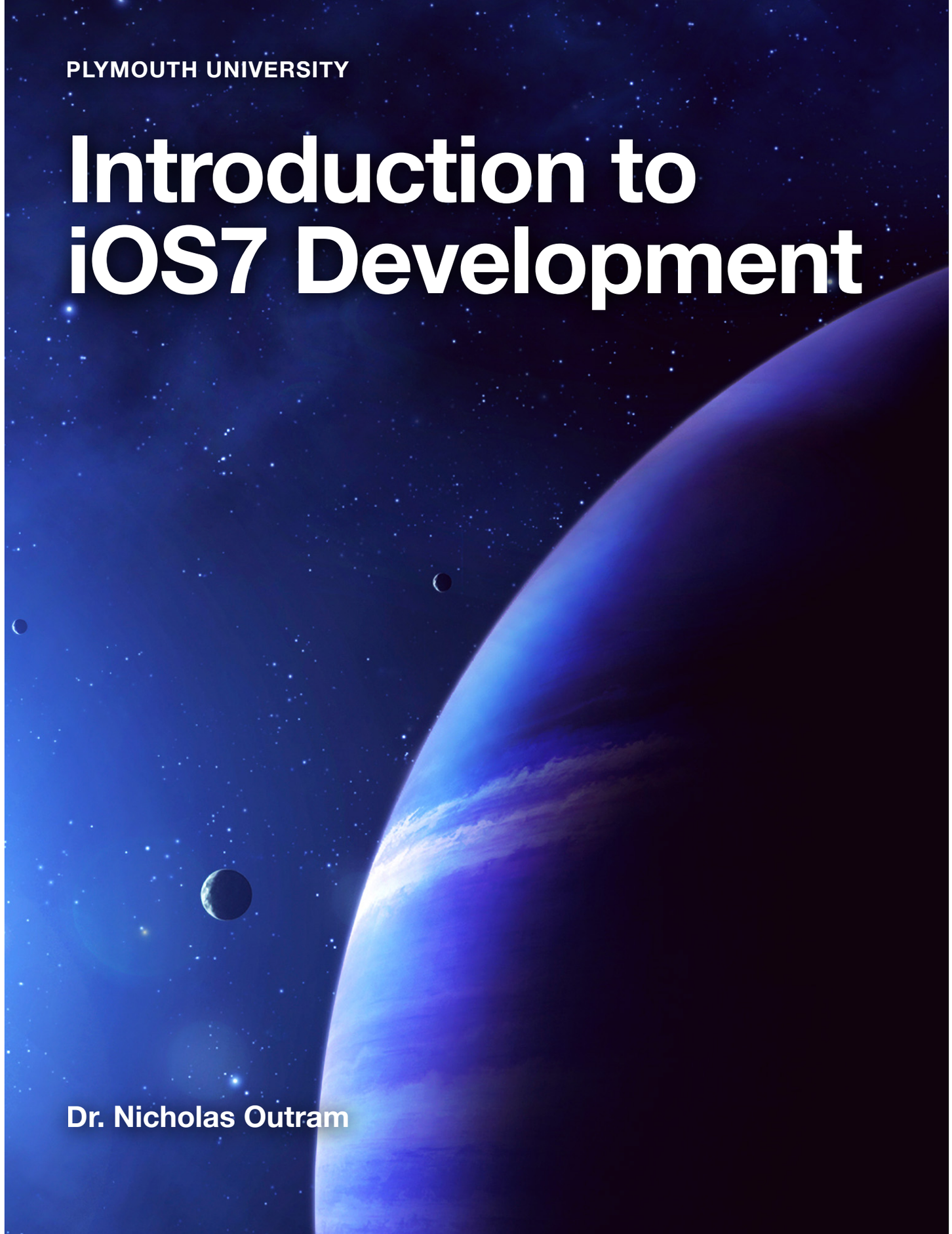
PLYMOUTH UNIVERSITY

Introduction to iOS7 Development



Dr. Nicholas Outram

Section

Task 02 - Detail View

INTENDED LEARNING OUTCOMES

1. **Identify the root view controller in StoryBoard**
2. **Embed a view controller in a NavigationController**
3. **Use a segue to specify navigation between controllers using swipe gestures**
4. **Specify the back button**
5. **Configure a table view controller to display an array of strings using a custom cell**
6. **Configure a table view controller to allow deletion of data**
7. **Configure a table view controller to allow re-ordering of data**
8. **Build a data stack, and use the delegation pattern (with protocols) to initialize detail view and push back changes**
9. **Persist data to data storage using a keyed archive**

One of the most important aspects of mobile app design is the ability to drill down to increasing levels of detail.

We are going to follow the data model stack pattern described in Lecture 1.2

On iPhone, one of the most common methods of displaying detail data is to use a table. It is so common, that there is a dedicated subclass of `UIViewController` called `UITableViewController` which will do most of the tasks for you.

We will build on the task in section 1.

TASK 02-01 ADDING THE TABLE VIEW AND NAVIGATION CONTROLLER

So far, we only have one scene in out storyboard. The top level is managed / owned by the view controller `SoCMViewController`, a subclass of `UIViewController`.

We will not meet two more subclasses of `UIViewController`,

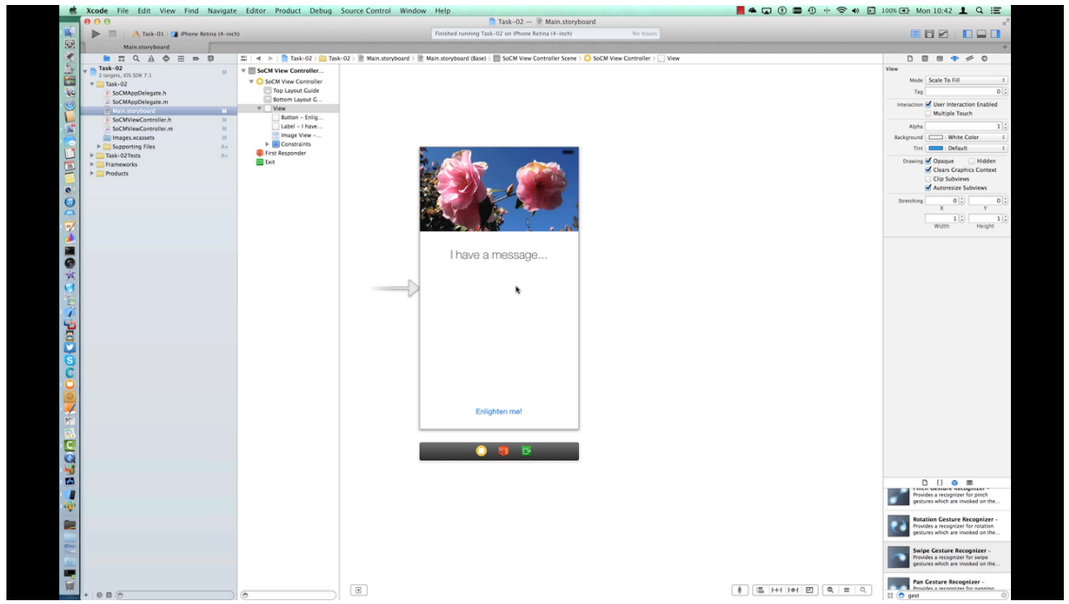
`UINavigationController` - this is a "ViewControllerContainer" and maintains a stack of view controllers and associated views.

`UITableViewController` - a dedicated view controller that managed a scrollable UITableView.

Watch Movie 2.1 and try for yourself:

- add a table view controller
- embed the initial view controller in a navigation controller and switch on the toolbar
- Fix any autolayout issues
- Add gesture recognizer
- Set titles for view controllers
- Hook up segue with name PushTableView
- Change the back button label
- Test

Movie 2.1 Adding the TableViewController and NavigationController



The UINavigationController manages a stack of view controllers and their associated views

TASK 02-02

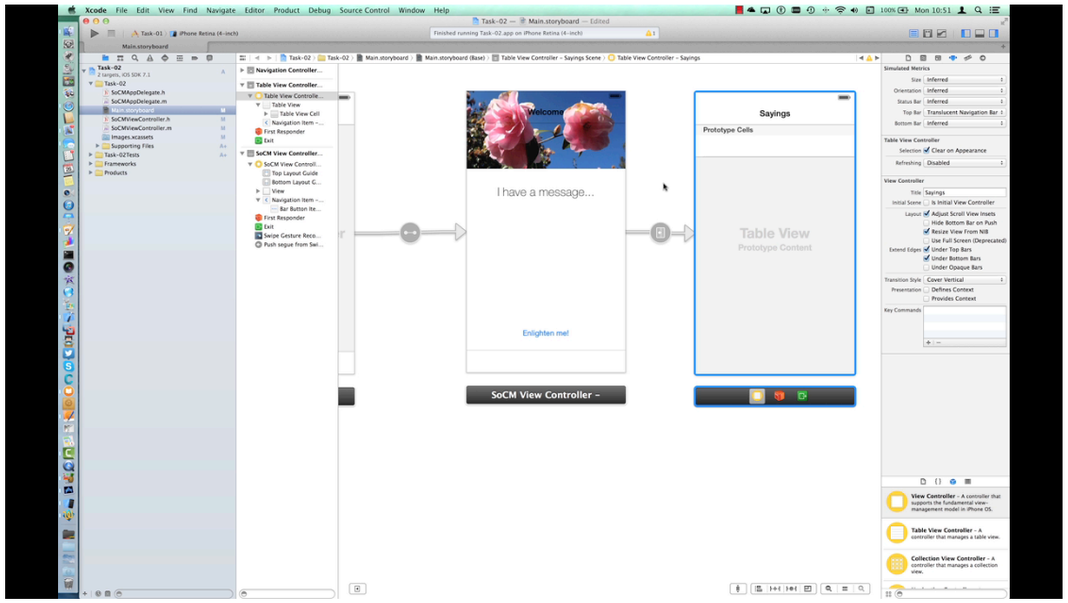
In this task, you are going to write your own UITableViewController subclass and UITableCell subclass.

Watch Movie 2.2 and change your code accordingly.

Summary of the task

- Create custom TableViewController class and set the class attribute in storyboard
- Give the table view controller its own public model. Add property “array” of type NSMutableArray in the header of table view controller
- Add prepareForSegue to the initial view controller, and set the property of the tableviewcontroller to be a mutable copy of the original data
- Address the warnings in the tableview controller
  - Change sections to 1
  - Set number of rows
- Create custom Cell Class with an identifier. Add a label to the prototype cell. Create an outlet.
- Modify TableViewController code to populate the cell

Movie 2.2 Adding a custom table view controller and cell



Before we display any data, we create a subclass of UITableViewController. Remember it is still the controller and plays the role of synchronising data model and view. No data is stored in a TableView as such - the tableview always asks the controller for data as each row becomes visible through a call-back mechanism called delegation.

TASK 02-03

In this task, you are going add the ability to delete records from your model using a standard left-swipe gesture.

Note that two things need to happen when we delete a record:

We delete the object from the array (our copy of the model)

```
[self.array removeObjectAtIndex:indexPath.row];
```

We delete the row from the table by uncommenting code which includes the following:

```
[tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
```

Notes

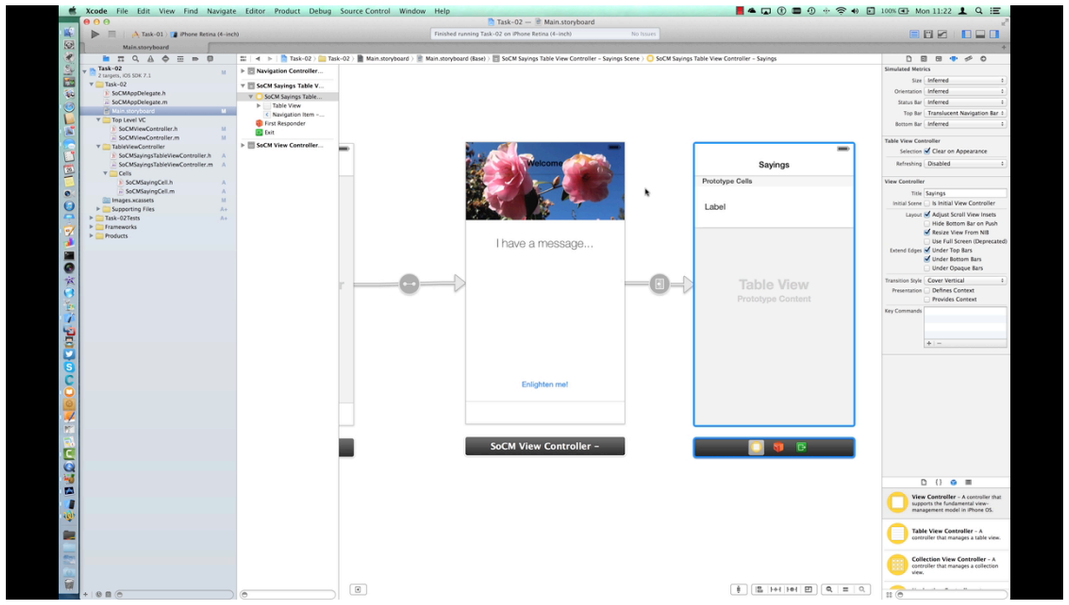
Note the order in which the above it done. You might want to try swapping the order of these two operations around. Why does it crash?

If you had made the model type NSArray, what would happen?

Note also - all these methods are in the controller. It is the UITableView that is sending the messages to the controller, and the controller is replying.

Right click on the TableView. You will see two references to the controller (already set up for you).

Movie 2.3 Adding swipe-to-delete functionality



Note that XCode has given you quite a substantial amount of boiler-plate code. If you ever use a UITableView without a UITableViewController, you can always copy and paste this code into your own controllers.

TASK 02-04

In this task, we add support for re-ordering rows in the table.

Watch Movie 2.4

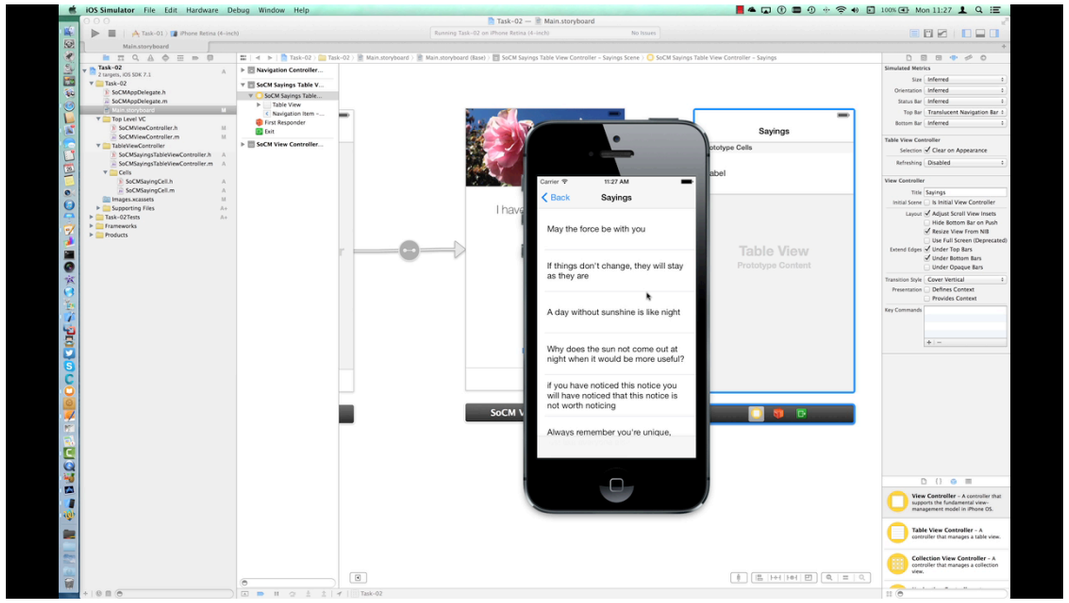
Key tasks:

Uncomment code in viewDidLoad to add the edit button

Uncomment code to support row re-ordering, and add the code to mirror the changes in the model:

```
[self.array exchangeObjectAtIndex:fromIndexPath.row
                withObjectAtIndex:toIndexPath.row];
```

Movie 2.4 Supporting cell re-ordering



Much like the previous task, the UITableViewController template provides commented code to quickly implement cell row reordering.

Again,you MUST make sure any changes to the UI are mirrored in the data model

TASK 02-05 - PUSHING DATA BACK DOWN THE DATA STACK

Changes in the table view need to be saved. Currently, when you navigate back to the first view controller, all changes are lost.

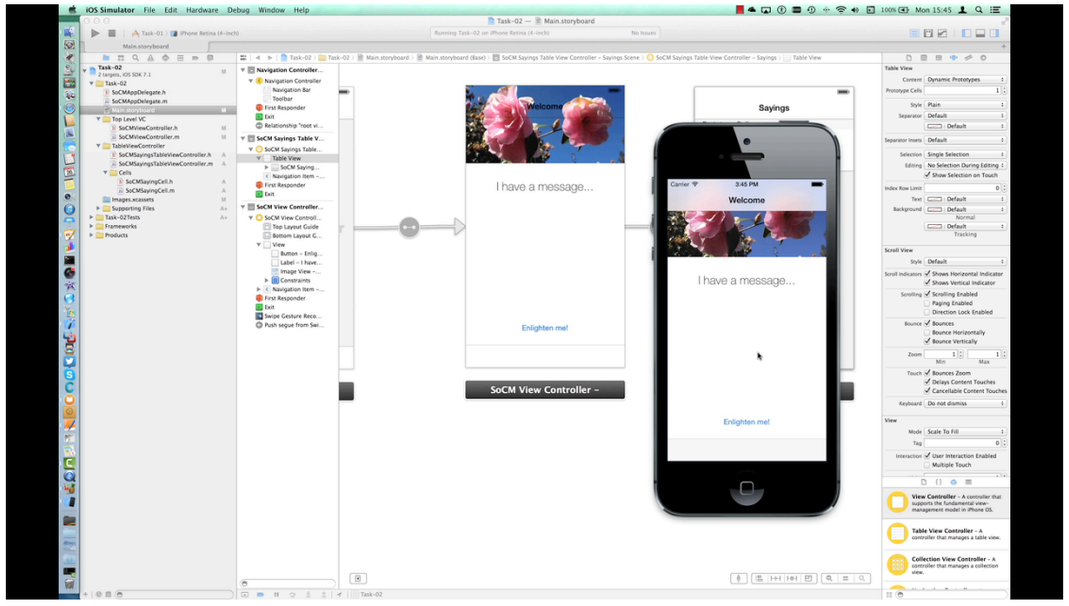
For this to work, the table view controller needs to send a message back to the first view controller with information on how to update the model (in this case, it is the complete model.

In prepareForSegue, the initial view controller needs to make it self the delegate of the table view (set up a reverse reference) as well as pass it model data.

The initial table view needs to “promise” to implement the call-back methods sent by the table view controller. It does this by “conforming to a protocol”

Watch Movie 2.5

Movie 2.5 Pushing data back down the data stack



A key aspect of this is delegation and protocol conformance.

TASK 02-06 PERSISTENCE

The last item in this task is to be able to save / load the model to / from disk (flash memory).

For this, we introduce two new classes:

NSKeyedArchiver

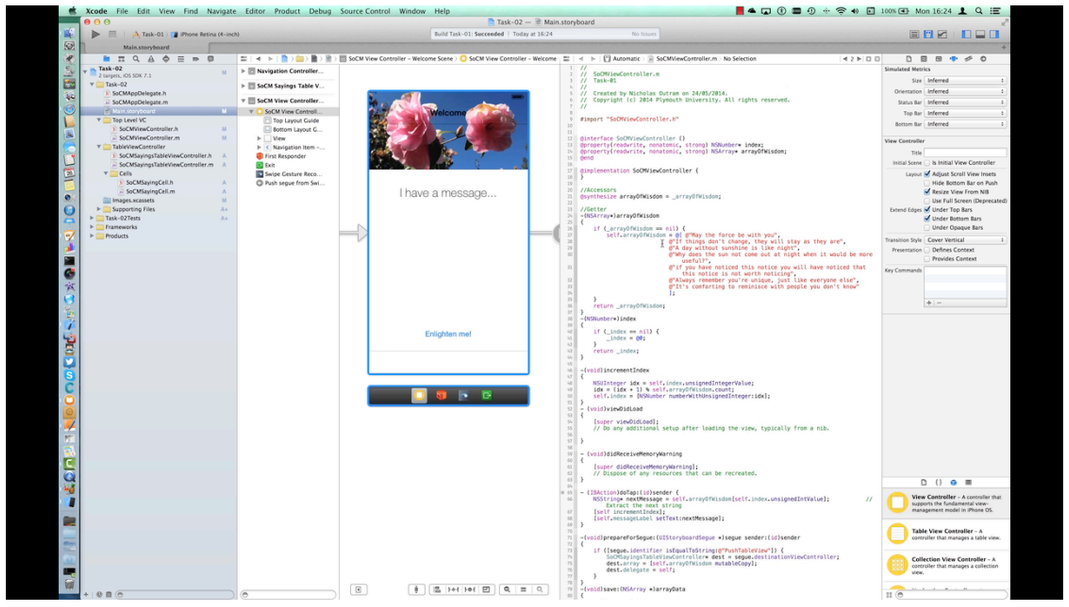
NSKeyedUnarchiver

These classes can recursively archive (that's serialization to Java programmers!) an object that conforms to the NSCoding protocol.

(All the collections, such as NSArray, NSSet, NSDictionary conform to this protocol)

I've added an additional save button in this application to help separate each step. You might want to improve on this design.

Movie 2.6 Persisting changes to disk



In this example, we use keyed archiving. Other methods could include CoreData, UIDocument, SQLlite or writing a bespoke filing system.

IBAction

Defined as void. When used as a return type, this serves two purposes - the method has no return data; XCode knows this method is to be used as an event handler. When you right click on a control, this method will be listed.

Related Glossary Terms

Drag related terms here

Index Find Term

Immutable

An object who's contents is defined at create time and which cannot be subsequently changed.

Related Glossary Terms

Drag related terms here

Index Find Term

Lazy loading

Where an object is not instantiated and initialized until it is first read. This technique has the advantage that if an object only consumed memory if it is actually needed. It also creates a tidy mechanism for initialization that does not depend on the life-cycle of other objects (constructors / initializers).

Related Glossary Terms

Drag related terms here

Index Find Term

Outlet

This term refers to a C pointer that is designated to point to a view object. This view object will be loaded using either StoryBoard or “Nib loading”. The Keywork IBOutlet is defined as blank, so does nothing to the code. However, it does tell XCode that this is a pointer to a view object.

Related Glossary Terms

Drag related terms here

Index Find Term

Property

Using the @property keyword, this creates one or more accessors on a property of an object. A property is often backed by an instance variable, but this is not always necessary (it could be derived from many others).

Related Glossary Terms

Drag related terms here

Index

Find Term