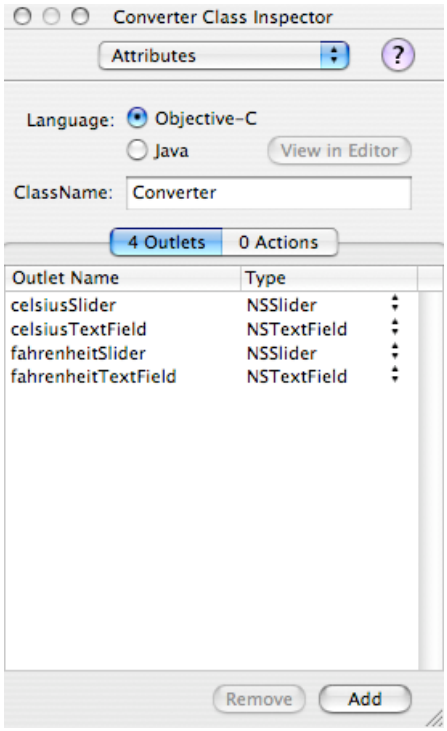
1. Switch to the Classes tab in the Interface Builder document
2. Select NSObject
3. To define a subclass, select the **Classes > Subclass NSObject** menu item
4. Name the new class 'Converter'.

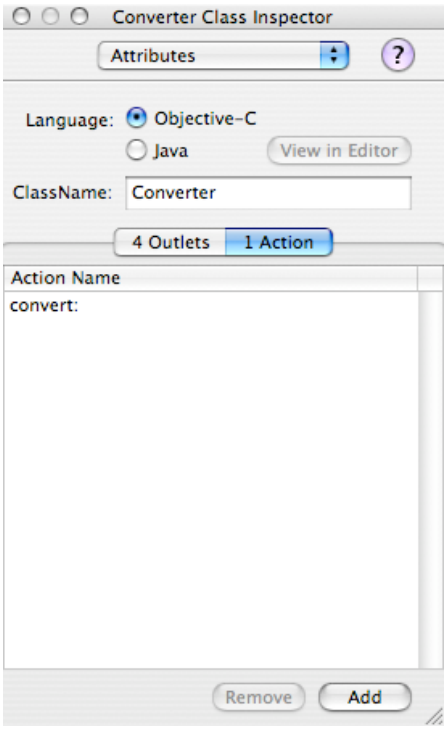


To add outlets:
1. Keep 'Converter' selected
2. Use the **Classes > Add Outlet To Converter** menu item
(You could also use the Add button in the inspector)

When you add an outlet, you also specify the type of object. Add the following outlets:

Outlet Name	**Type**
celsiusSlider	NSSlider
celsiusTextField	NSTextField
fahrenheitSlider	NSSlider
fahrenheitTextField	NSTextField



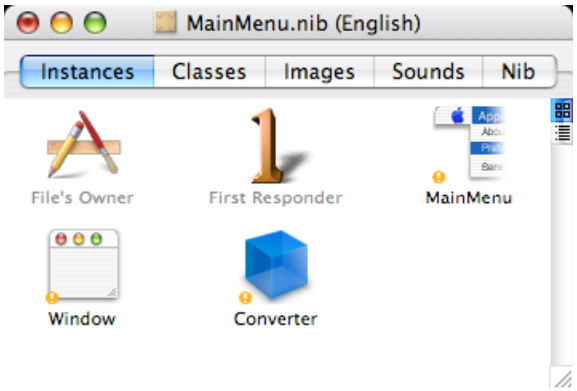


Next add an action:
1. Choose **Classes > Add Action To Converter** from the menu
(or switch to the Actions tab in the Inspector and click the Add button).

Name the action 'convert:' Note the colon at the end of the action, it will be automatically added if you do not type it yourself.

Create an instance of the new class
To create a new instance:
1. Select the class and choose **Classes > Instantiate Converter** from the menu.

The new instance appears in the document window as a blue cube.

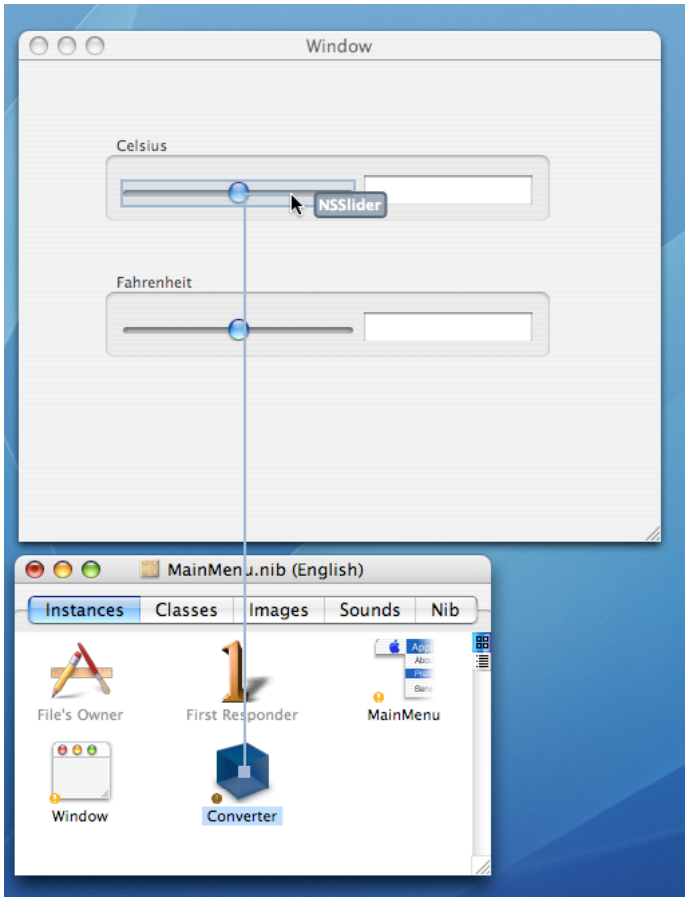


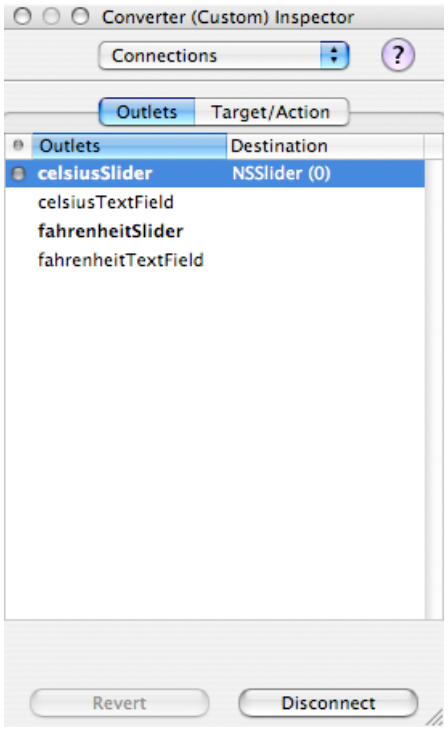
Connect the user interface and controller

The Converter class is now defined and we have an instance, now we need to establish connections between the converter and the user interface controls.

Connecting the Outlets

First, connect the outlets. An outlet is a reference from one object to another.
Establish a connection from the converter to the celsius slider by:
1. Control-drag from the Converter to the celsius slider. (A blue line shows the connection)
2. When the destination of the drag is highlighted, release the mouse.
3. In the inspector, choose celsiusSlider and click the Connect button
(or just double click on celsiusSlider).
4. The small dot and the value in the Destination column shows that the connection is now made.





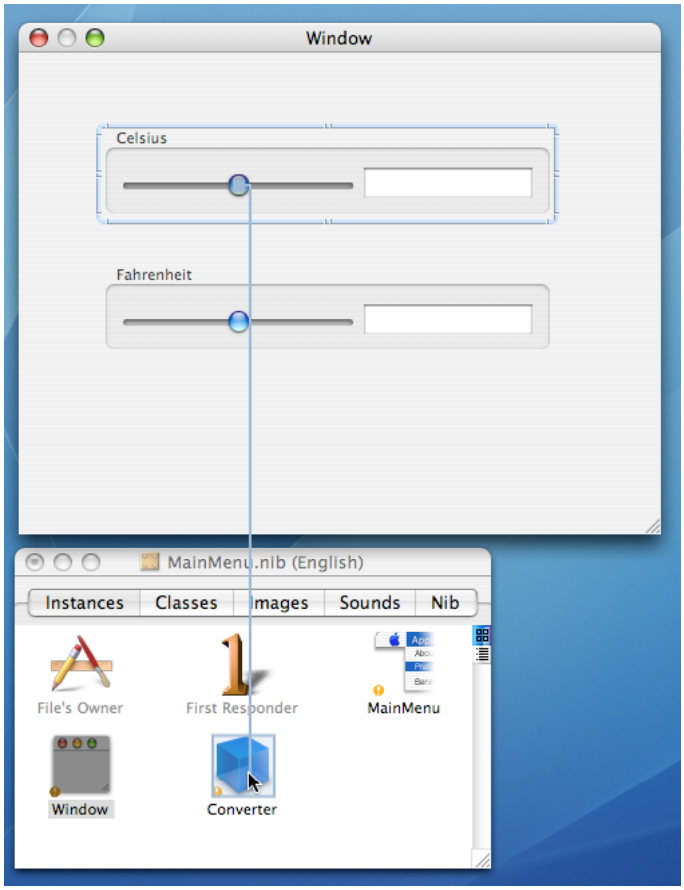
Repeat for the remaining three outlets dragging from the controller to the appropriate control. At runtime, the converter object will have references to these controls, and will be able to send messages to them, in our case, to get and set their values.

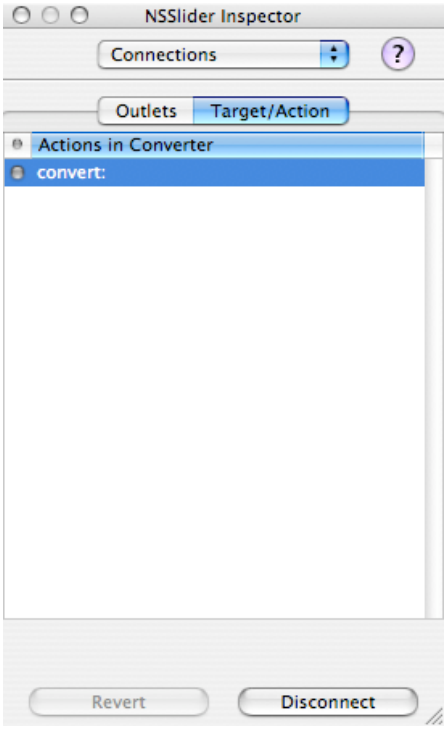
Save your changes.

Connecting the controls to a target and action
Each control is able to store a reference to an object - known as the control's *target* – and the method to be called on the target, known as the *action*.

To set the target of a control, and select the action to be sent:
1. Control-drag from the celsius slider to the Converter object.
2. Once the destination is highlighted, release.
3. In the inspector, in the Target/Action tab, select the action 'convert:' and click the Connect button (or just double-click the action name).
4. The dot indicates that the connection has been made.

Set the target and action of the three remaining controls in the same way.

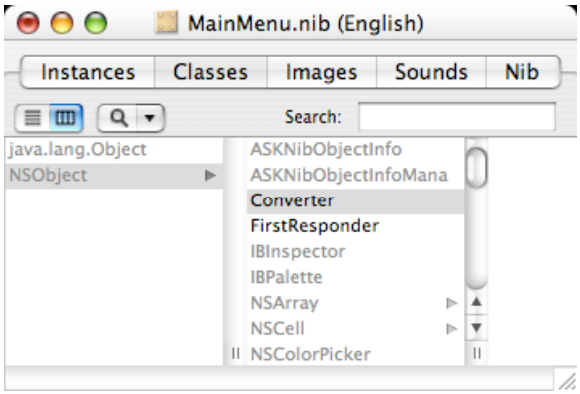




Save your changes.

Generate source files for the new class

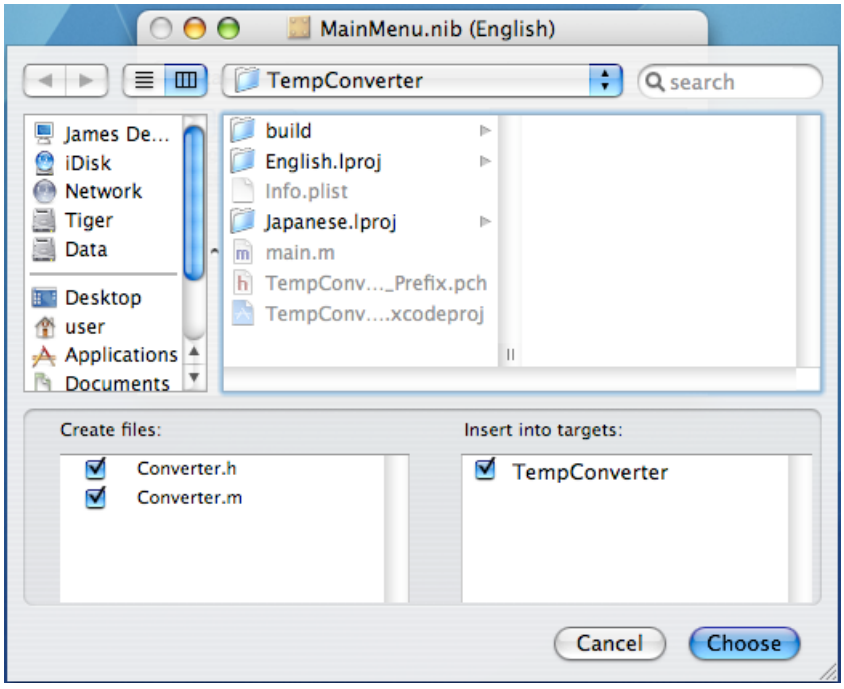
The Converter class is defined in Interface Builder, but since it is a class we just created, there is currently no Objective-C source code that defines the class. At the moment we are in a state where at runtime when the nib file is loaded, there will be a runtime error because an object of type Controller will want to be created, yet there is no compiled code that defines that class.



To generate the source files:
1. Switch to the Classes tab in the Interface Builder document.
2. Select Converter – it is a subclass of NSObject
3. Select the menu item Classes > Create Files For Converter
4. A Save sheet will appear.

Notice the Save sheet will create a header file and an Obj-C implementation file for the class, it is saving the files in the project folder, and inserting them into the project (in Xcode a set of files to be compiled and processed to generate some built result is known as a *target*).

5. Click Choose to accept the default settings.



In a moment, you will switch over to Xcode to view and edit the newly created file.

Save changes to the nib file.

Using Xcode

Switch to Xcode.

You will notice the Controller.h and Controller.m classes are now in your project. First take a look at the header file. Double-click Controller.h to open it in another window, or click the Editor button in the toolbar, a single click then will show the selected file in the editor in the main project window.

Header file
The Converter.h file should look something like:

```
/* Converter */

#import <Cocoa/Cocoa.h>

@interface Converter : NSObject
{
    IBOutlet NSSlider *celsiusSlider;
    IBOutlet NSTextField *celsiusTextField;
    IBOutlet NSSlider *fahrenheitSlider;
    IBOutlet NSTextField *fahrenheitTextField;
}
- (IBAction)convert:(id)sender;
@end
```

Objective-C can look quite foreign to someone viewing it for the first time, even if familiar with C, C++, or Java. The next assignment will get into the details of Objective-C, for now just look it over to get the gist.

The code defines the interface for a class Converter which is a subclass of NSObject.

Within braces you see the outlets defined in Interface Builder, these are instance variables – and each is literally a pointer to an object (Note the NSSlider *). At runtime, when the nib is loaded all of the connections you made in Interface Builder will be hooked up, so you can message these objects.

You also see the declaration of the convert: method. Notice it takes one argument, the object that sent the message (more on that for a moment). Also note that you can think of the type *id* as meaning *any object type*.

So, every outlet and action defined in Interface Builder is represented here in code.

Implementation File

Next, open the Converter.m file

Tip: You can jump between the header and implementation file using the Go To Counterparts  button in the directly above the text. You can change the behavior of this to switch between the two in the same editor by adjusting **Xcode > Preferences… > General > Open counterparts in same editor.**

```
#import "Converter.h"

@implementation Converter

- (IBAction)convert:(id)sender
{
	// Implementation goes here
}

@end
```

The .m file contains a stub for the implementation of the convert: action method. In Interface Builder each control was set to have the converter object as its target and convert: as its action. Whenever a slider is moved or a text field is edited, it will also send a convert: message to the converter.

Checkpoint: Log, Build, and Test
Before you write the guts of the convert: method, you may want to put in a simple logging statement, then build and test your application. If you've hooked up all the actions correctly, moving the sliders or changing text field values should display the log message in the console.

A simple log statement in Cocoa:

```
NSLog(@"I'm in the convert method");
```

To view the console in Xcode, select **Debug > Console Log**.

A little Objective-C

Future lectures will detail Objective-C, but here's enough to get you going. To implement the convert: method you'll be sending messages to the various controls.

To get the int value of a control:

```
int celsiusValue = [celsiusSlider intValue];
```

To set the int value of a control:

```
int anInt;
[celsiusTextField setIntValue: anInt];
```

You can determine if two objects are the same instance by testing for pointer equality. For example, to see if the incoming 'sender' value is the same object as the celsius slider:

```
if (sender == celsiusSlider) {
  // Do something here
}
```

To determine if two objects are logically equal, use the –isEqual: method.

```
if ([sender isEqual: celsiusSlider]) {
      // Do something here
}
```

For purposes of this assignment, either is appropriate.

Hints for implementing the convert: method

The Converter object is your controller: it should do the work to get values from the inputs and set the output fields appropriately. (Note that in this case, we don't really have an explicit model.)

To convert Celsius into Fahrenheit: multiply by 9, divide by 5, and add 32.

You now should have enough information now to implement the convert: method

Unrelated Tip: A handy way to convert mentally from Celsius to Fahrenheit (and impress your friends): double the temperature, subtract 10%, and add 32.

Build and Test

Once you have implemented the convert: method, build in Xcode, and run the application.