Objectives

- Register for an Apple account
- Create a simple application using Xcode
- Test your application with the iPhone simulator
- Import certificates for development
- Test your application to a device
- Expand your application
- Show your finished app to the TA

Submission

Complete the online lab feedback form and show the TA your work when indicated.

Register for an Apple account

The iOS development center requires that you register before you can access documentation for the iOS platform. This process is free, but it may take some time for you to receive the confirmation e-mail. You DO need to register for an Apple account to access documentation related to work in later labs.

You can register for a free Apple ID account by going to:
http://developer.apple.com/programs/register/

Notes on Saving Projects

Projects saved on the Mac Mini's hard drive will stay with that Mac Mini (this includes the Desktop).
Your network share should be available as an icon on the dock on the bottom of the screen towards the right.

Xcode Keyboard Shortcuts

Auto-complete Feature – Press the Esc key

Jump to Definition – Right-click (or Ctrl-click) a keyword and select *Jump to Definition*.

Jump to Definition – (Alternative) Hold the Command key (⌘) and double click.

View Documentation - Right-click (Ctrl-click) on a class and select *Find Text in Documentation*.

Build your project – Command (⌘) + b
Save the current file – Command (⌘) + s
Quit the current program – Command (⌘) + q

Jump to beginning of a line – Command + Left Arrow
Jump to end of a line – Command + Right Arrow
Jump to beginning of current word – Option + Left Arrow
Jump to end of current word – Option + Right Arrow
Jump to beginning of all text – Command + Up Arrow
Jump to end of all text – Command + Down Arrow
* You may select text by adding the shift key to the text navigation commands above

Create a Simple Application Using Xcode

Xcode is an IDE (Integrated Development Environment) for developing software on Mac OS X and Apple's mobile platforms, like the iPhone, iPod Touch, and iPad. The mobile operating system is called iOS. The Xcode development environment offers a wealth of tools and features.

To begin developing an application you will need to launch Xcode and start a new project.

Step 1) Start Xcode. You can either locate it in Finder (Mac's version of Windows Explorer) or use Spotlight. If using Finder, Xcode is located under the *Macintosh HD/Developers/Application* directory. To find Xcode using Spotlight, simply search for "Xcode" using the magnifying glass in the top-right corner of the desktop (⌘+Space).

Step 2) Select *Create a new Xcode project* from the *Welcome to Xcode* dialog box.

Step 3) For your first app, choose the *Single View Application* template.

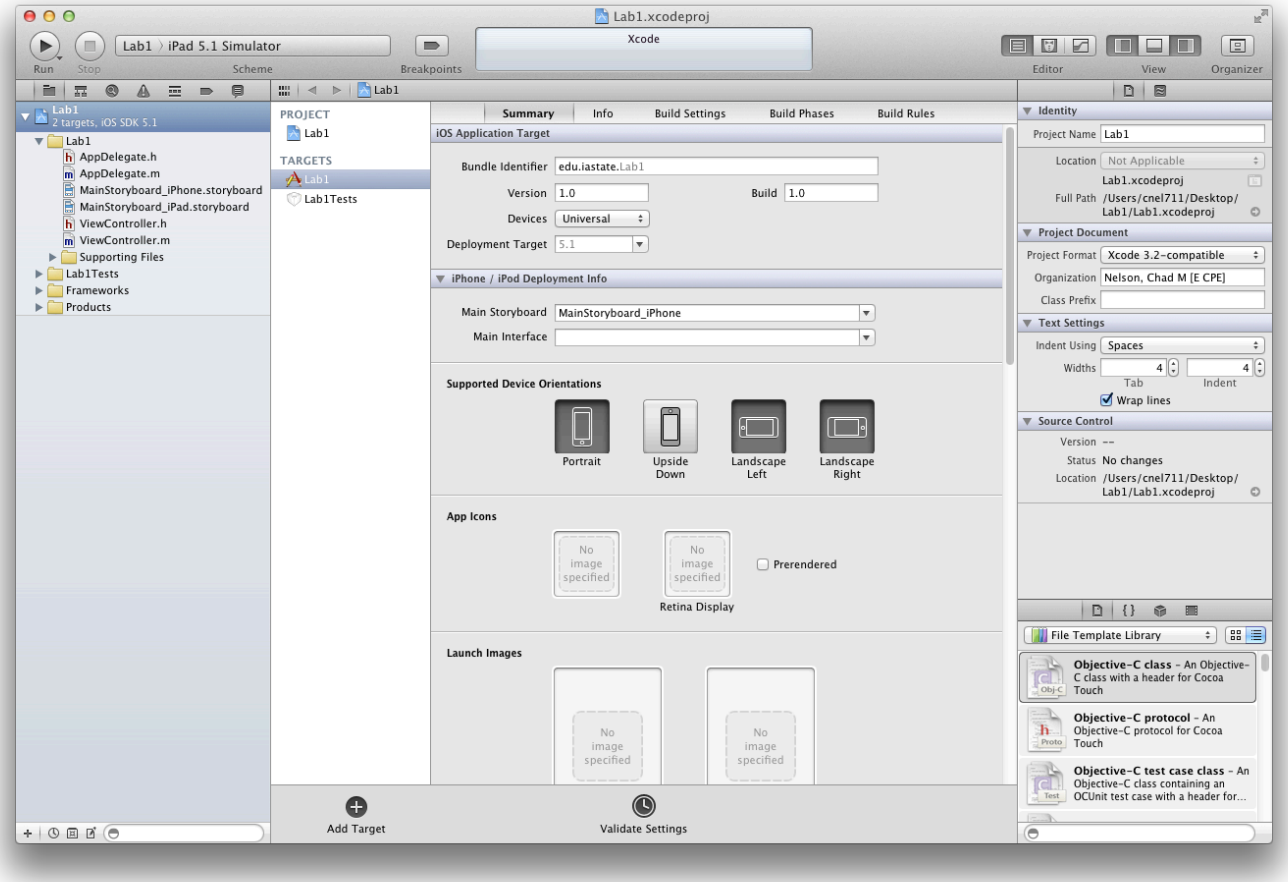
> **Note:** There are a number of other options here, and you are encouraged to explore the different project templates, but for now use the view-based application template.

Step 4) Give the project a *Product Name* (Lab1) and *Bundle Identifier* (edu.iastate). If you have a device running iOS 5 or greater, you may use Storyboarding and Auto Reference Counting (ARC); otherwise, toggle them off. Click Next.

> **Note:** For this lab, do not use Storyboarding or Auto Reference Counting (ARC). Apps that use Storyboarding will only work with devices running iOS 5 or greater. Auto Reference Counting will work with devices running iOS 5 or greater, and will work for devices running iOS 4.0 or greater but will not set weak references to nil (ARClite).

Step 5) Select a Location for your project (Desktop). Click *Create*.

You have successfully created a view-based application project. If you did everything successfully, you should see a window similar to this:



This is the main window in which you will do most of your coding during future labs.

Step 5) Browse the files of the project. A class and its public methods are defined in .h files, and the methods are implemented in .m files. Notice the "Supporting Files" folder. The entry point of the program is in the main.m file. The .plist file contains settings for your project, including things like the Bundle Identifier. If you did not choose Storyboarding when creating your project, there will be a few .xib files. These are XML based Interface Builder files, and are pronounced "nib" files after their former file extension. If you chose to use Storyboarding when creating your project, the Interface Builder files will have a .storyboard extension.

Test Your Application in the iPhone Simulator

At this point our application can be simulated, though it will not do much since we have not written any code. It will simply be a blank gray application.

Step 1) Select the iPhone Simulator from the drop down at the top of the main window.
Step 2) Click the Run button located at the top left corner of the main window. You should see the simulator appear with a blank application started (image to left).

Import Certificates for Development

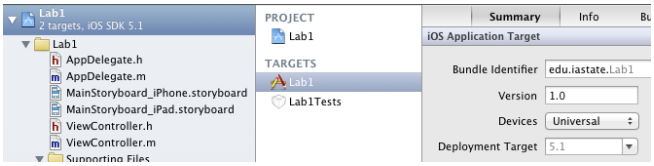
In order to run your application on an iOS device, you must install a mobile provisioning profile to the device and install a developer identity (and private key) into the Apple Keychain. In Xcode 4, much of this process has been automated using the Xcode Organizer.

Step 1) Download the .developerprofile file from BB Learn.
Step 2) Double click the file (*.developerprofile) to add install it. The password is "password".
Note: You will have to repeat steps 1 and 2 if you use a different Mac Mini in future labs.
Step 3) Connect your iPod Touch (or iPhone/iPad). The Xcode organizer should appear. Automatically provision the device in the Xcode organizer. If your device is new, have the TA use his login credentials to provision your device using his Apple ID and password.

Your developer identity and private key are used to sign your application when sent to the device. The Mobile Provisioning Profile allows your device to check the signature of the application and run your program. This is Apple's way of ensuring that only legitimate developers install applications to devices. The Mobile Provisioning Profile only allows certain device UUIDs and certain developer identities to install applications to a mobile device.

Build Your Test Application to the iPod Touch

Note: If you are using an old iPod Touch with a version of iOS lower than iOS 5.1, you need to change the **deployment target** to support the earlier operating system. Select your project at the top of the Project Navigator and you'll see the Deployment Target dropdown on the Summary tab.

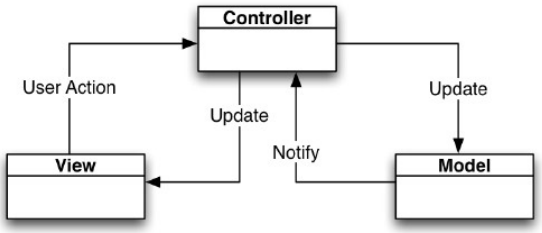


Step 1) Click the drop down box next to the Stop button on the top left of the Xcode main window. Select your device instead of the simulator. Then click the Run button.
Step 2) When the blank application has loaded on your iPod Touch, hold your iPod in the air to show the TA.

Preparing Code for Interface Builder

Interface Builder provides a way to graphically build user interfaces. In this section, we will add a label and a button to our application. We'll write an event handler for our button to change the text of the label when we press the button.

First, let's write some code. Try to gain an understanding of how the connection between views and controllers work. Apple encourages the use of the MVC software design paradigm (Model-View-Controller):



In general, Views are stored in .xib or .storyboard files. The Controllers will be .h and .m files that have Controller in their name (UIViewController, UITableViewController).

The Controller updates the View using IBOutlets.

The View notifies the Controller of user actions using IBActions and delegation. Delegation will be covered in later labs.

Step 1) Edit your *ViewController.h* and *ViewController.m* files by adding and IBOutlet and IBAction:

ViewController.h:

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController {
    IBOutlet UILabel *myLabel;
}

- (IBAction) userClickedButton;

@end
```

ViewController.m:

```
#import "ViewController.h"

@implementation ViewController

- (IBAction) userClickedButton {
    myLabel.text = @"Hello World!";
}

@end
```

Here are some notes about what you just added to your ViewController:

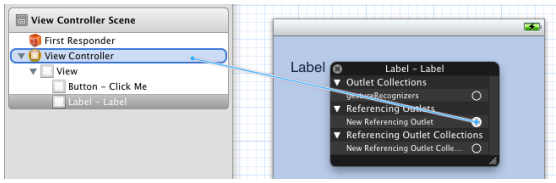
- IBOutlet is a keyword (Interface Builder Outlet). It allows Interface Builder to see that you have defined an object (UILabel) that you want to connect with code. Connecting an object in the Interface Builder (by dragging referencing outlets) allows you to edit the attributes with code of an object that was initialized in the .xib or .storyboard file. If you wanted a UILabel in your View that was static (you don't want to edit its text like we're doing), there would be no need to create an IBOutlet. You create an IBOutlet in order to give your controller the ability to update the view.
- IBAction is a keyword. Interface builder uses it to show functions that interface elements can call. In code, it is equivalent to *void*, and thus myFunction has no return type.

Step 2) Find the iPhone Interface Builder file in the Project Navigator (MainStoryboard_iPhone.storyboard or ViewController_iPhone.xib). Click it once to open it in Interface Builder. Interface Builder is Xcode's WYSIWYG editor (What you see is what you get).

Interface Builder (continued)

Step 3) Locate the Library. The Library is located on the bottom right of the Interface Builder screen and contains all of the UI (User Interface) elements.

Step 4) Add a label from the library onto the view by dragging and dropping it from the Library onto the view. Resize it large enough to fit a reasonable amount of text and position it centered on the view. As shown in the picture, Ctrl-click (right-click) the visual instance of the UILabel you just created on the view, and click and drag a blue connector from the circle labeled *New Referencing Outlet* to File's Owner (if using .xib files) or View Controller (if using .storyboard files), then select the name of your UILabel (myLabel). This connects your UILabel in Interface Builder with the one in your code.



Alternatively, you could make your connections using the Connections Inspector or by right-clicking on File's Owner/View Controller and dragging the Outlet to the Label.

Step 5) Add a Round Rect Button (UIButton), resize, and position it on the view. Ctrl-click (right-click) the visual instance of the button, and click and drag a blue connector from the Touch Down event of the button to File's Owner / View Controller, then select the name of your function (userClickedButton).

Step 6) Save the .xib / .storyboard file from the file menu. (⌘ + S)

Step 7) Test your program in the simulator. When you press the button, the label should read "Hello World!".

Step 8) Return to Interface Builder and change some of the properties of your view, button, and label. This is your chance to see all the available options in the Attributes Inspector window. Try to center your label text, add text to your button, change font colors, change background colors, etc.

Step 9) (Optional) Change your application's icon (select your Project's name at the top of the Project Navigator).

Step 10) (Optional) Add a background image for your app. Import an image (approximately 320x480) for your background into Xcode (right click on the project navigator -> Add Files to "Lab1" …), then add an UIImageView in Interface Builder. Adjust the Image View's size and edit its *Image* property in the Inspector to use your background image. If you expand the left panel in Interface Builder (the panel that contains the File's Owner object), than you can arrange the order of the children on the View (and thus arrange the order in which the children are drawn.

Step 11) (Optional) Creatively expand your project. Implement the same functionality for the iPad.

Step 12) Put your program on the device. Show the TA when complete.

Review Questions:

You should know the answers to the following questions before you leave lab.

1) What is the keyboard shortcut for:
   - Auto completion?
   - Building your project?
   - Saving the currently loaded file?
2) How would you view the header file for the UIViewController class?
3) What are IBActions? How are they used?
4) What are IBOutlets? How are they used?