Stanford CS193p

Developing Applications for iOS
Fall 2011

Today

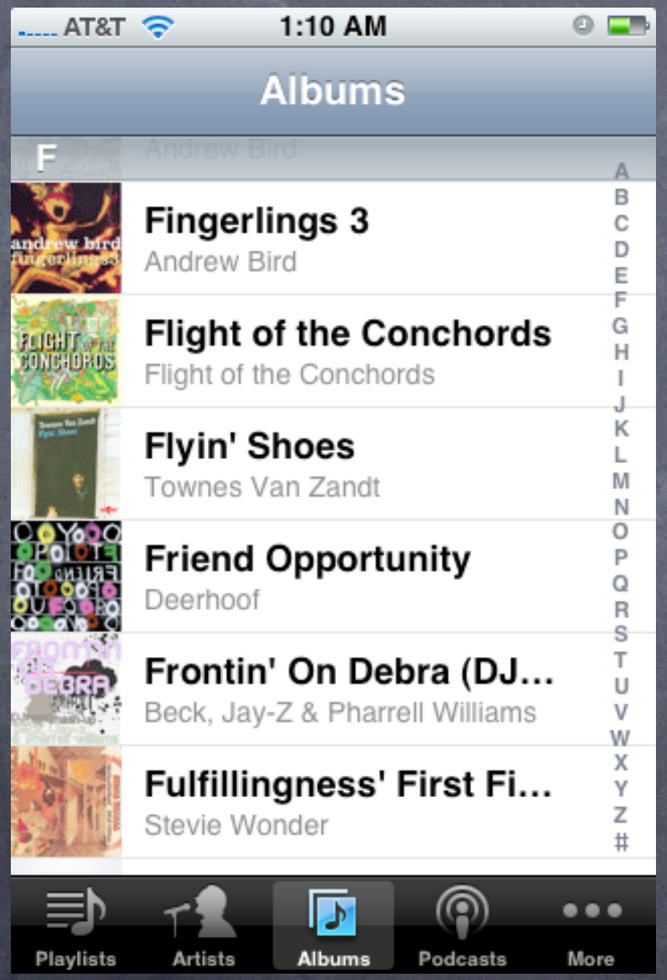
- Table View
  Displaying a dynamic list of data.
  Or displaying a fixed table of data.
- Demo
  Favorite graphs feature in Calculator.
  Popover with delegate call back.
  Table view with dynamic list of data.

UITableView

- Very important class for displaying data in a table
  - One-dimensional table.
  - It's a subclass of `UIScrollView`.
  - Table can be static or dynamic (i.e. a list of items).
  - Lots and lots of customization via a `dataSource` protocol and a `delegate` protocol.
  - Very efficient even with very large sets of data.
- Displaying multi-dimensional tables ...
  - Usually done via a `UINavigationController` containing multiple MVC's where View is `UITableView`
- Kinds of `UITableViews`
  - Plain or Grouped
  - Static or Dynamic
  - Divided into sections or not
  - Different formats for each row in the table (including completely customized)

UITableView

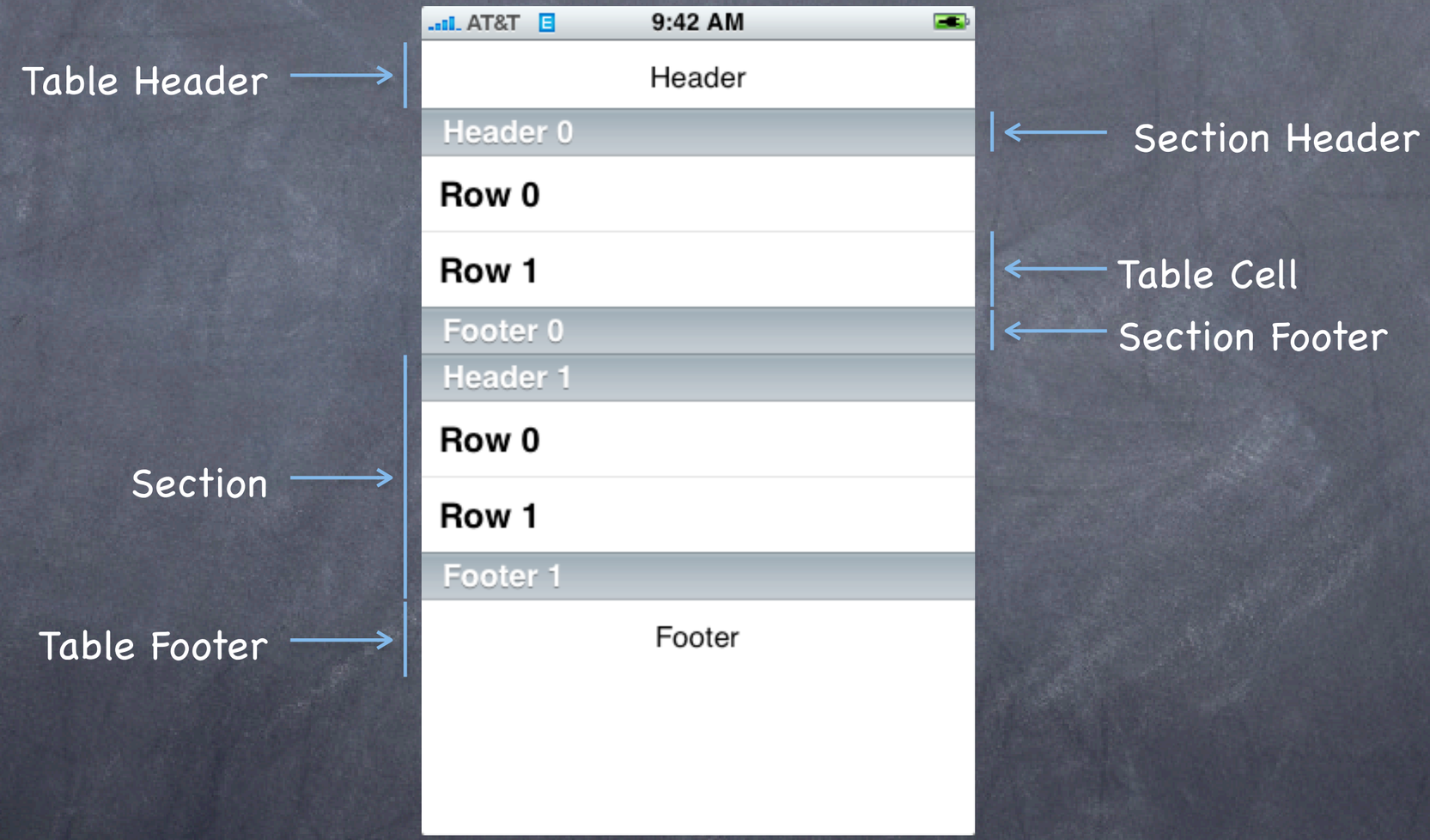
UITableViewStylePlain



UITableViewStyleGrouped

UITableView

Plain Style



UITableView

Grouped Style

Table Header

Section

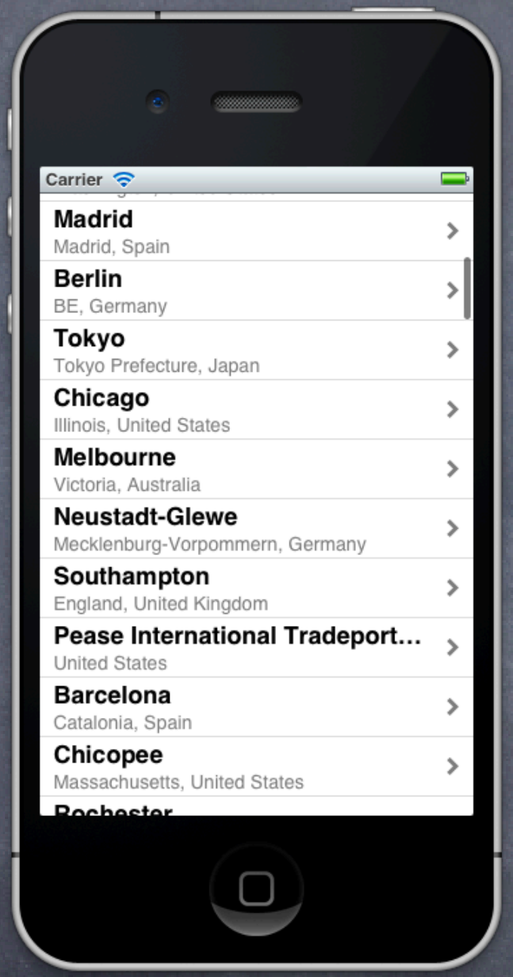
Table Footer

Section Header

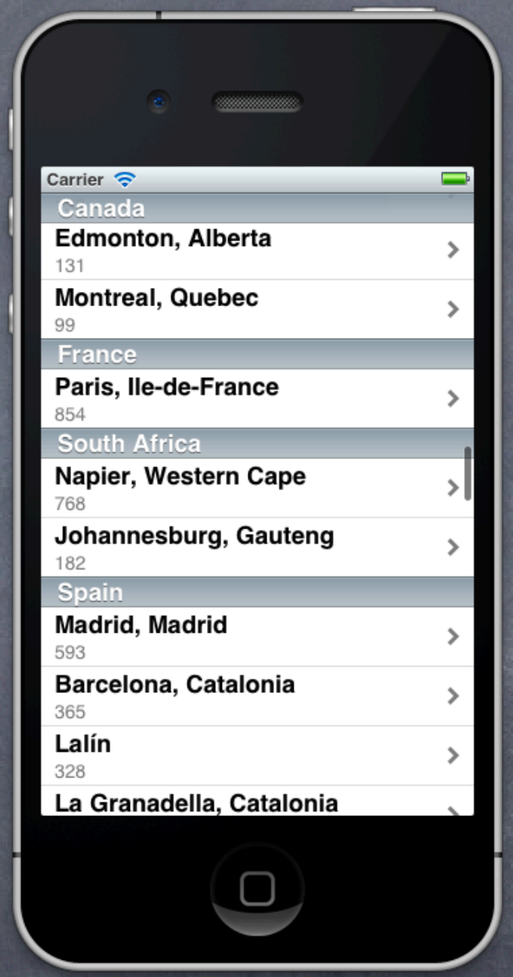
Table Cell

Section Footer

Sections or Not

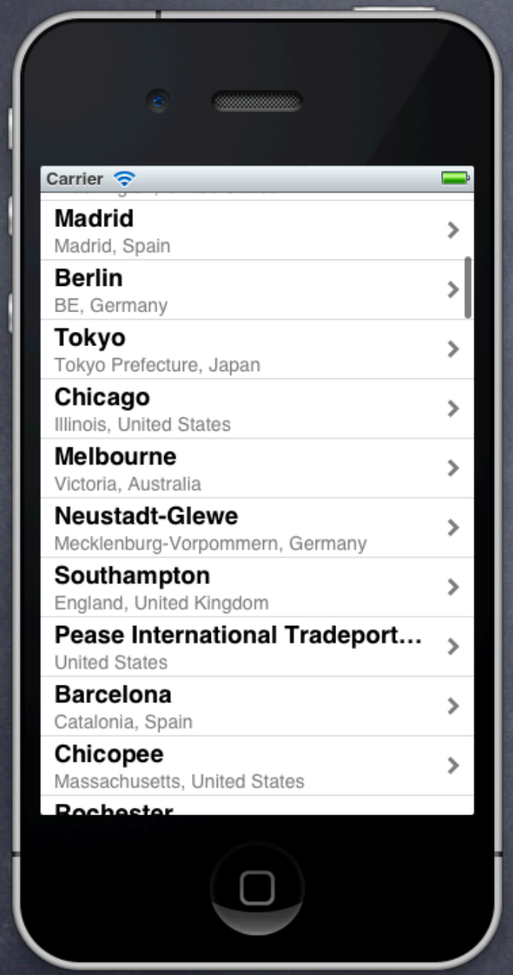


No Sections



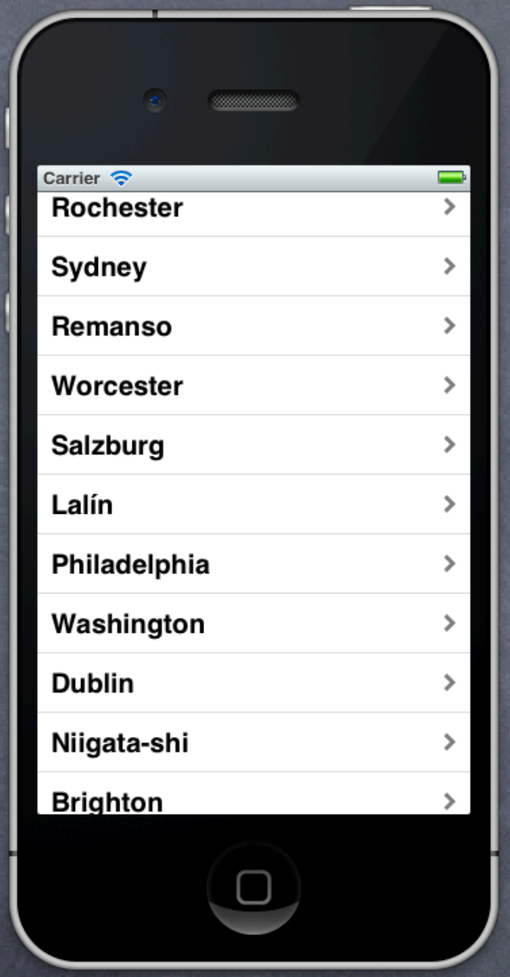
Sections

Cell Type



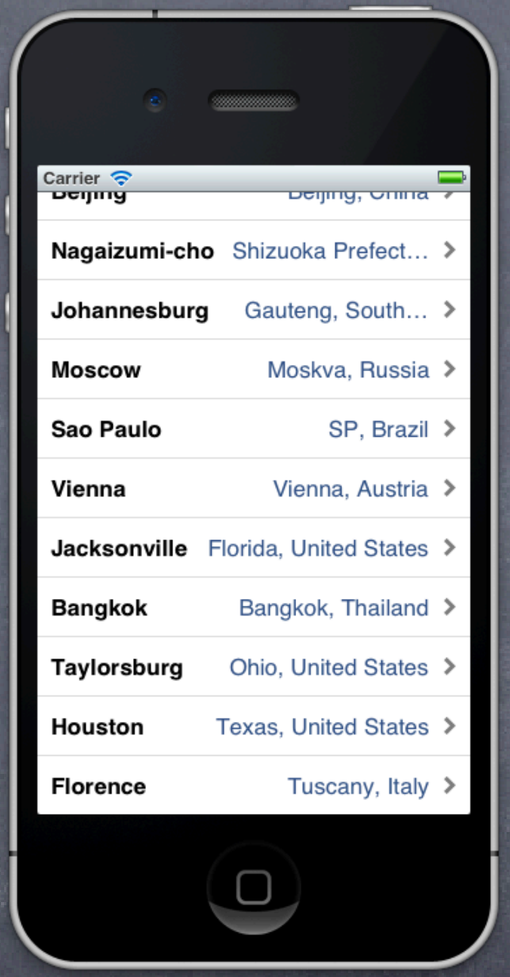
Subtitle

`UITableViewCellStyleSubtitle`



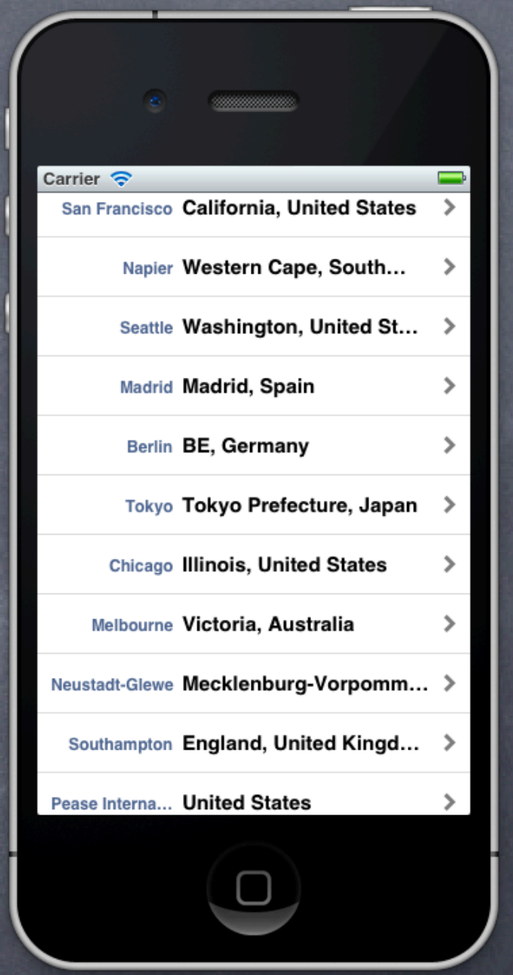
Basic

`UITableViewCellStyleDefault`



Right Detail

`UITableViewCellStyleValue1`



Left Detail

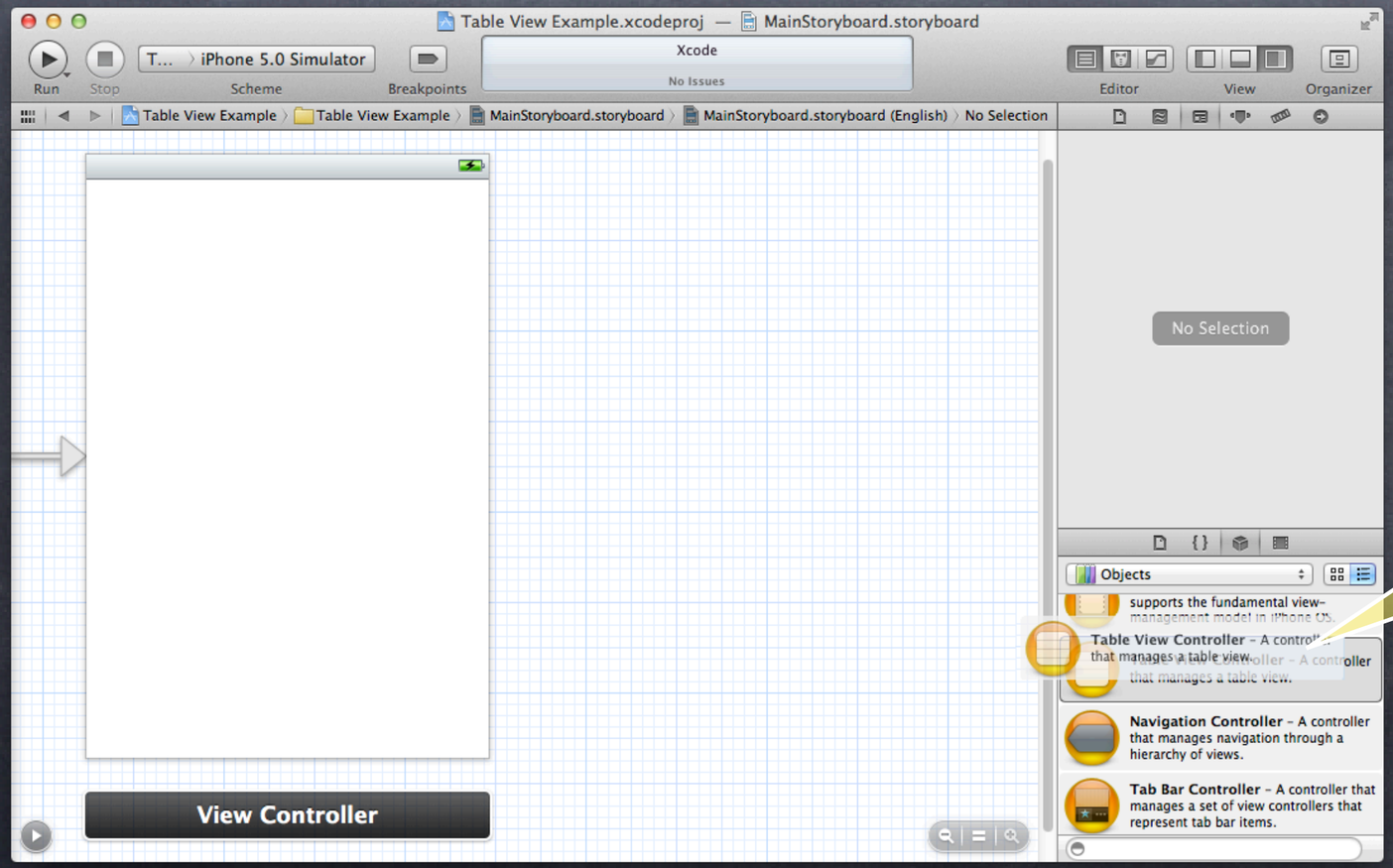
`UITableViewCellStyleValue2`

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just drag one out in Xcode, create a subclass of it and you're on your way!



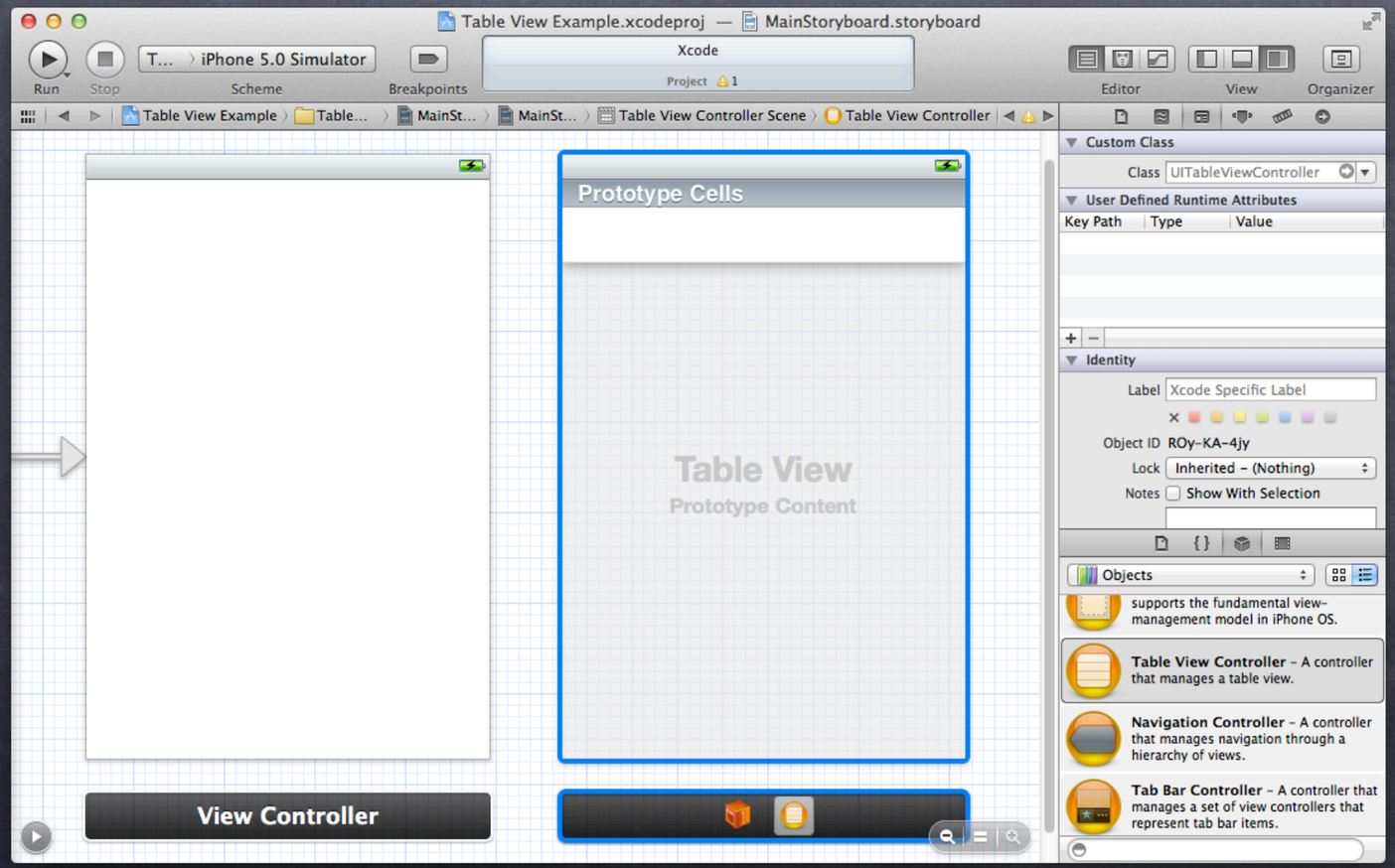
Dragging out a `UITableViewController`

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just drag one out in Xcode, create a subclass of it and you're on your way!

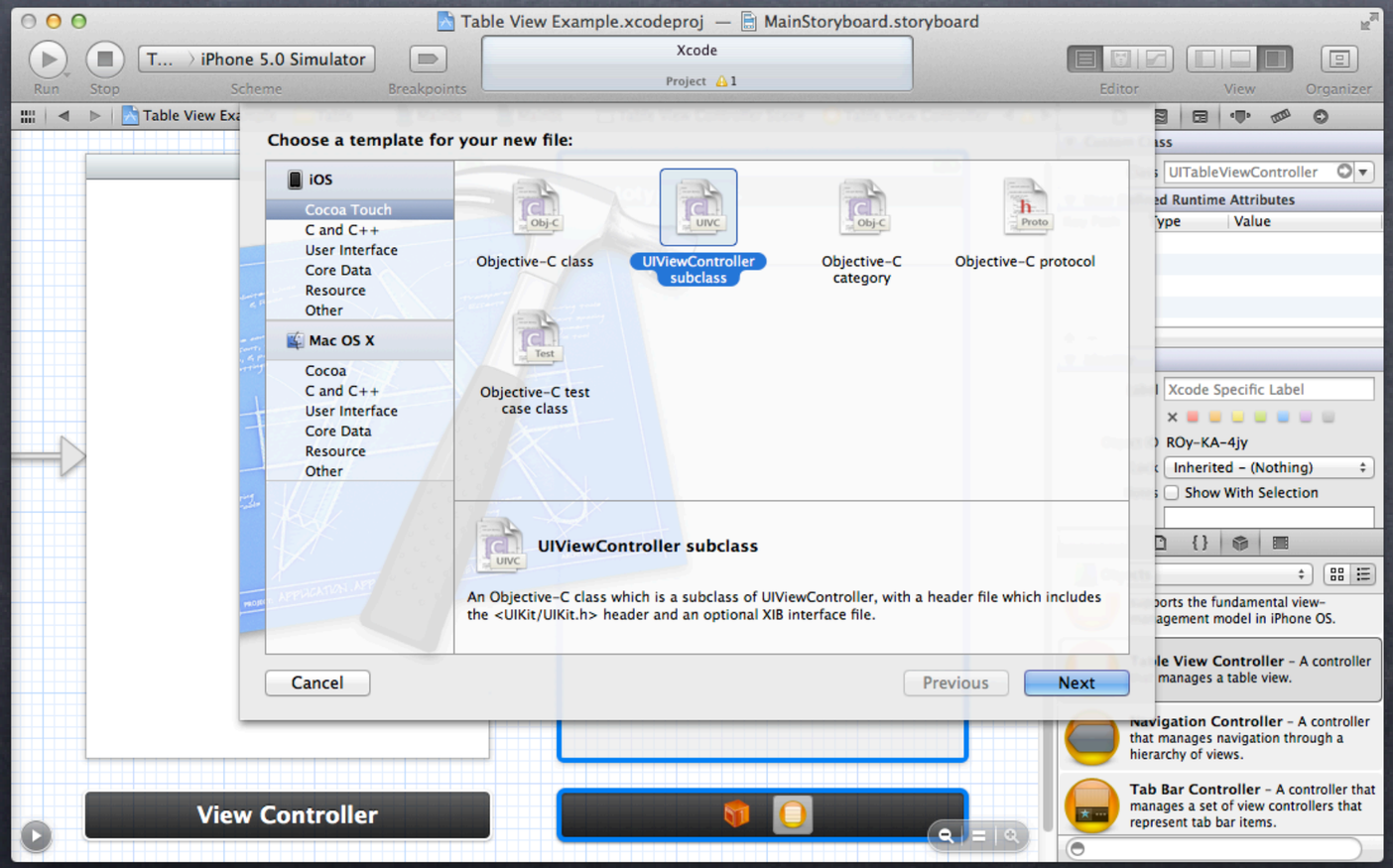


Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just drag one out in Xcode, create a subclass of it and you're on your way!



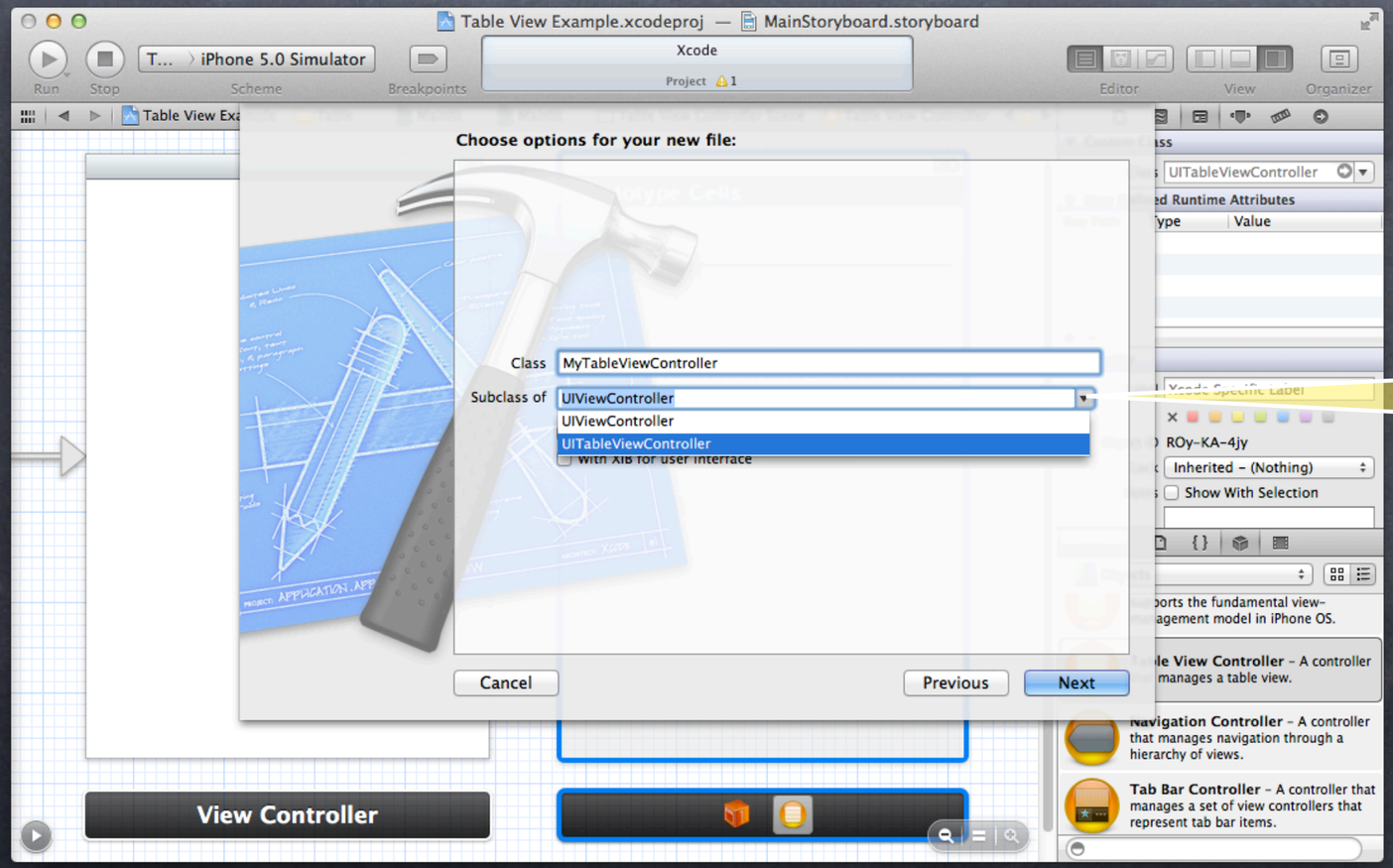
Choose New File ... from the File menu to create a custom subclass of `UITableViewController`

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just drag one out in Xcode, create a subclass of it and you're on your way!



Choose New File ... from the File menu to create a custom subclass of `UITableViewController`

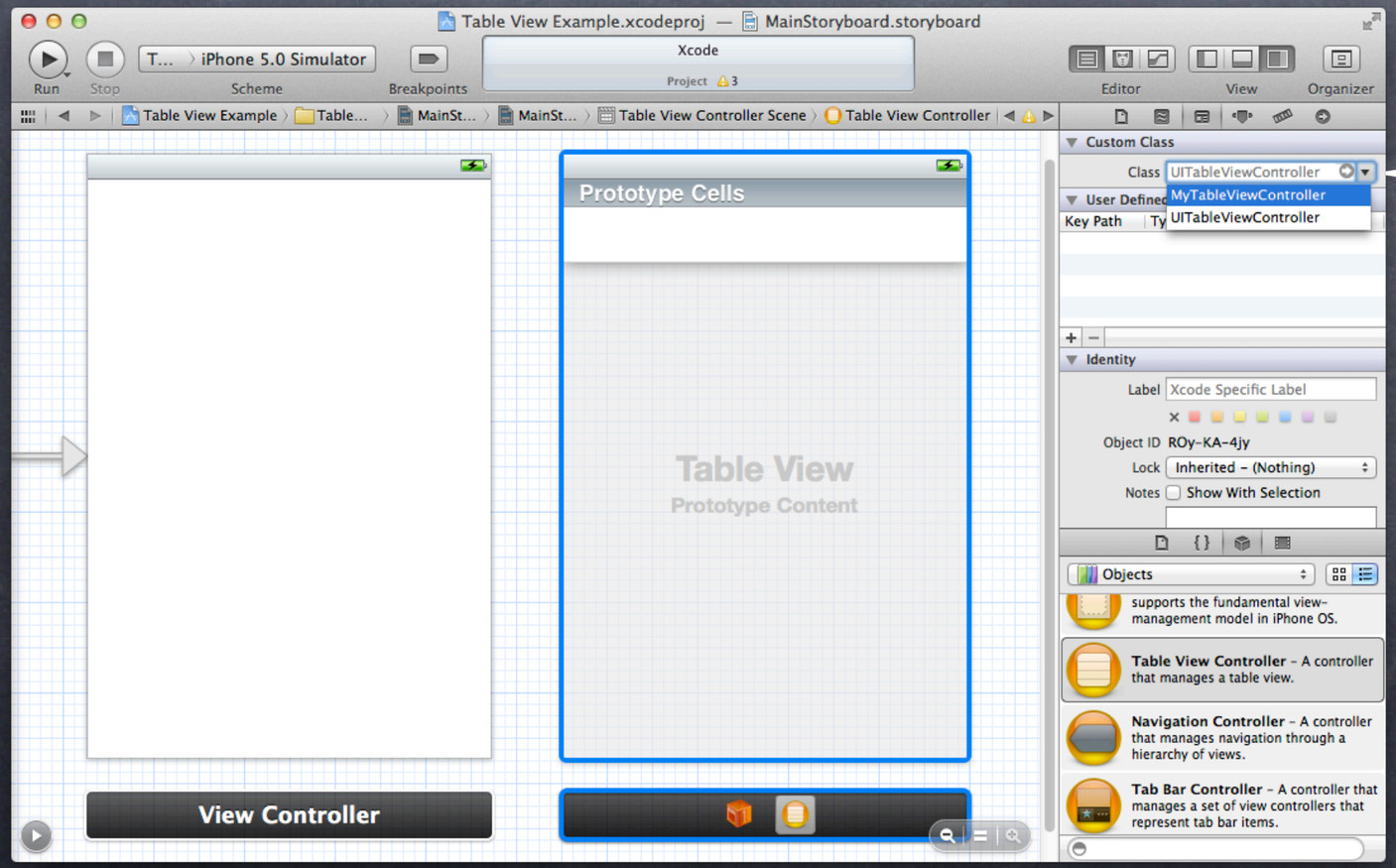
Be sure to set the superclass to `UITableViewController`!!

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just drag one out in Xcode, create a subclass of it and you're on your way!

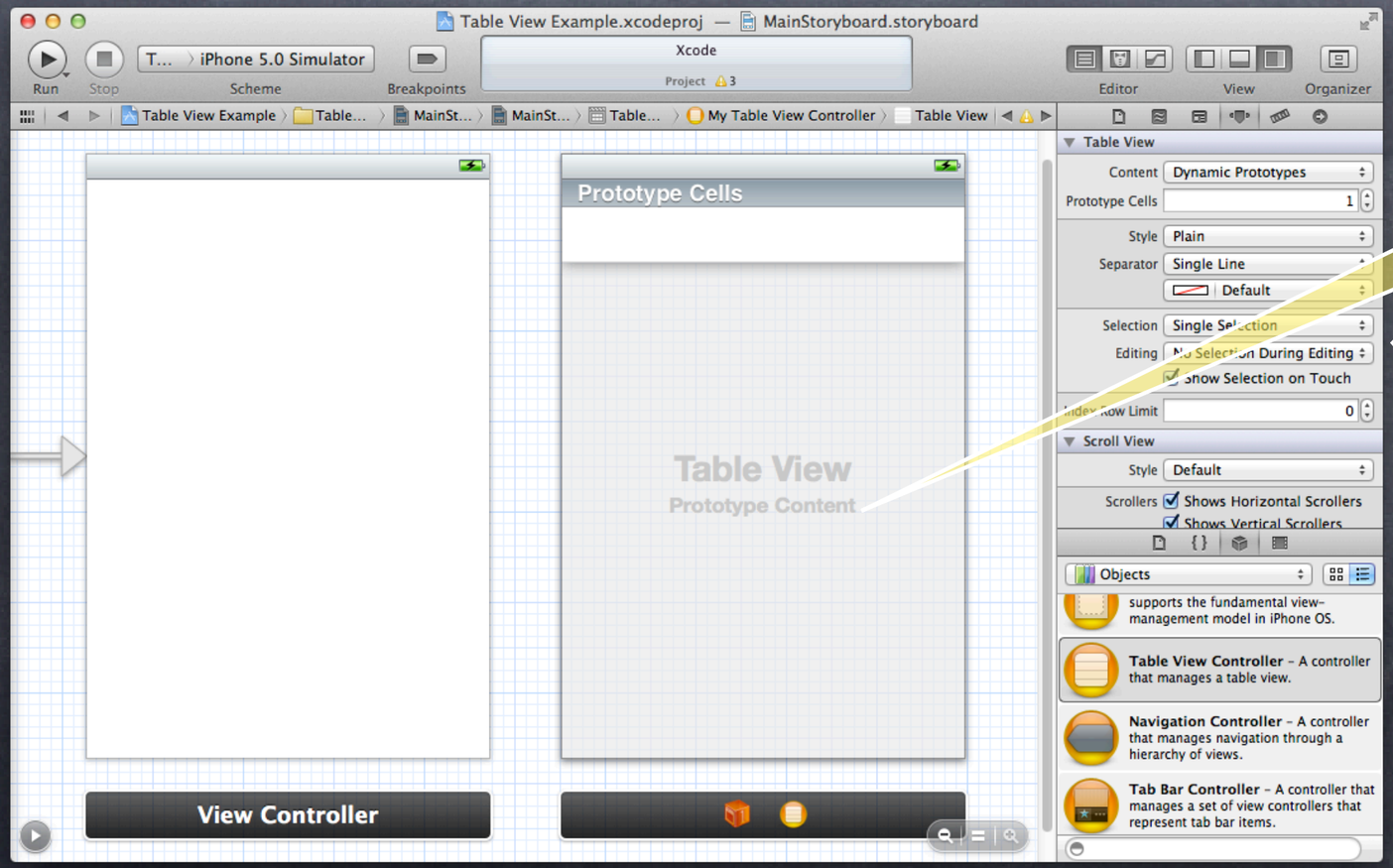


And to set it in your storyboard too!!

Creating Table View MVCs

- UITableView

  You can customize both the look of the table view and its cells from Xcode.
  Click on the table view (not the table view controller) to see its properties in the Inspector.



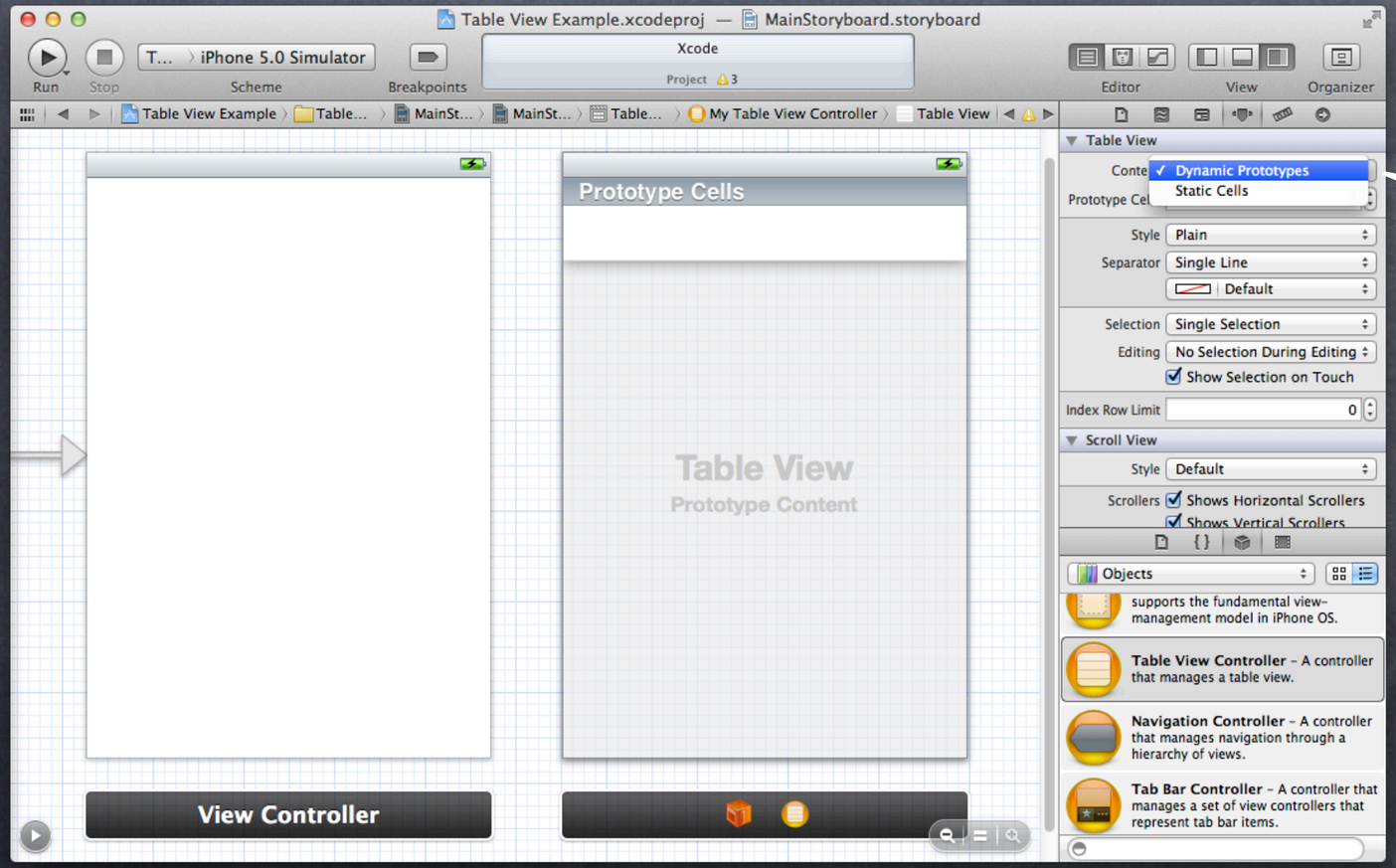
Click here to select the UITableView inside the UITableViewController's View.

Then you should see table view properties in the Attributes Inspector.

Creating Table View MVCs

- UITableView

  You can customize both the look of the table view and its cells from Xcode.
  Click on the table view (not the table view controller) to see its properties in the Inspector.

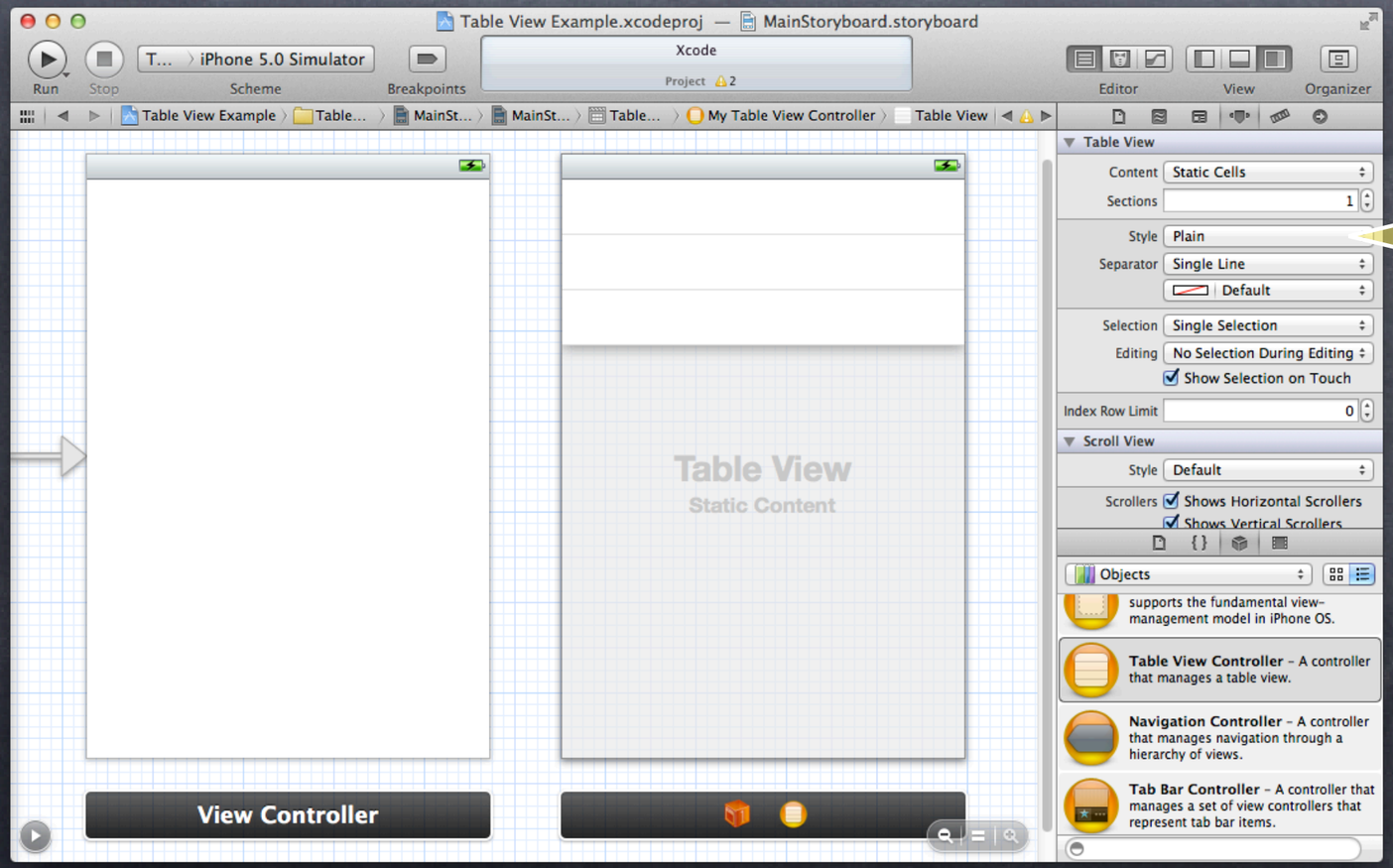


We'll talk about the prototype for a Dynamic Table View in a moment, but for now, switch to a Static Table View.

Creating Table View MVCs

- UITableView

  You can customize both the look of the table view and its cells from Xcode.
  Click on the table view (not the table view controller) to see its properties in the Inspector.

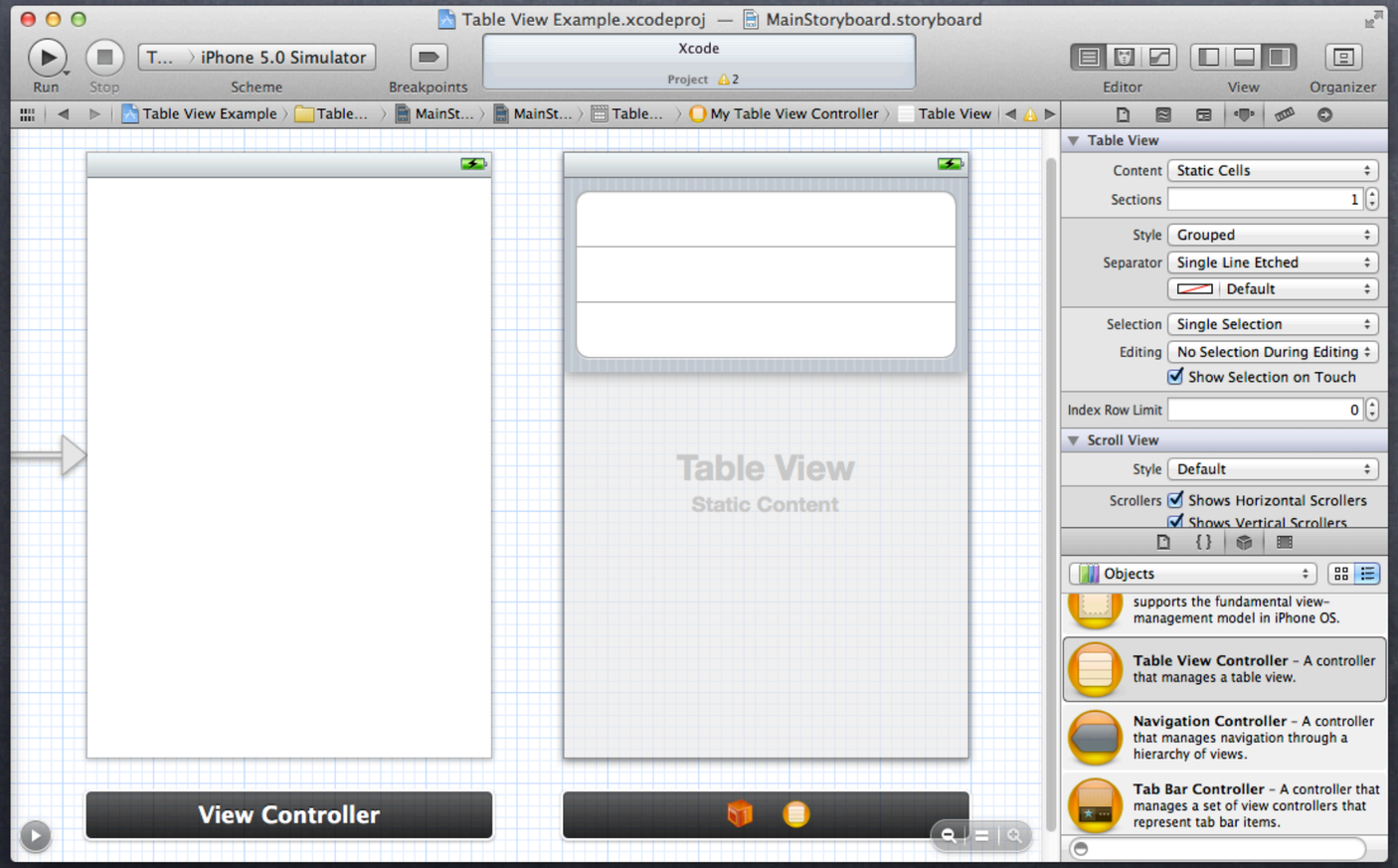


This table view has the Plain style. Let's change it to Grouped.

Creating Table View MVCs

- UITableView

  You can customize both the look of the table view and its cells from Xcode.
  Click on the table view (not the table view controller) to see its properties in the Inspector.

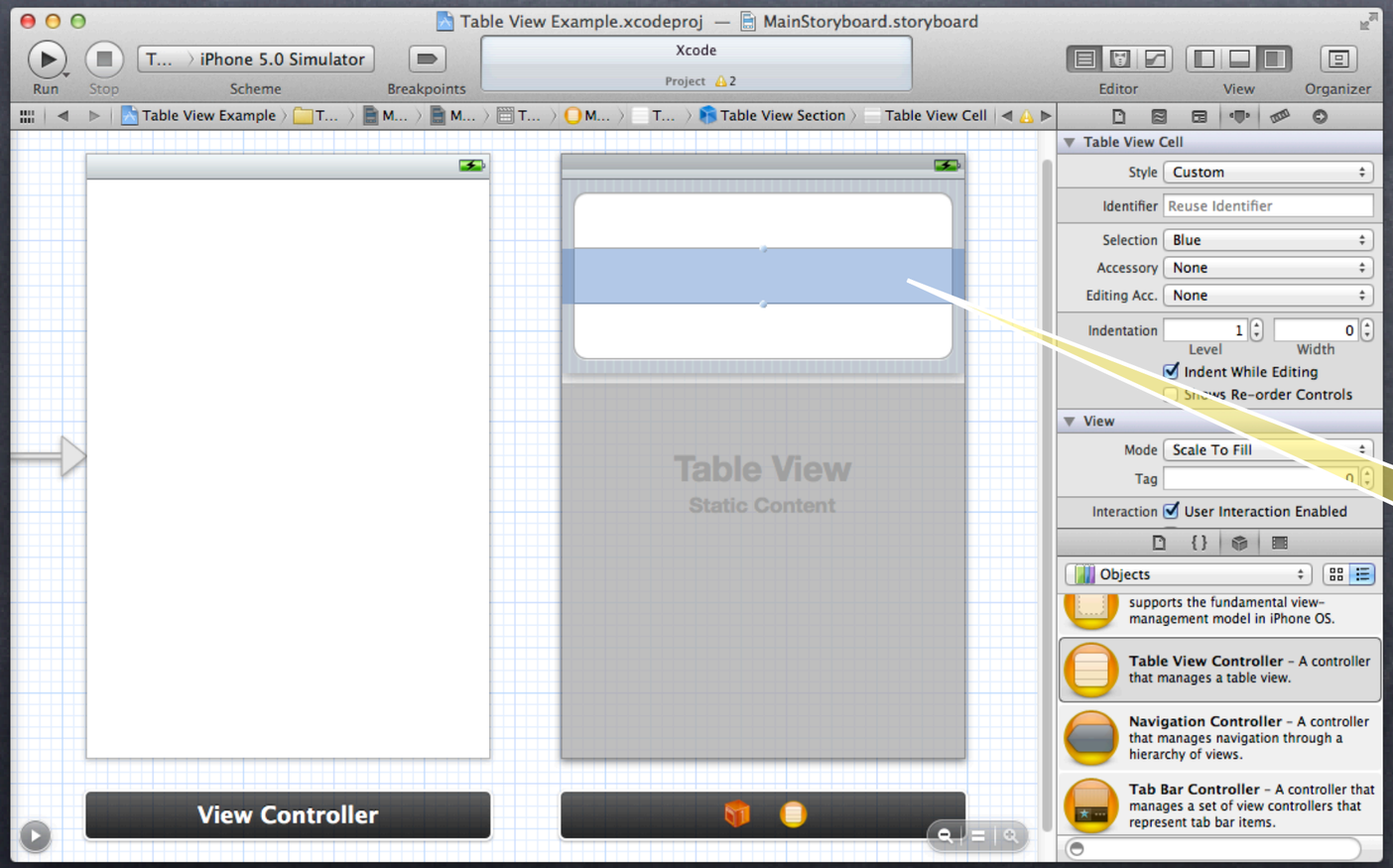


Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



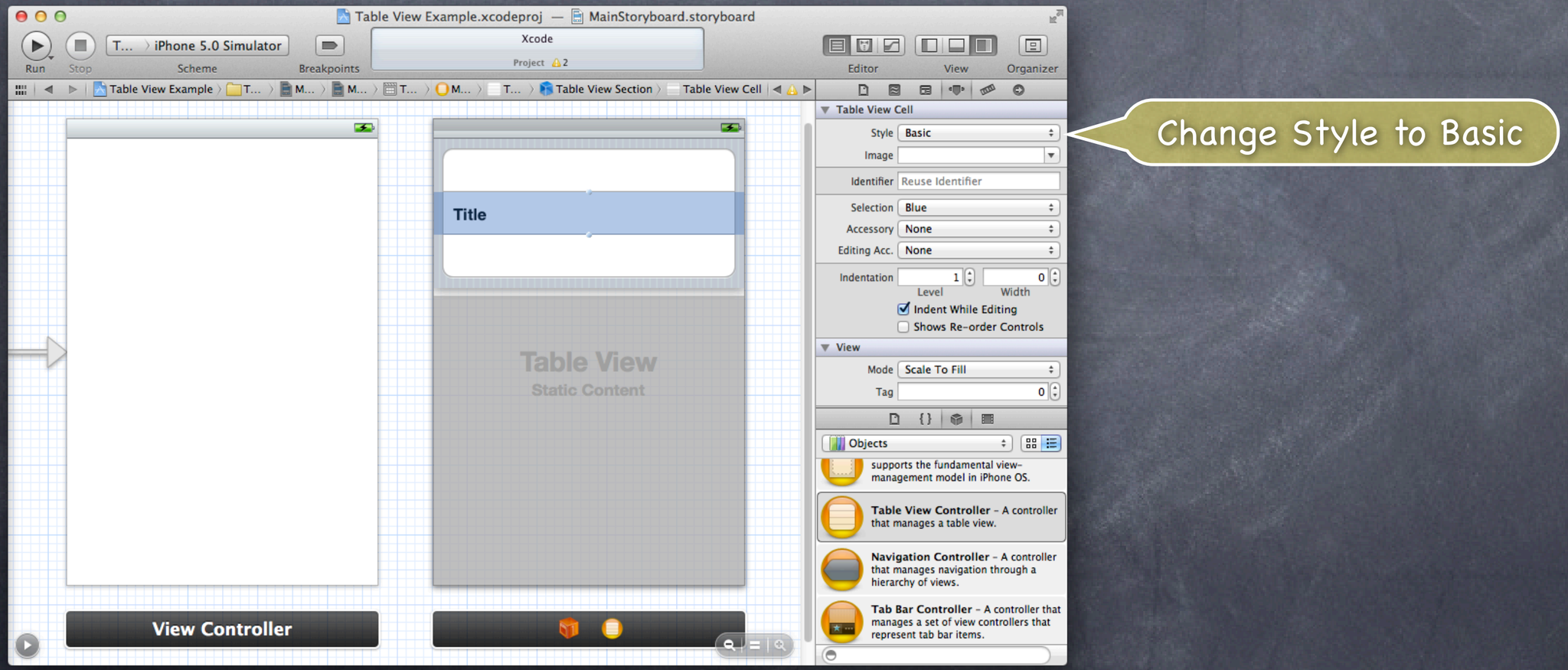
Click to select the cell whose properties we want to edit.

Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



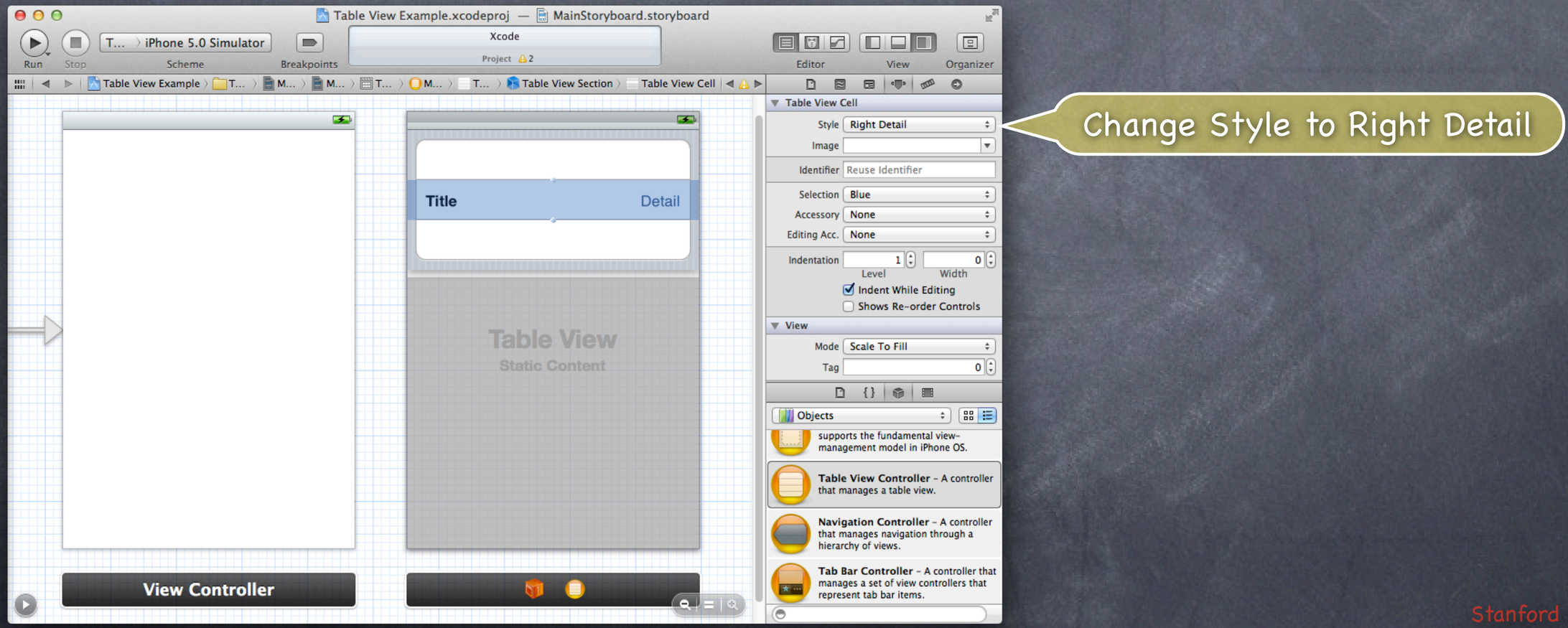
Change Style to Basic

Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



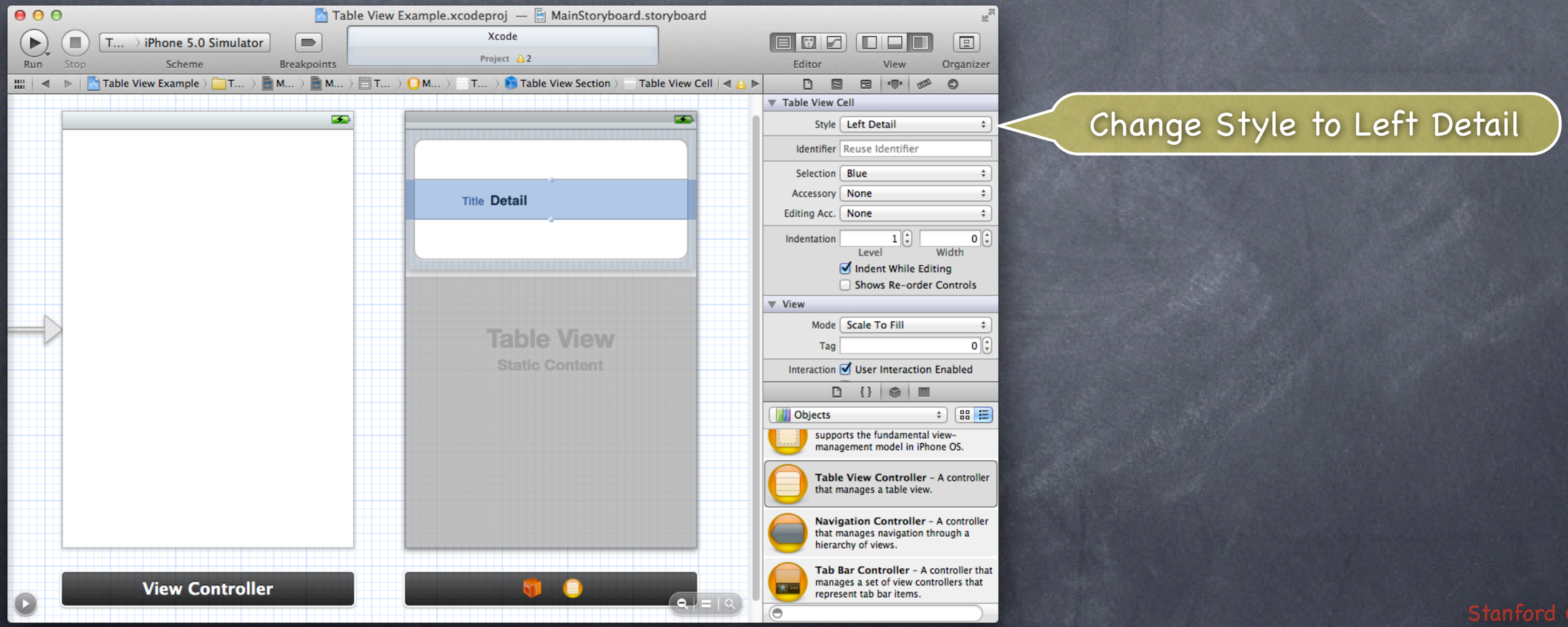
Change Style to Right Detail

Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



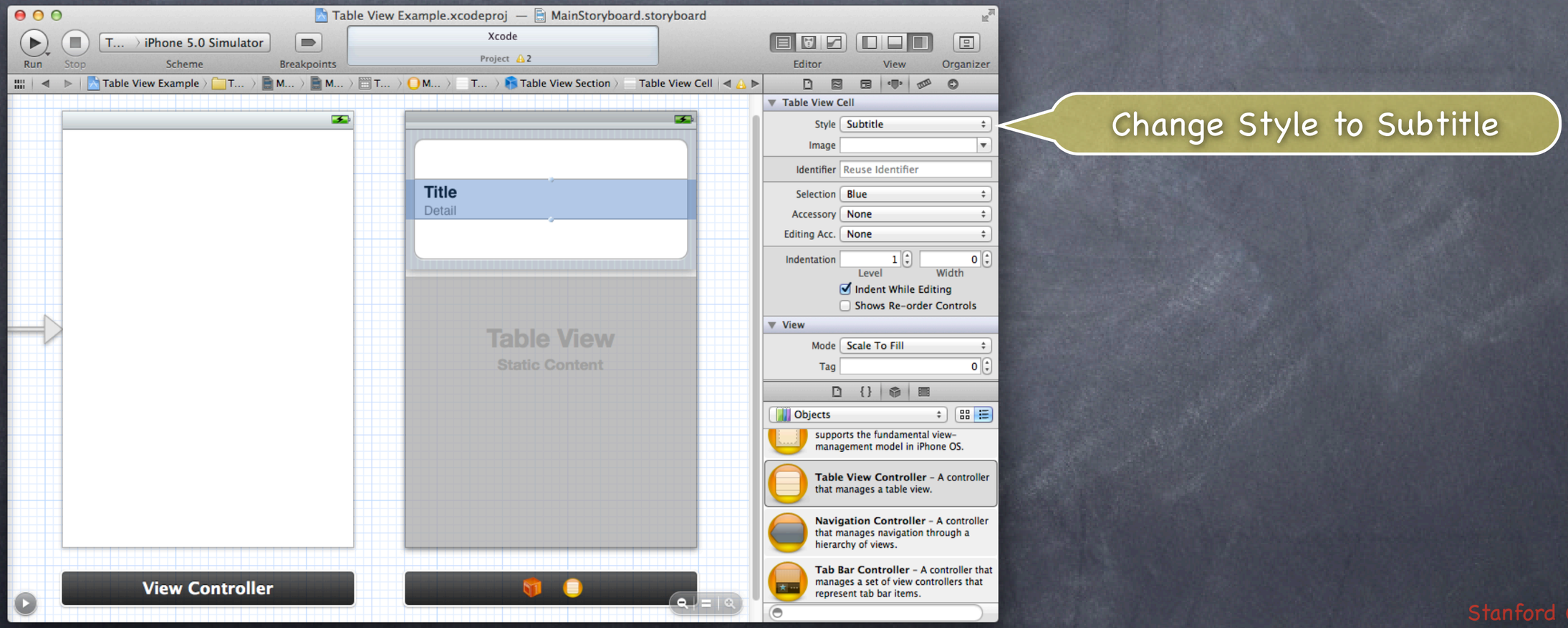
Change Style to Left Detail

Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



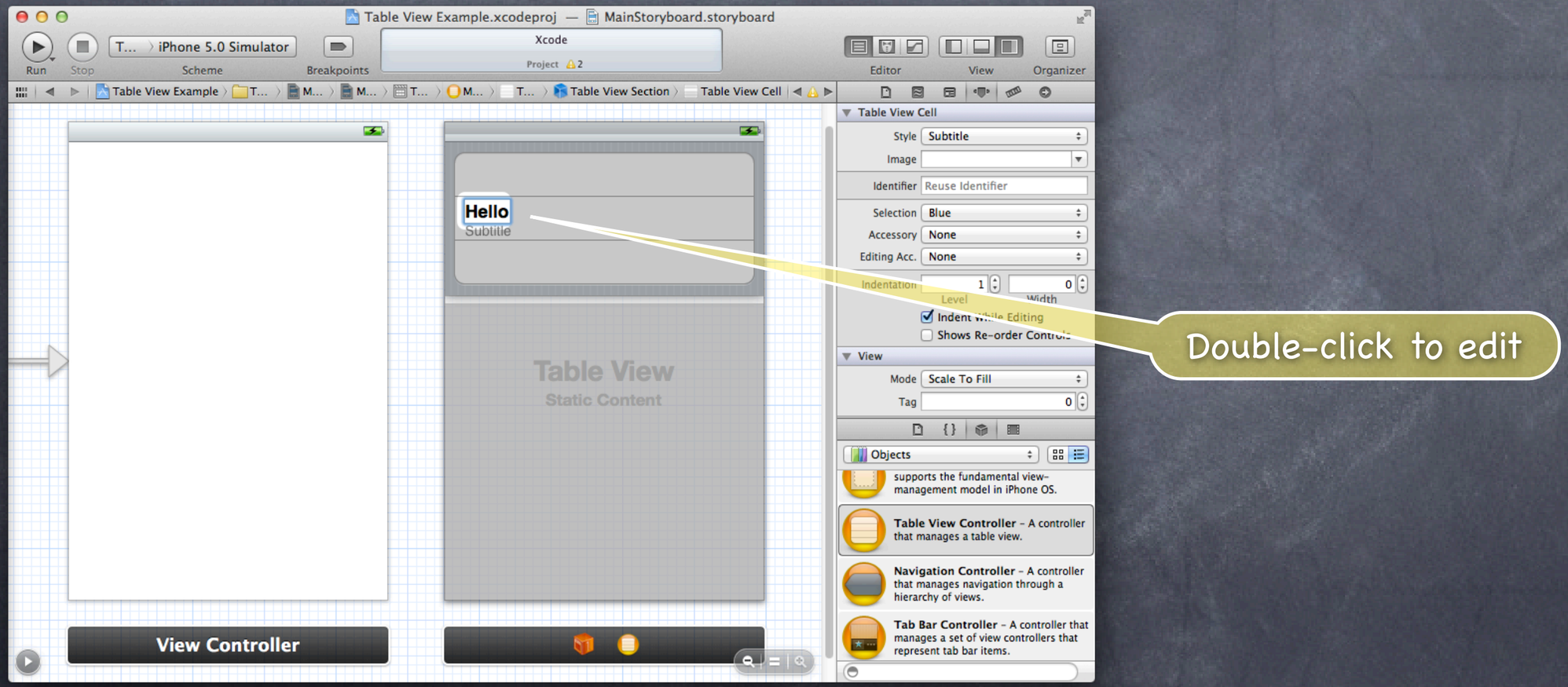
Change Style to Subtitle

Creating Table View MVCs

- `UITableViewCell`

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.

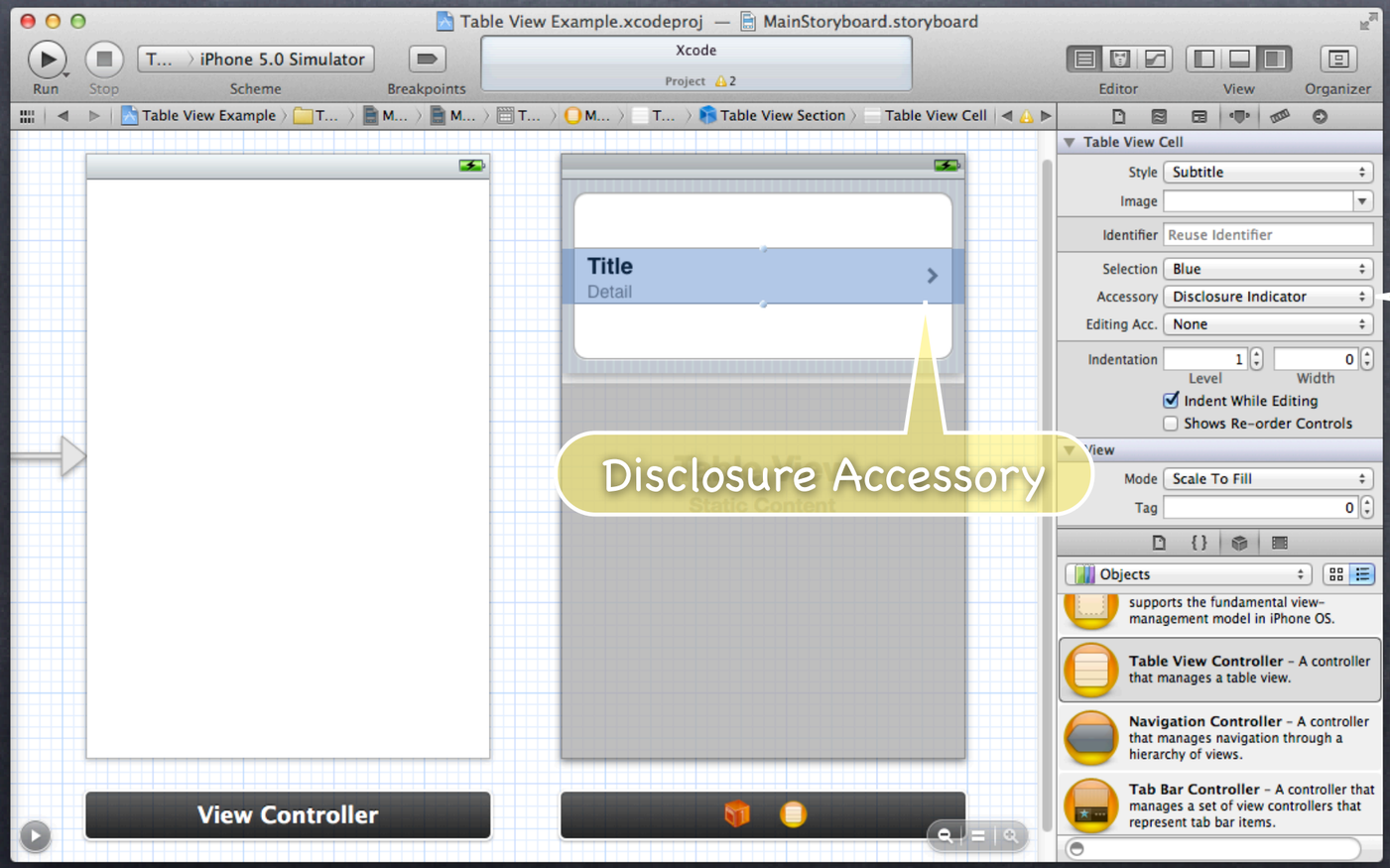


Creating Table View MVCs

- UITableViewCell

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



Disclosure Accessory

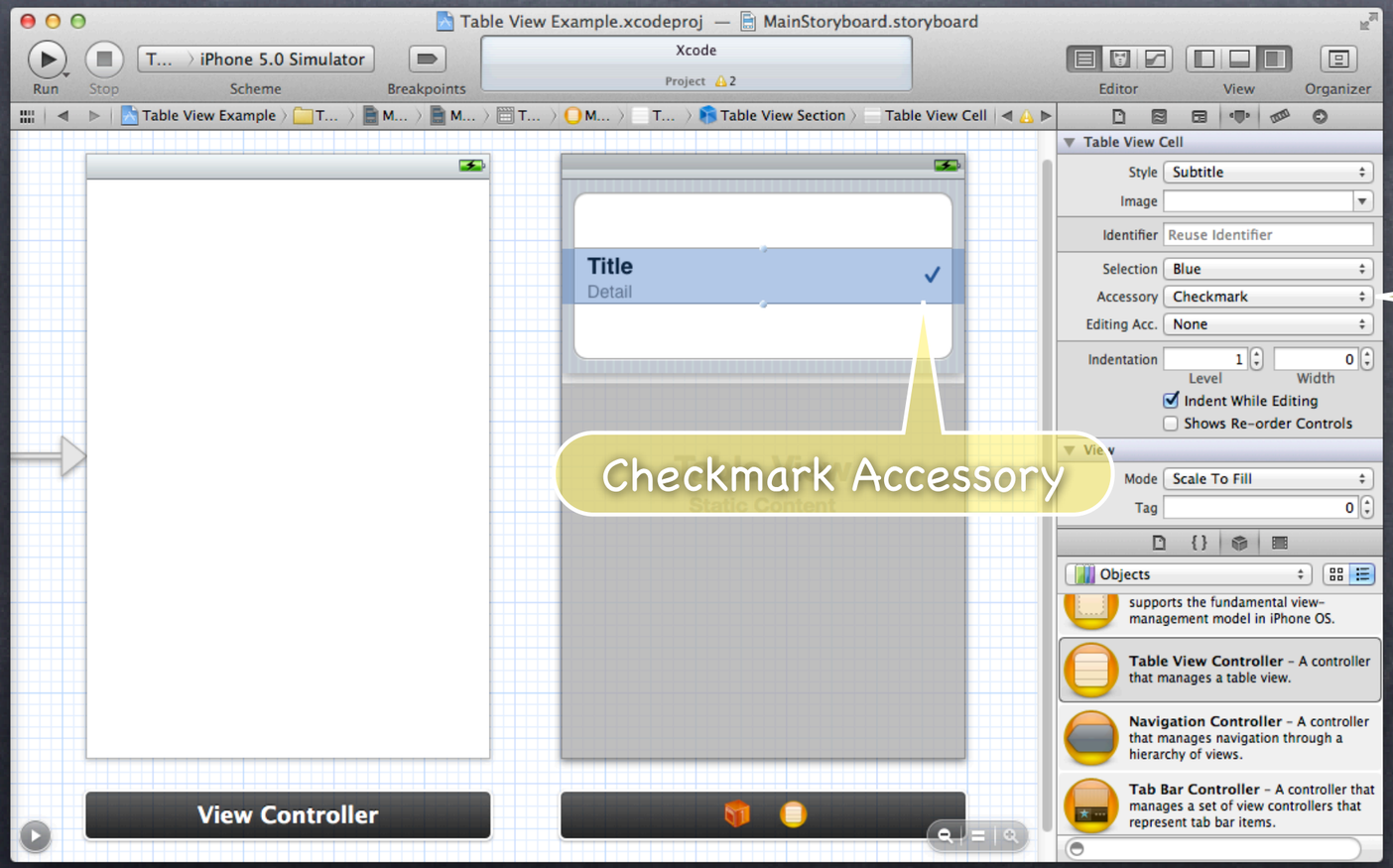
Show disclosure accessory. This should be on whenever clicking on a row in the table brings up another MVC.

Creating Table View MVCs

- UITableViewCell

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



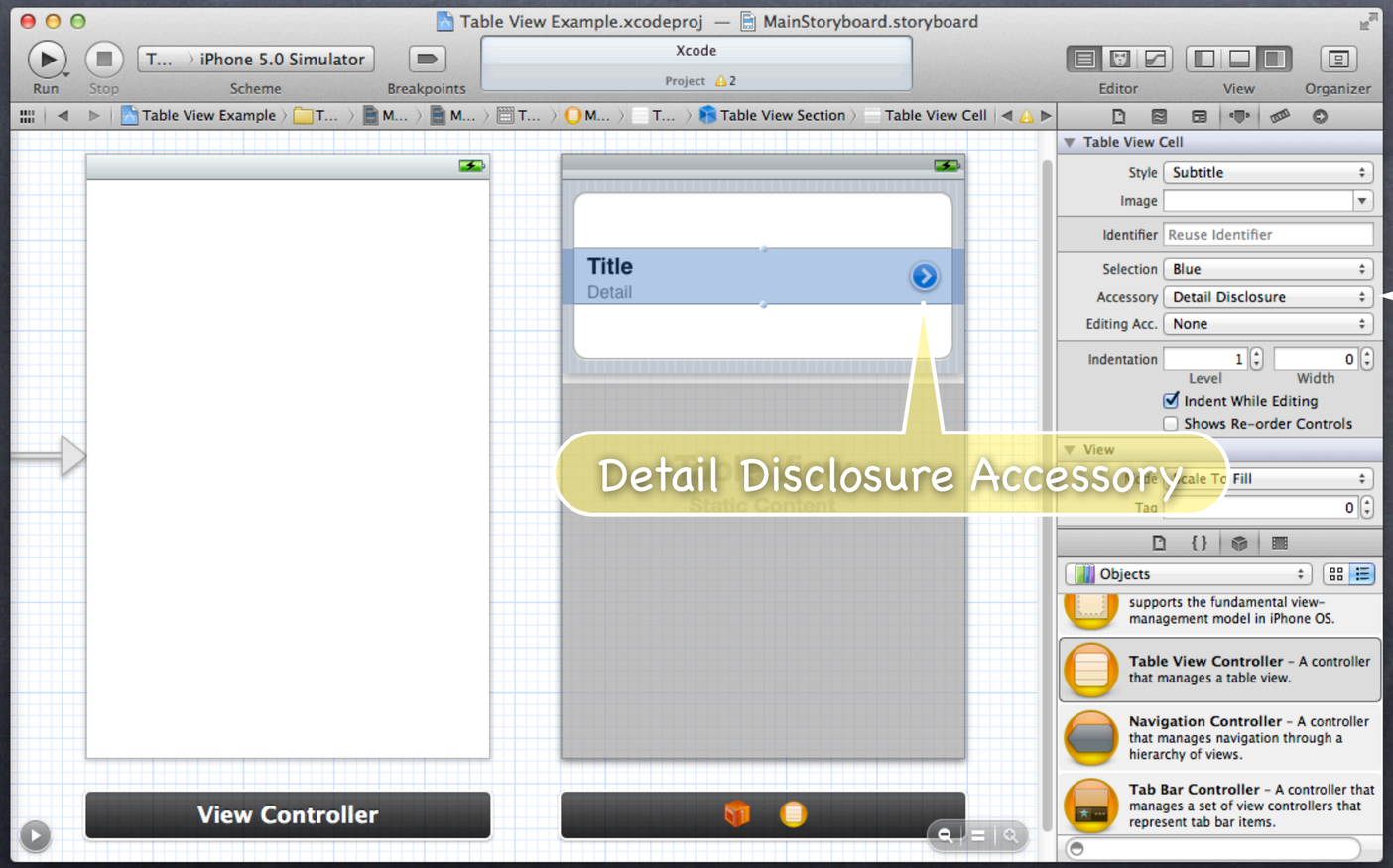
Show checkmark accessory. This can be used to show multiple selection in the table (requires some other API use).

Creating Table View MVCs

- UITableViewCell

  And you can change the look of each cell as well.

  Click on a cell that you want to change and set its attributes in the Inspector.



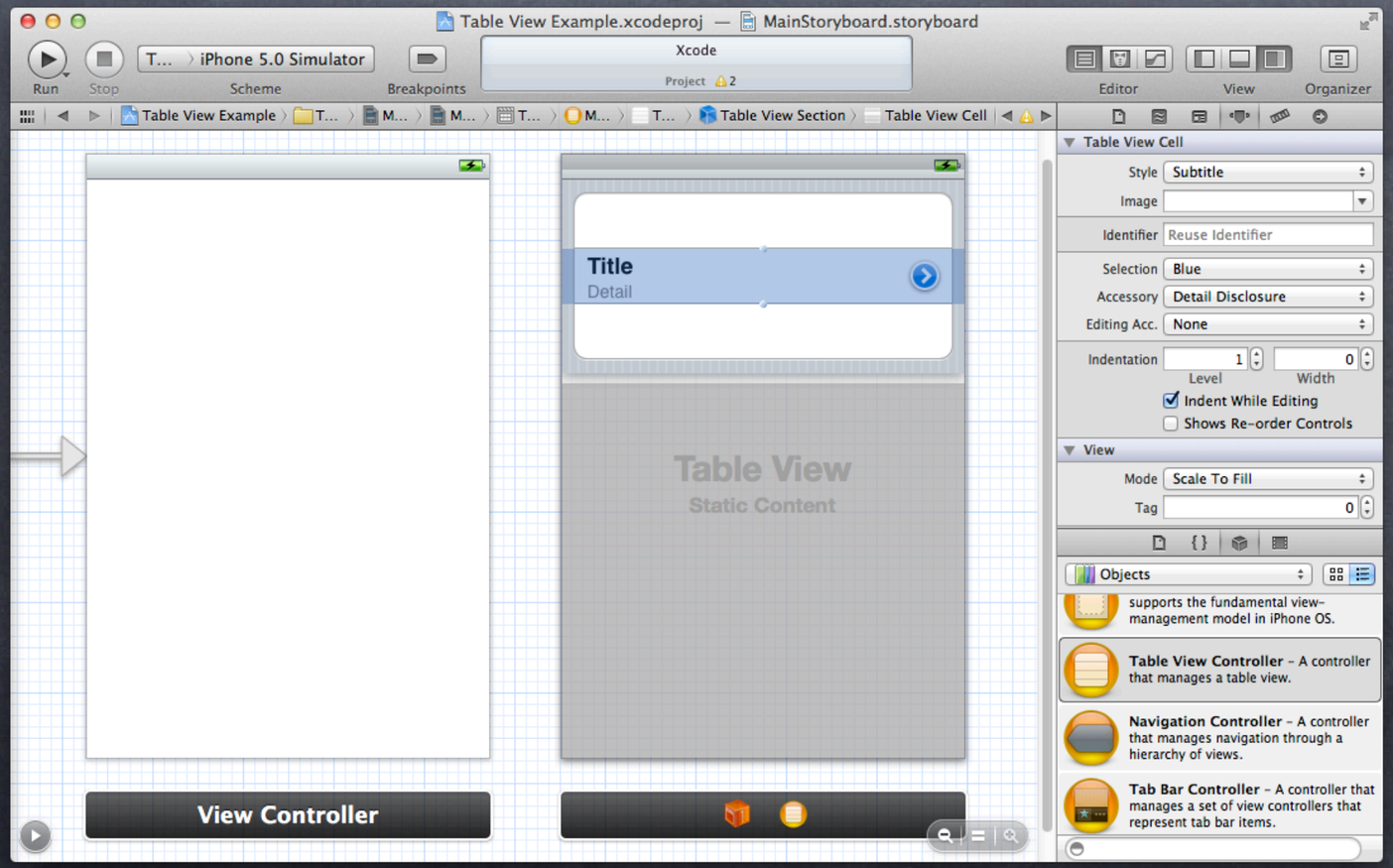
Show detail disclosure accessory.
This is an active control.
Use it to show ancillary info.
Clicking on the row should still do the "main thing" for this row.

Creating Table View MVCs

- UITableViewCell

  User taps on the blue detail disclosure below? This will be sent to your UITableViewController:

  ```
  - (void)tableView:(UITableView *)tv accessoryTappedForRowAtIndexPath:(NSIndexPath *)ip;
  ```

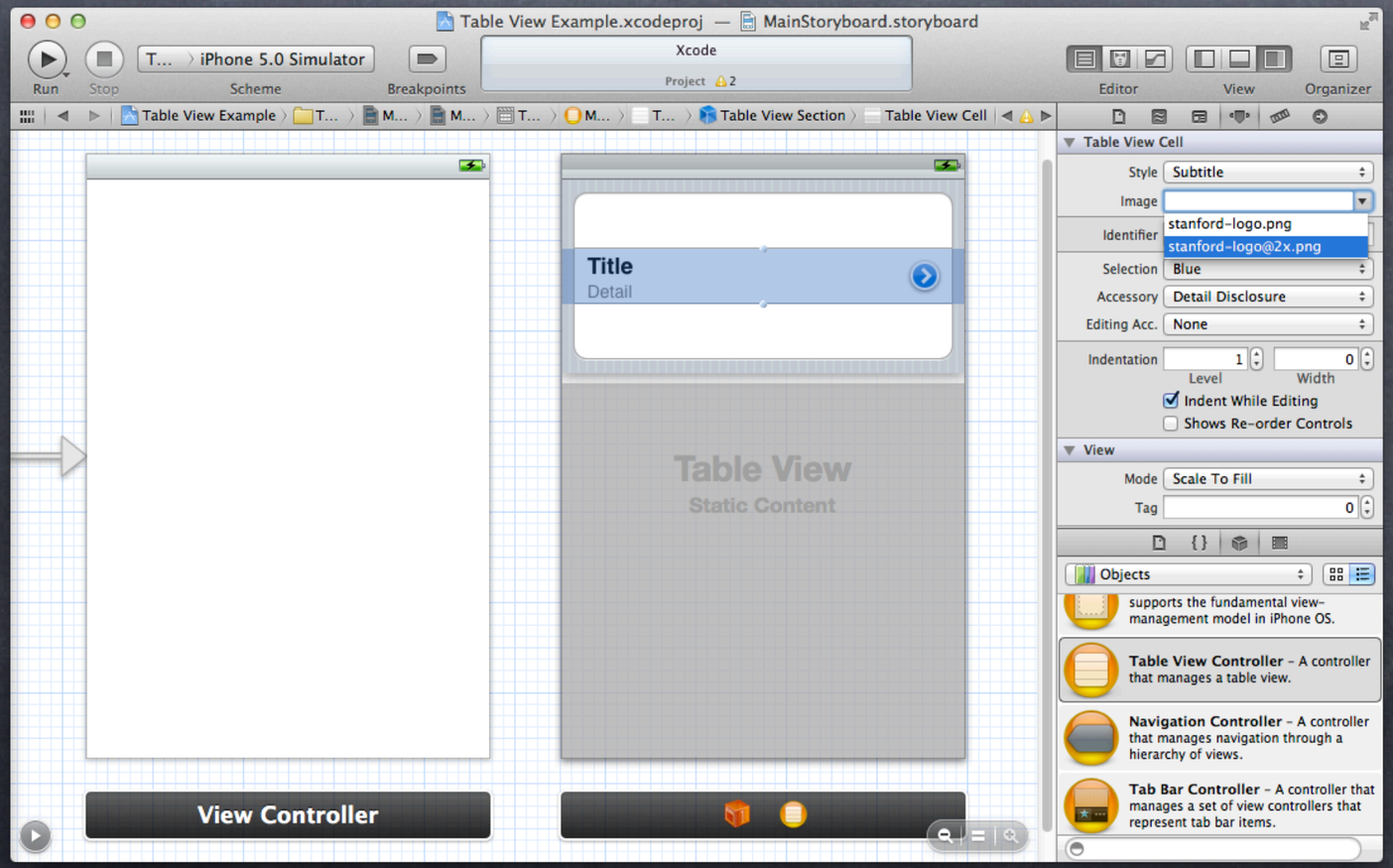


Creating Table View MVCs

- `UITableViewCell`

  Notice that some cell styles can have an image.

  You can set this in the code as well (more in a moment on this).

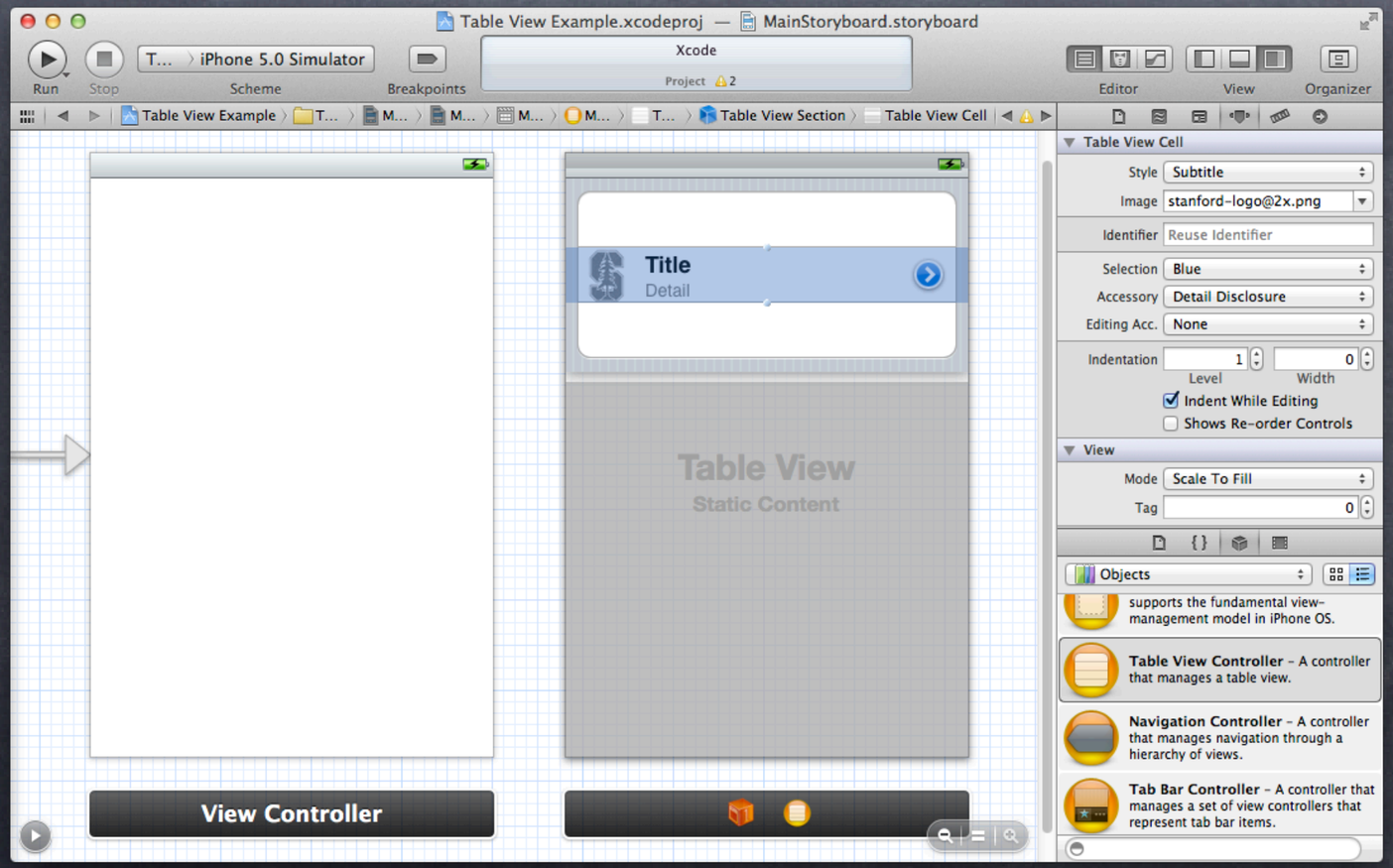


Creating Table View MVCs

- `UITableViewCell`

  Notice that some cell styles can have an image.

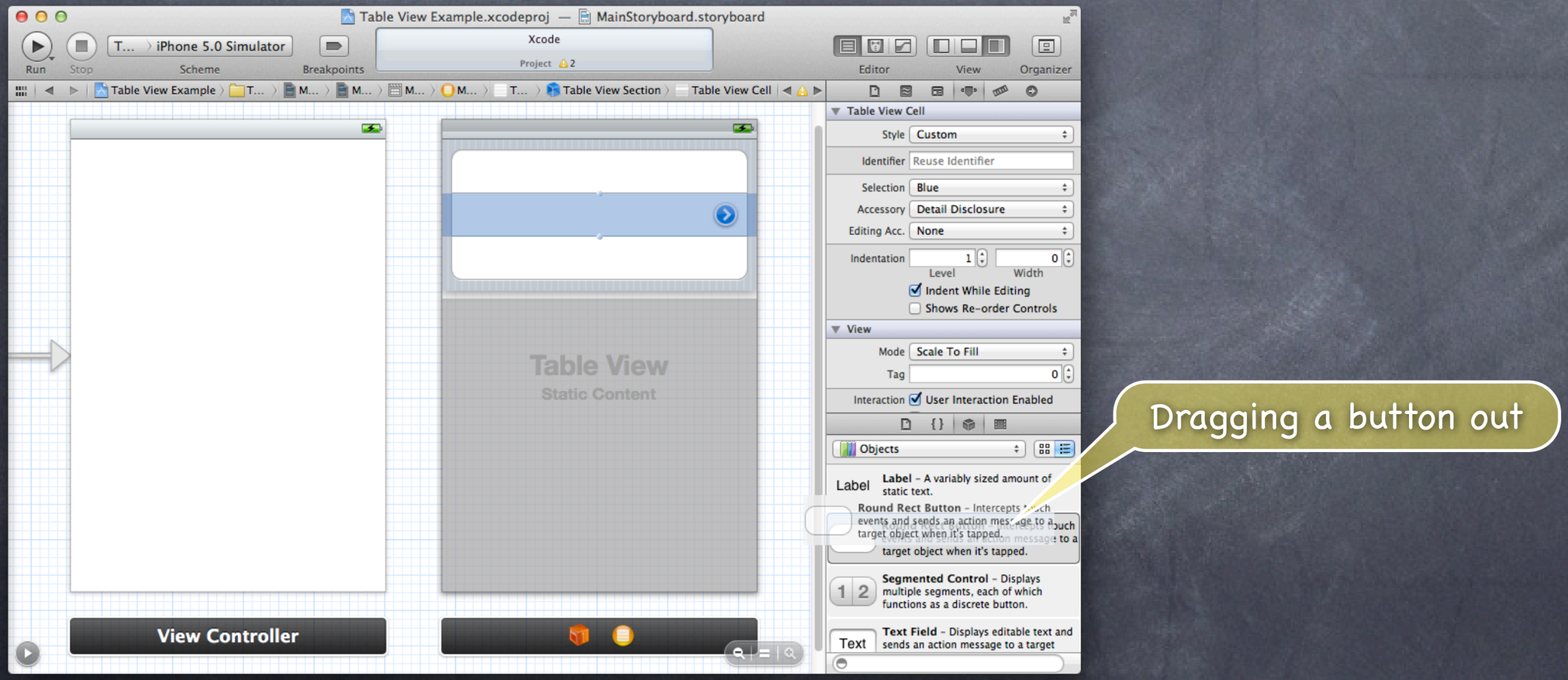
  You can set this in the code as well (more in a moment on this).



Creating Table View MVCs

- UITableViewCell

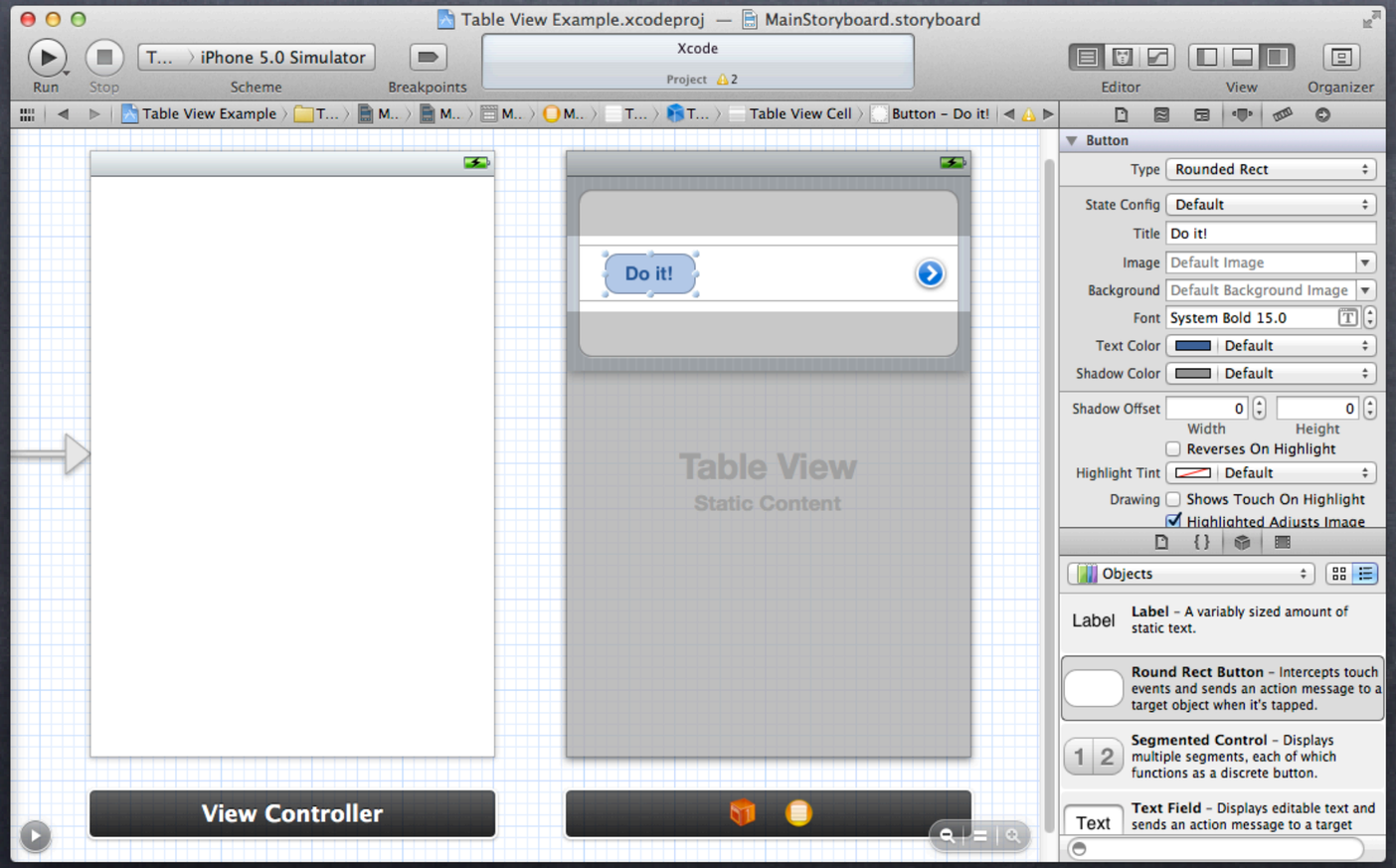
  In the Custom style, you can drag out views and wire them up as outlets!



Creating Table View MVCs

- `UITableViewCell`

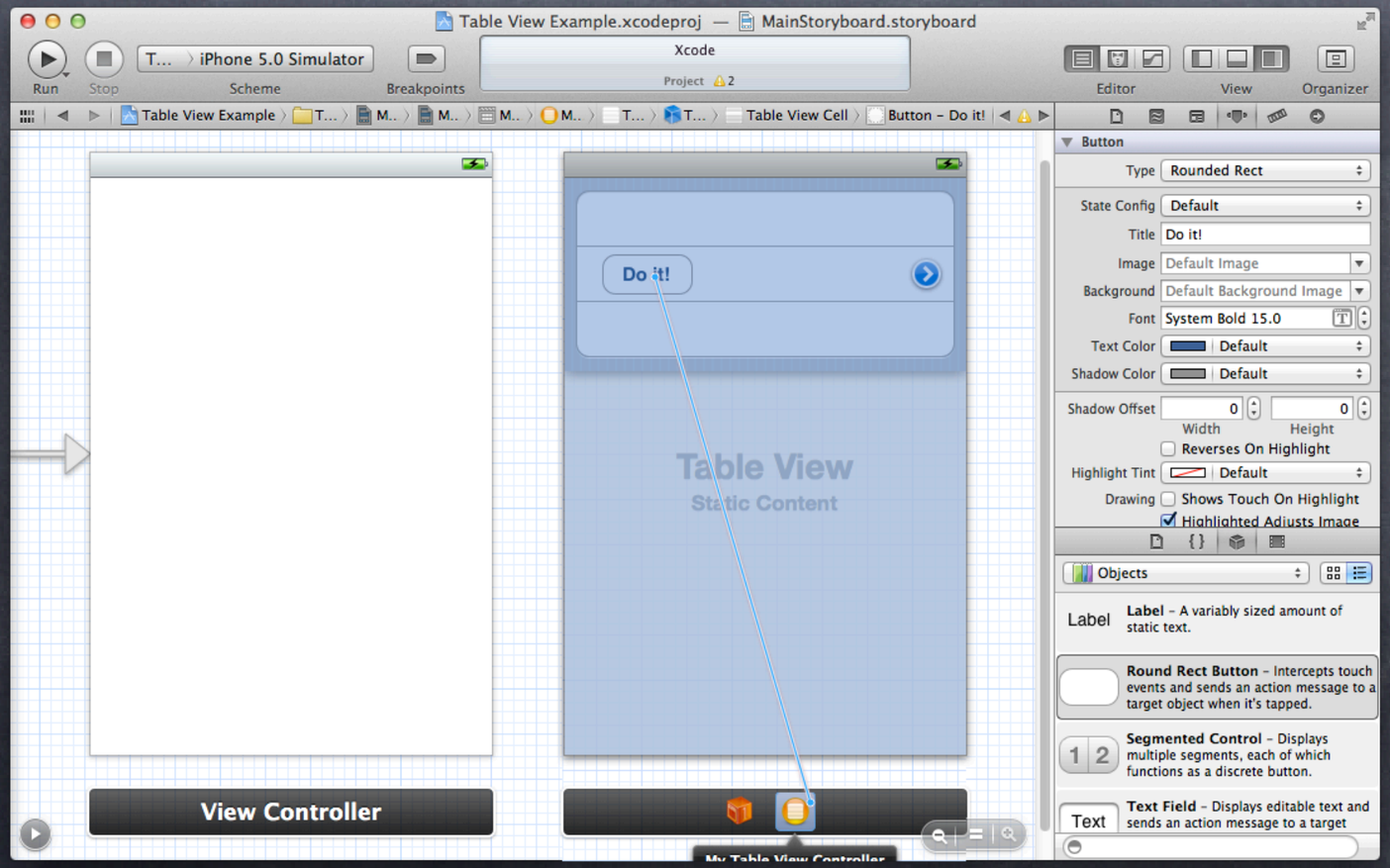
  In the Custom style, you can drag out views and wire them up as outlets!



Creating Table View MVCs

- `UITableViewCell`

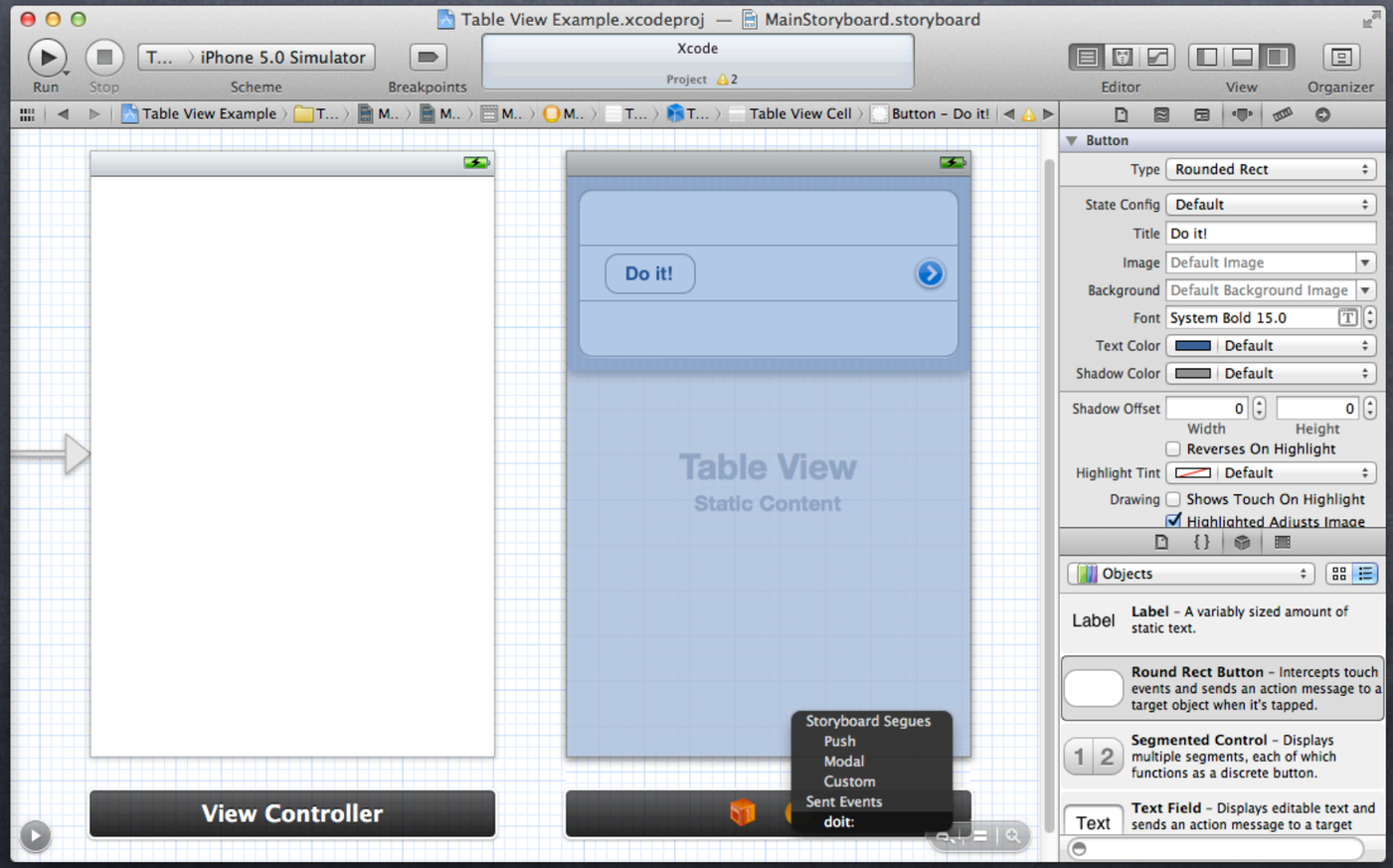
  In the Custom style, you can drag out views and wire them up as outlets!



Creating Table View MVCs

- `UITableViewCell`

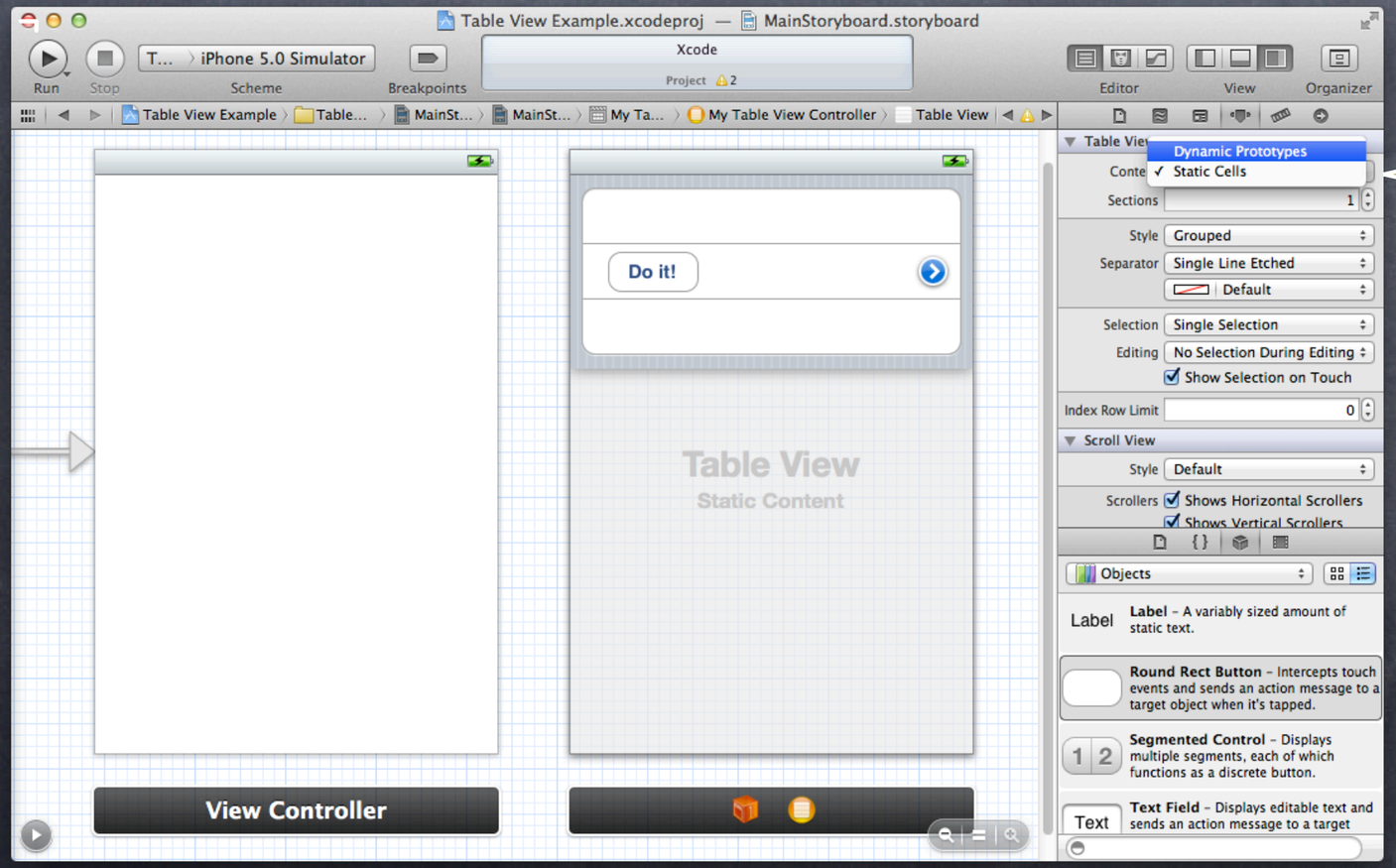
  In the Custom style, you can drag out views and wire them up as outlets!



Creating Table View MVCs

- UITableViewCell

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.

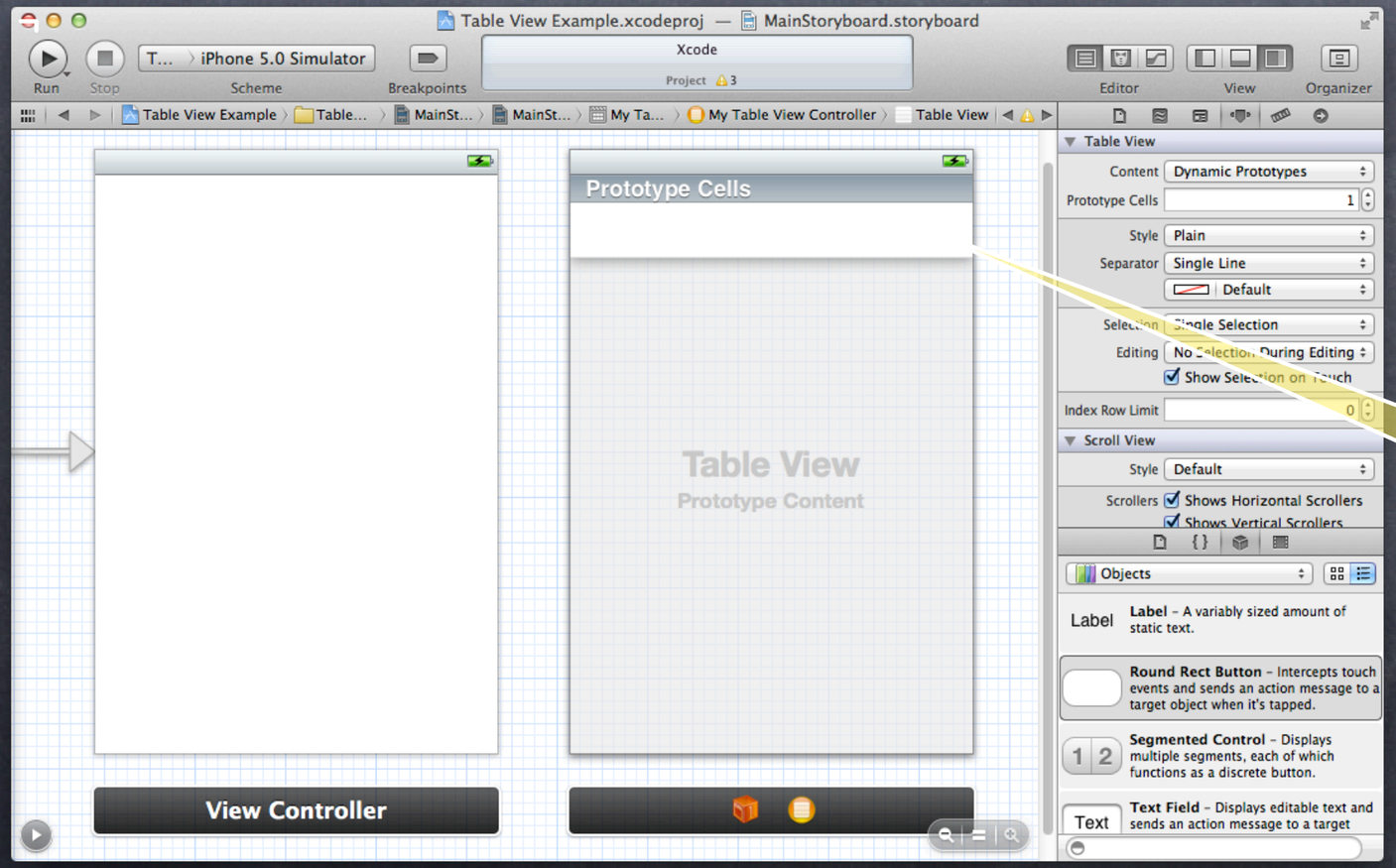


Switch to a Dynamic Table with Prototype Cells

Creating Table View MVCs

- UITableViewCell

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.

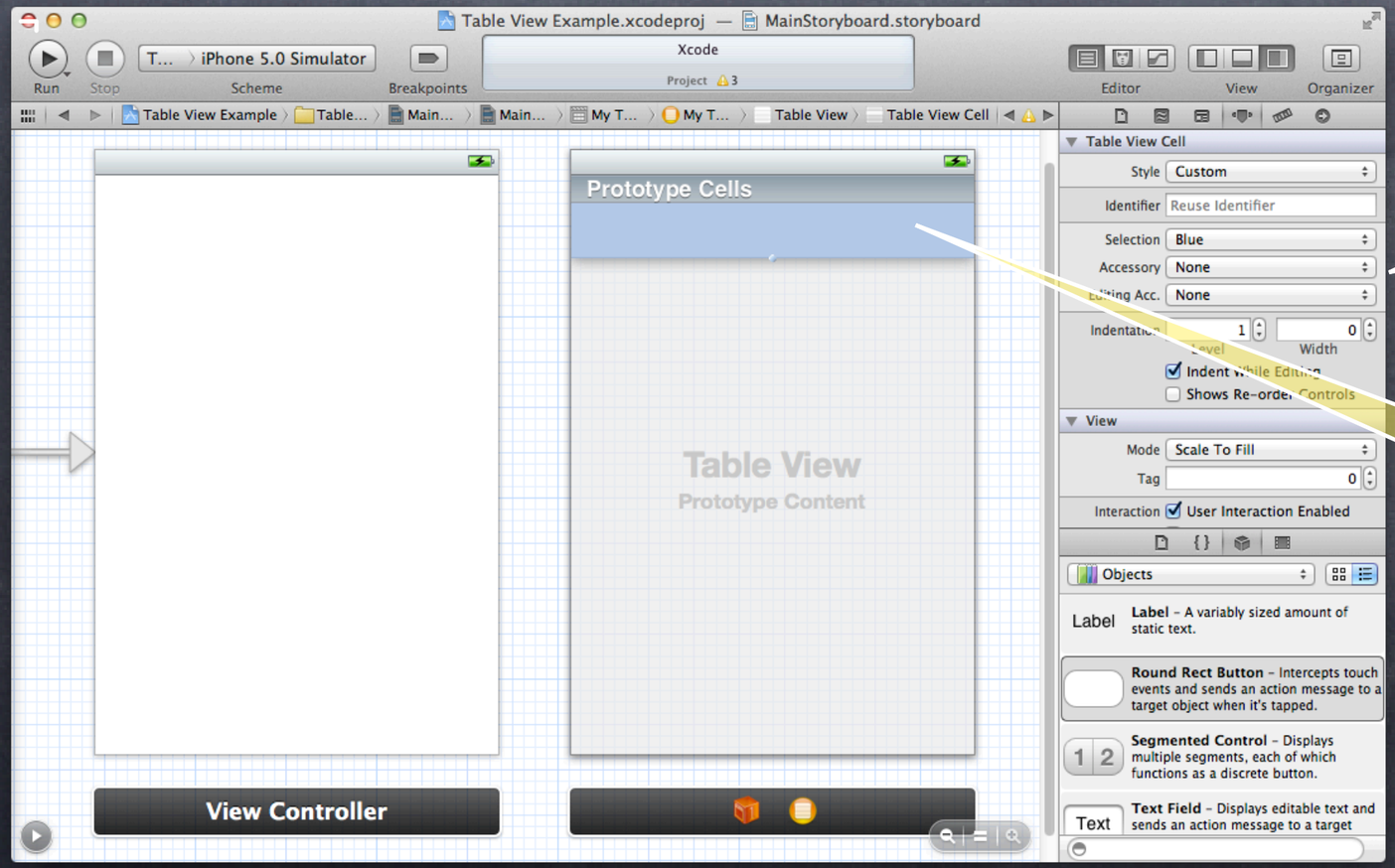


Now click on the Prototype to edit it. All cells in this table will be like this Prototype (though we'll set the contents to be different in code).

Creating Table View MVCs

- UITableViewCell

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.



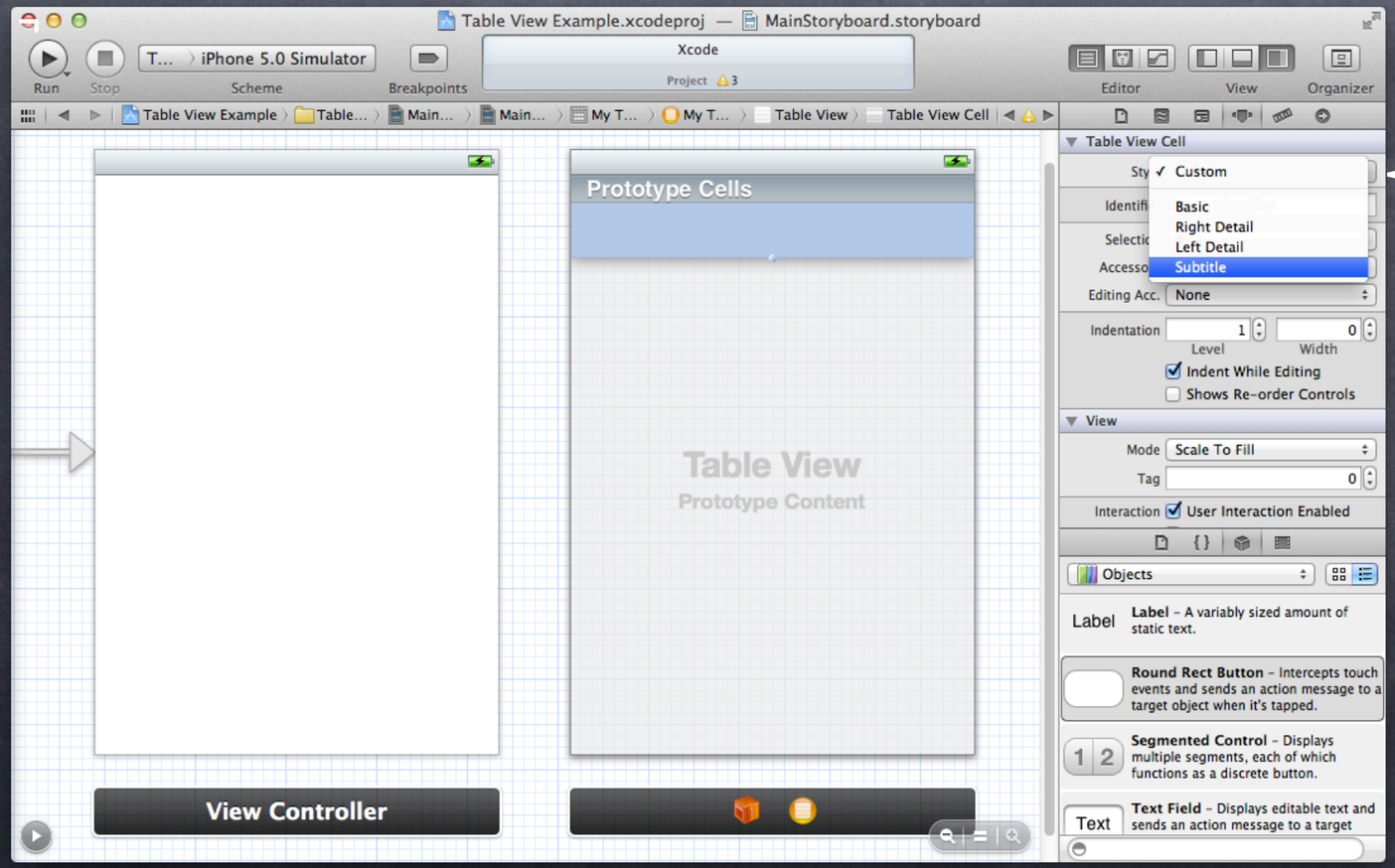
You should see Table View Cell properties appear in the Attributes Inspector.

Now click on the Prototype to edit it. All cells in this table will be like this Prototype (though we'll set the contents to be different in code).

Creating Table View MVCs

- UITableViewCell

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.

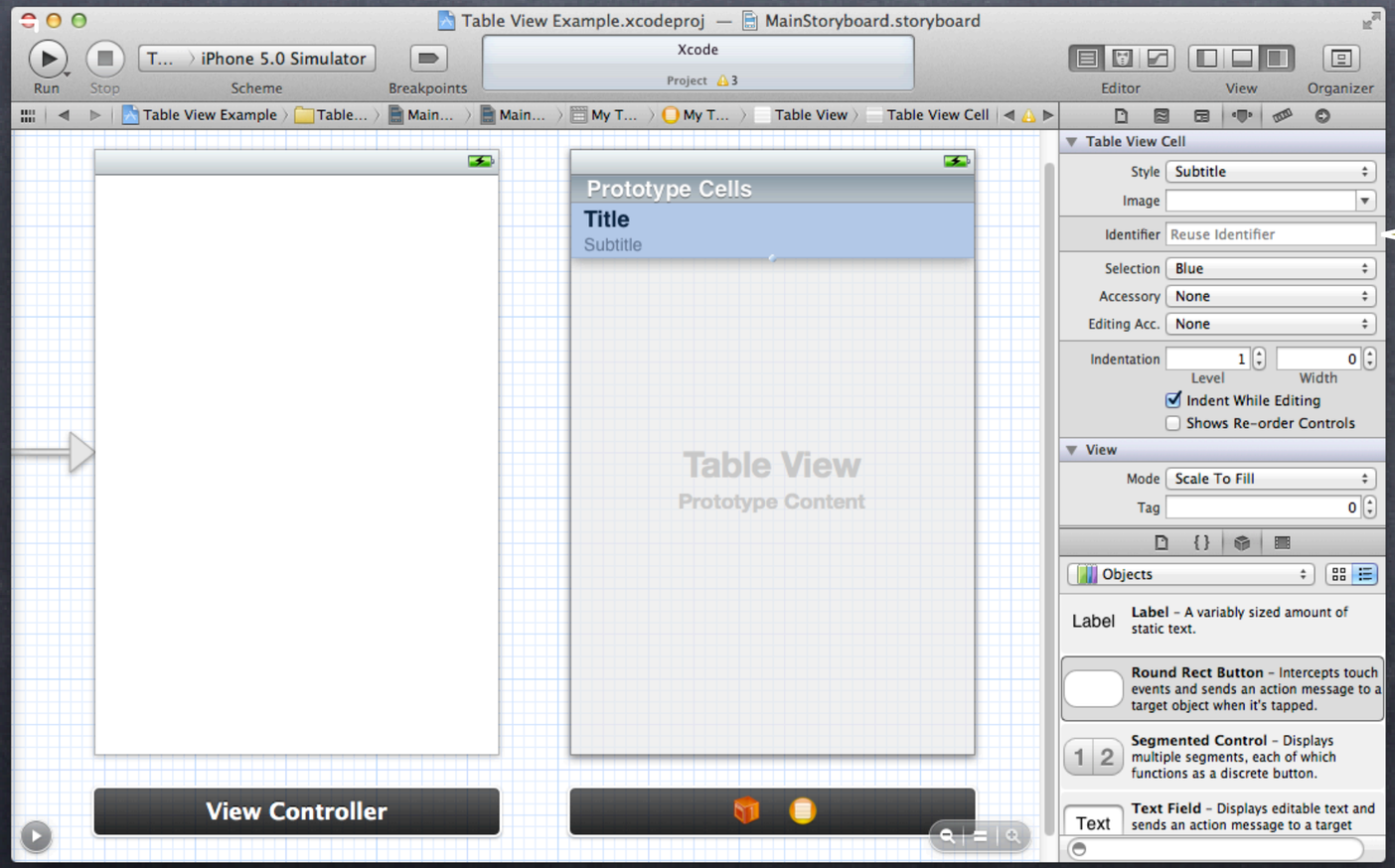


Let's change the Prototype's style to be Subtitle, for example.

Creating Table View MVCs

- UITableViewCell

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.

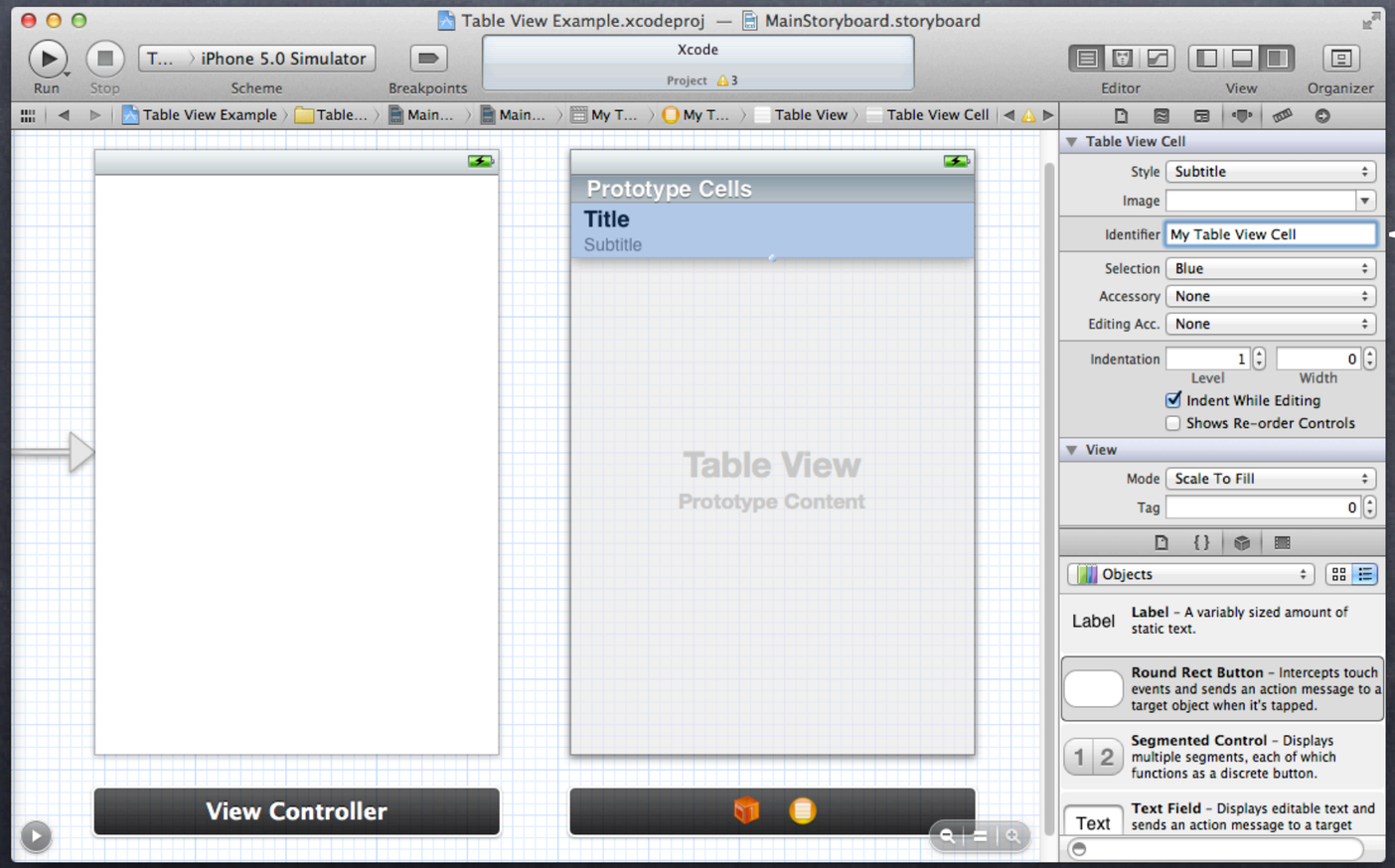


This is a very important field! It is the name that we will reference in our code to identify this prototype (more on this in a moment).

Creating Table View MVCs

- `UITableViewCell`

  All of the above examples were "static" cells (setup in the storyboard). Not a dynamic list.
  If you switch to dynamic mode, then the cell you edit is a "prototype" for all cells in the list.



Pick a name that is meaningful. "My Table View Cell" would probably not be that great. Something like "Photo Description" (if this were a list of photos) would be better.

UITableView Protocols

- How do we connect to all this stuff in our code?

  A `UITableView` has two important `@property`s: its `delegate` and its `dataSource`.

  The `delegate` is used to control how the table is displayed.

  The `dataSource` provides the data what is displayed inside the cells.

  Your `UITableViewController` is automatically set as the `UITableView`'s `delegate` & `dataSource`.

  Your `UITableViewController` subclass will also have a property that points to the `UITableView`:

```
@property (nonatomic, strong) UITableView *tableView;
```

- To be "dynamic," we need to be the `UITableView`'s `dataSource`

  Three important methods in this protocol:

  How many sections in the table?

  How many rows in each section?

  Give me a `UIView` to use to draw each cell at a given row in a given section.

  Let's cover the last one first …

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?
  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).
  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{


}
```

In a static table, you do not need to implement this method (though you can if you want to ignore what's in the storyboard).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?
  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).
  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{


}
```

NSIndexPath is just an object with two important properties for use with UITableView: row and section.

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?
  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).
  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    // set @propertys on the cell to prepare it to display
}
```

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@“My Table View Cell”];

    // set @propertys on the cell to prepare it to display
}
```

This MUST match what is in your storyboard if you want to use the prototype you defined there!

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"My Table View Cell"];

    // set @properties on the cell to prepare it to display
}
```

The cells in the table are actually reused.
When one goes off-screen, it gets put into a "reuse pool."
The next time a cell is needed, one is grabbed from the reuse pool if available.
If none is available, one will be put into the reuse pool if there's a prototype in the storyboard.
Otherwise this dequeue method will return `nil` (let's deal with that ...).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@“My Table View Cell”];
    if (!cell) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
                                      reuseIdentifier:@“My Table View Cell”];
    }

    // set @propertys on the cell to prepare it to display
}
```

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"My Table View Cell"];
    if (!cell) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
                                      reuseIdentifier:@"My Table View Cell"];
    }
    cell.textLabel.text = [self getMyDataForRow:indexPath.row inSection:indexPath.section];
    return cell;
}
```

There are obviously other things you can do in the cell besides setting its text (detail text, image, checkmark, etc.).

UITableViewDataSource

- How does a dynamic table know how many rows there are?
  And how many sections, too, of course?
  ```
  - (NSInteger)numberOfSectionsInTableView:(UITableView *)sender;
  - (NSInteger)tableView:(UITableView *)sender numberOfRowsInSection:(NSInteger)section;
  ```
- Number of sections is 1 by default
  In other words, if you don't implement `numberOfSectionsInTableView:`, it will be 1.
- No default for number of rows in a section
  This is a required method in this protocol (as is `tableView:cellForRowAtIndexPath:`).
- What about a static table?
  Do not implement these `dataSource` methods for a static table.
  `UITableViewController` will take care of that for you.

UITableViewDataSource

- There are a number of other methods in this protocol
  But we're not going to cover them today.
  They are mostly about getting the headers and footers for sections.
  And about dealing with editing the table (moving/deleting/inserting rows).

UITableViewDelegate

- All of the above was the UITableView's dataSource
  But UITableView has another protocol-driven delegate called its delegate.
- The delegate controls how the UITableView is displayed
  Not what it displays (that's the dataSource's job).
- Common for dataSource and delegate to be the same object
  Usually the Controller of the MVC in which the UITableView is part of the (or is the entire) View.
- The delegate also lets you observe what the table view is doing
  Especially responding to when the user selects a row.
  We often will use segues when this happens, but we can also track it directly.

Table View "Target/Action"

- UITableViewDelegate method sent when row is selected

  This is sort of like "table view target/action"

  You might use this to update a detail view in a split view if master is a table view

```
- (void)tableView:(UITableView *)sender didSelectRowAtIndexPath:(NSIndexPath *)path
{
    // go do something based on information
    // about my data structure corresponding to indexPath.row in indexPath.section
}
```

- Lots and lots of other delegate methods

  will/did methods for both selecting and deselecting rows

  Providing UIView objects to draw section headers and footers

  Handling editing rows (moving them around with touch gestures)

  willBegin/didEnd notifications for editing.

  Copying/pasting rows.

Table View Segues

- You can segue when a row is touched, just like from a button
  Segues will call `prepareForSegue:sender:` with the chosen `UITableViewCell` as `sender`.
  You can tailor whatever data the MVC needs to whichever cell was selected.

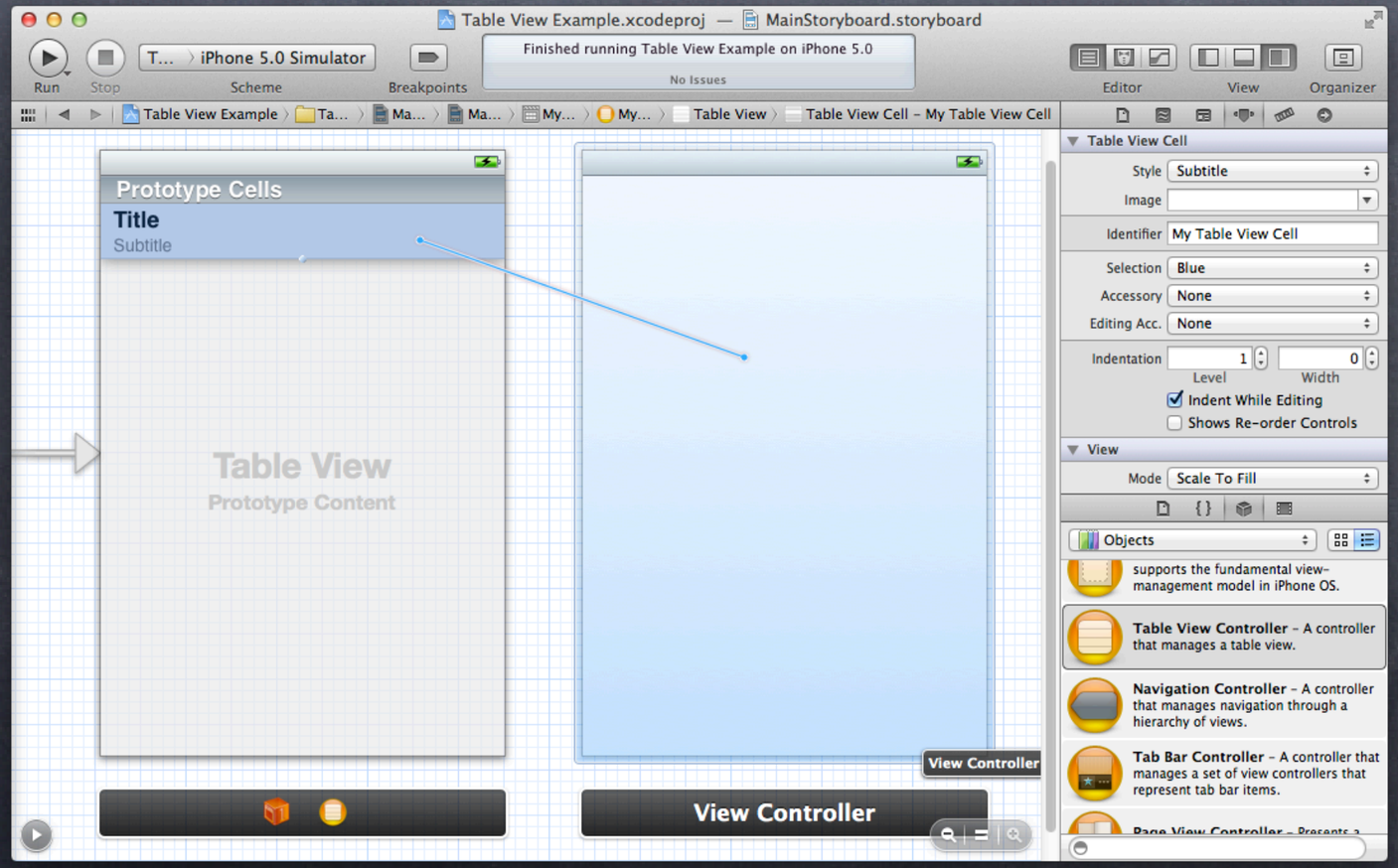


Table View Segues

- You can segue when a row is touched, just like from a button
  Segues will call `prepareForSegue:sender:` with the chosen `UITableViewCell` as `sender`.
  You can tailor whatever data the MVC needs to whichever cell was selected.

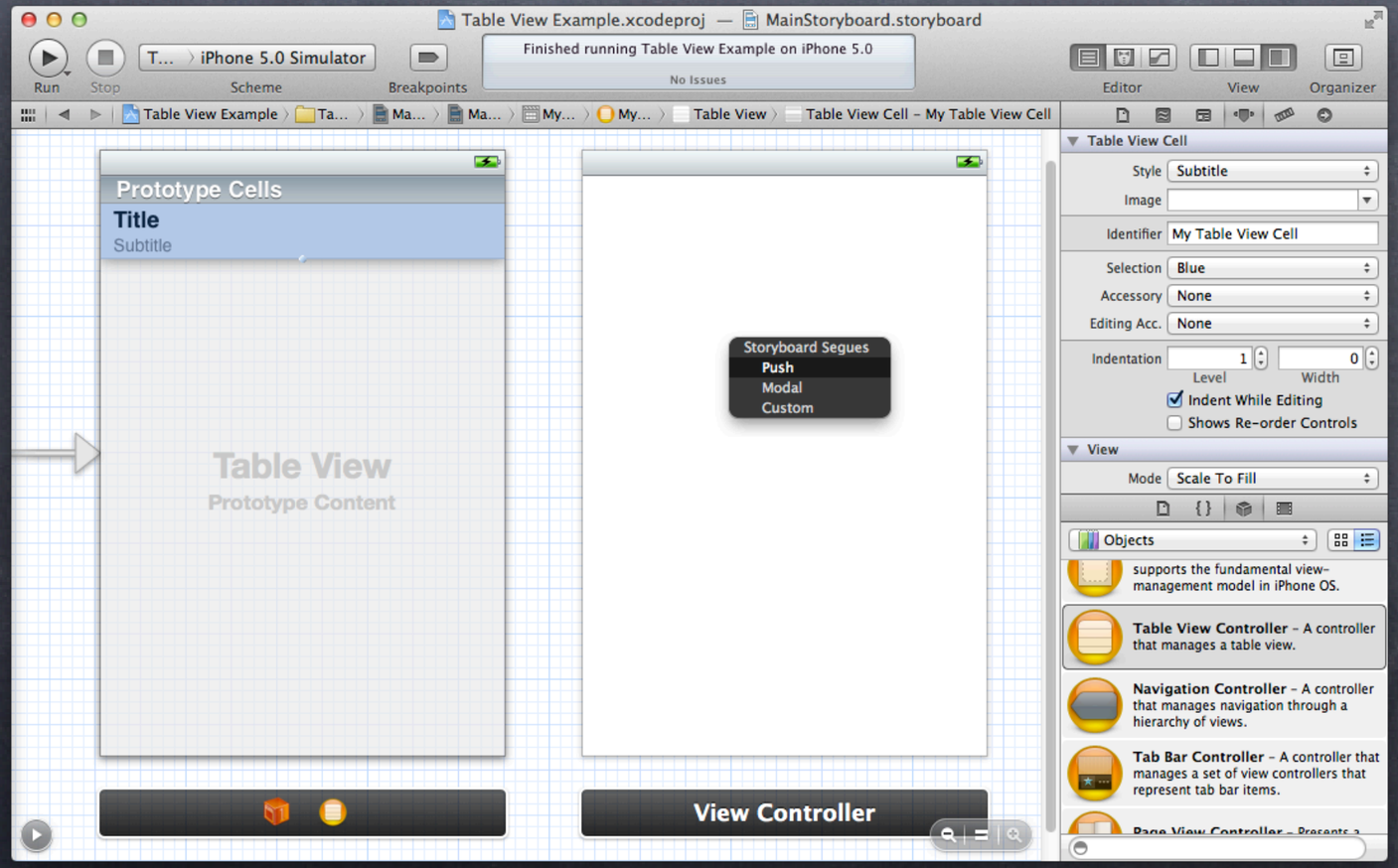


Table View Segue

- This works whether dynamic or static

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];
    // prepare segue.destinationController to display based on information
    // about my data structure corresponding to indexPath.row in indexPath.section
}
```

UITableView

- What if your Model changes?

```
- (void)reloadData;
```

Causes the table view to call `numberOfSectionsInTableView:` and `numberOfRowsInSection:` all over again and then `cellForRowAtIndexPath:` on each visible cell.

Relatively heavyweight obviously, but if your entire data structure changes, that's what you need.

If only part of your Model changes, there are lighter-weight reloaders, for example …

```
- (void)reloadRowsAtIndexPaths:(NSArray *)indexPaths
              withRowAnimation:(UITableViewRowAnimation)animationStyle;
```

- There are dozens of other methods in `UITableView`

Setting headers and footers for the entire table

Controlling the look (separator style and color, default row height, etc.)

Getting cell information (cell for index path, index path for cell, visible cells, etc.)

Scrolling to a row

Selection management (allows multiple selection, getting the selected row, etc.)

Moving, inserting and deleting rows, etc.

Demo

- Favorite Graphs

  Add a feature to homework 3.

  Put an "Add to Favorites" button in graph view.

  Add a "Favorites" bar button which brings up a list of the favorite graphs.

- Watch for ...

  Deciding and advertising that an `id` parameter in a public API is a "property list".
  (we just say it is so and then be sure not to break our promise to users of our public API).

  Storing a property list in `NSUserDefaults`.

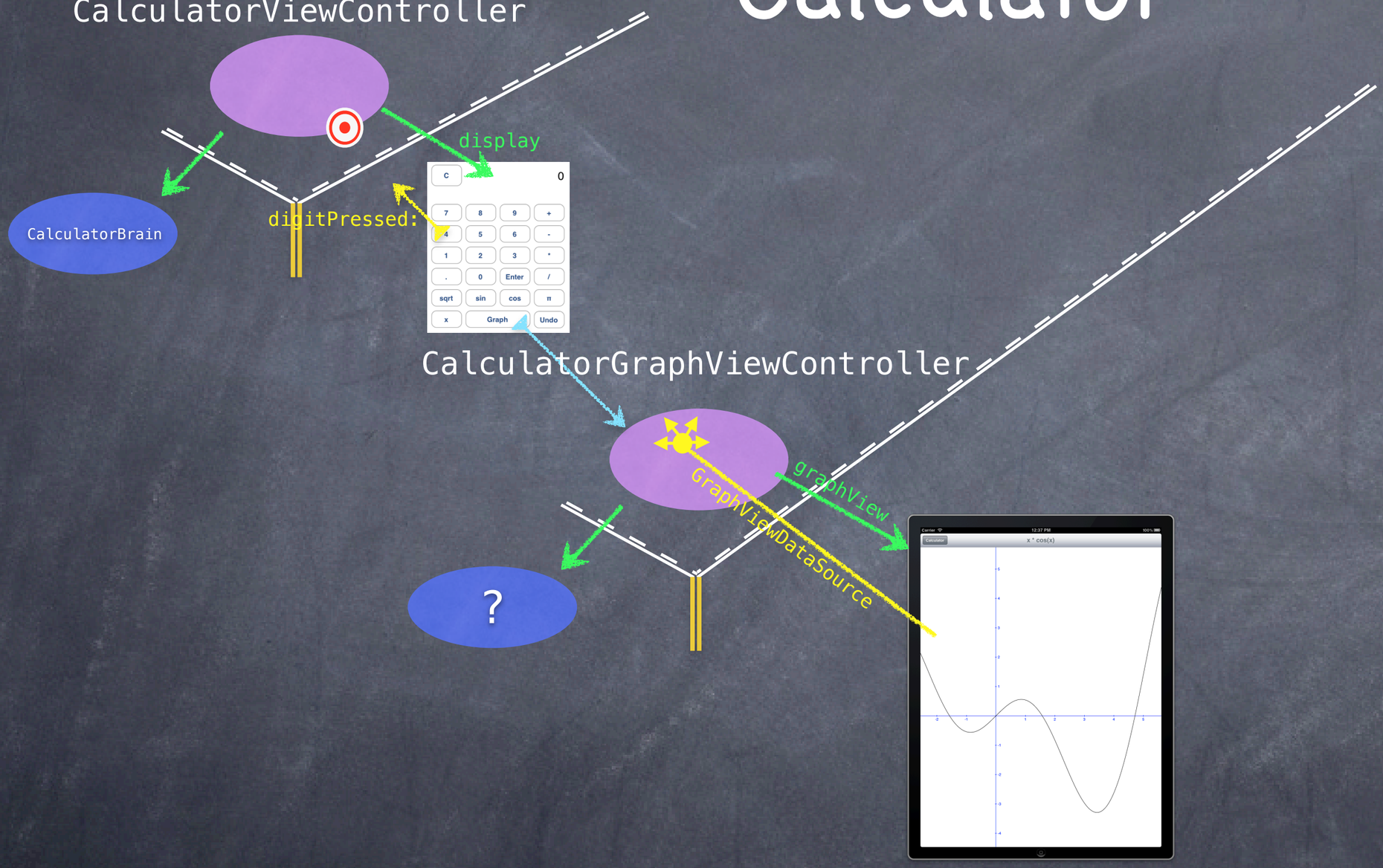
  Building a table-view-based view controller subclass.

  Implementing the `UITableView dataSource` protocol to show a dynamic list of data (graphs).

  Using a delegate to report the results of a popover to the object which puts the popover up.

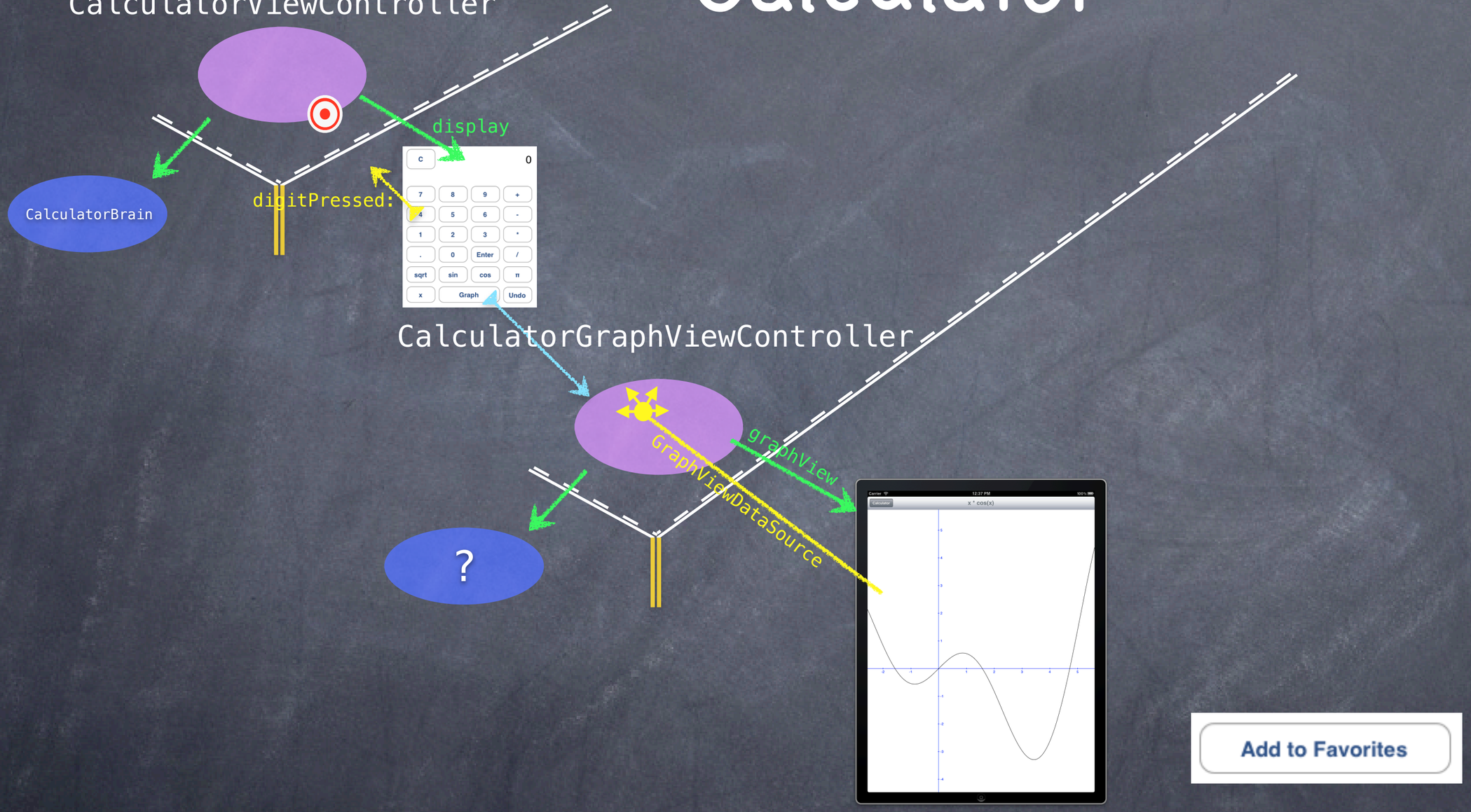
  Calling `popViewControllerAnimated:` (and/or `dismissPopoverAnimated:`) from delegate method.

Calculator


CalculatorViewController
display
digitPressed:
CalculatorBrain
C
0
7
8
9
+
4
5
6
-
1
2
3
*
.
0
Enter
/
sqrt
sin
cos
π
x
Graph
Undo
CalculatorGraphViewController
graphView
GraphViewDataSource
?

Calculator

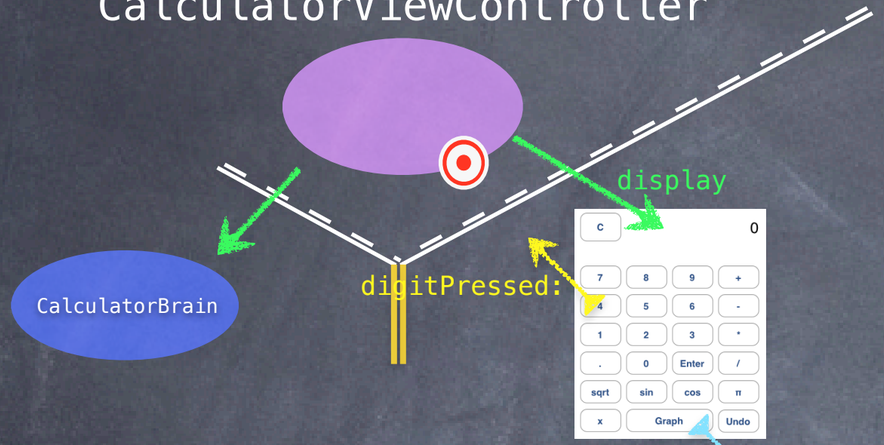
CalculatorViewController



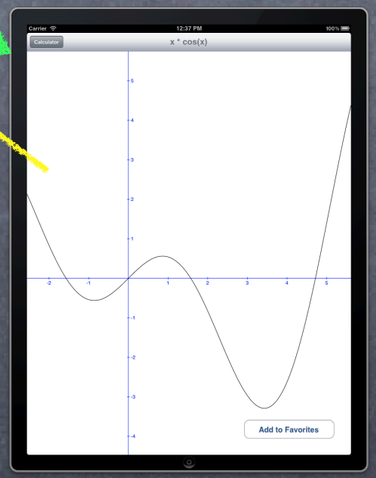
Add to Favorites

Calculator

CalculatorViewController



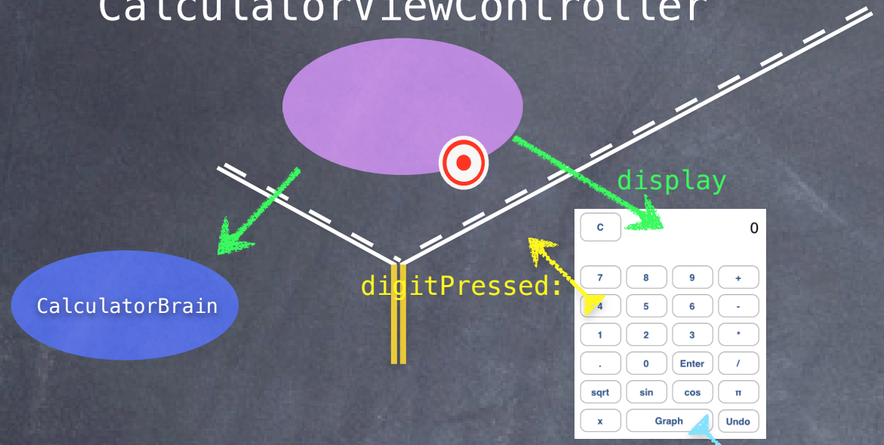
CalculatorGraphViewController



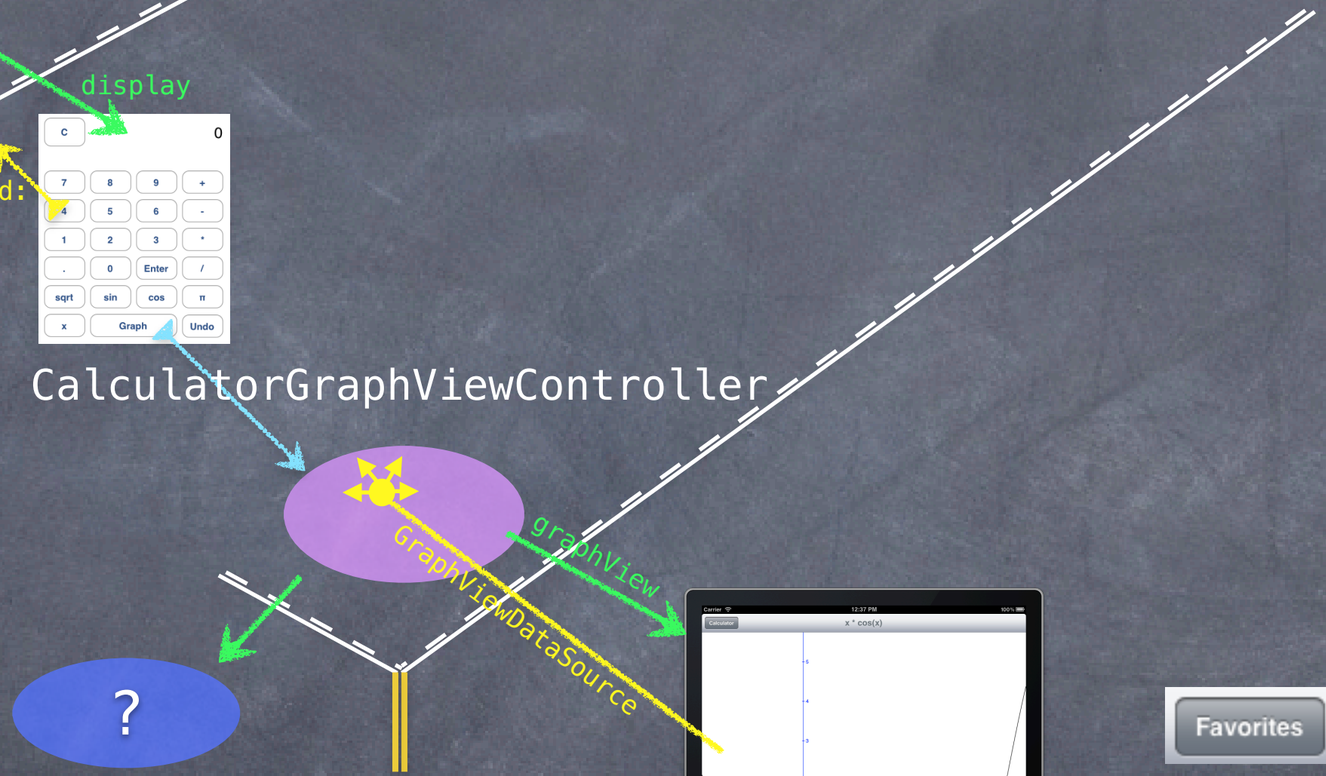
Stanford CS193p
Fall 2011

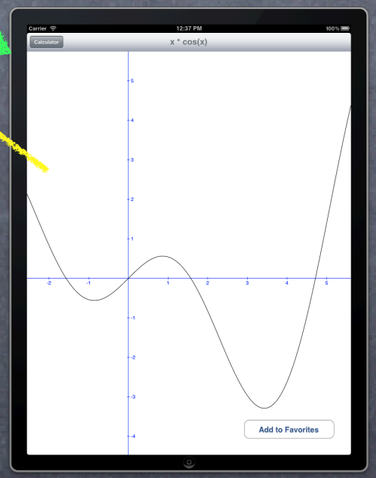
Calculator

CalculatorViewController



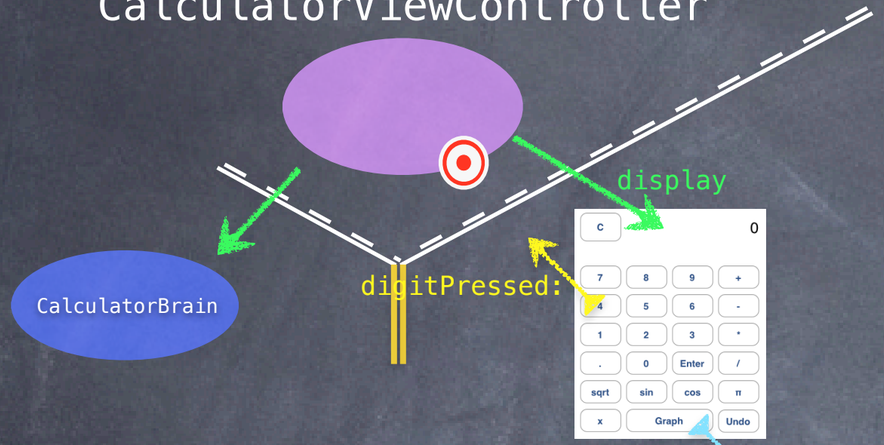
CalculatorGraphViewController



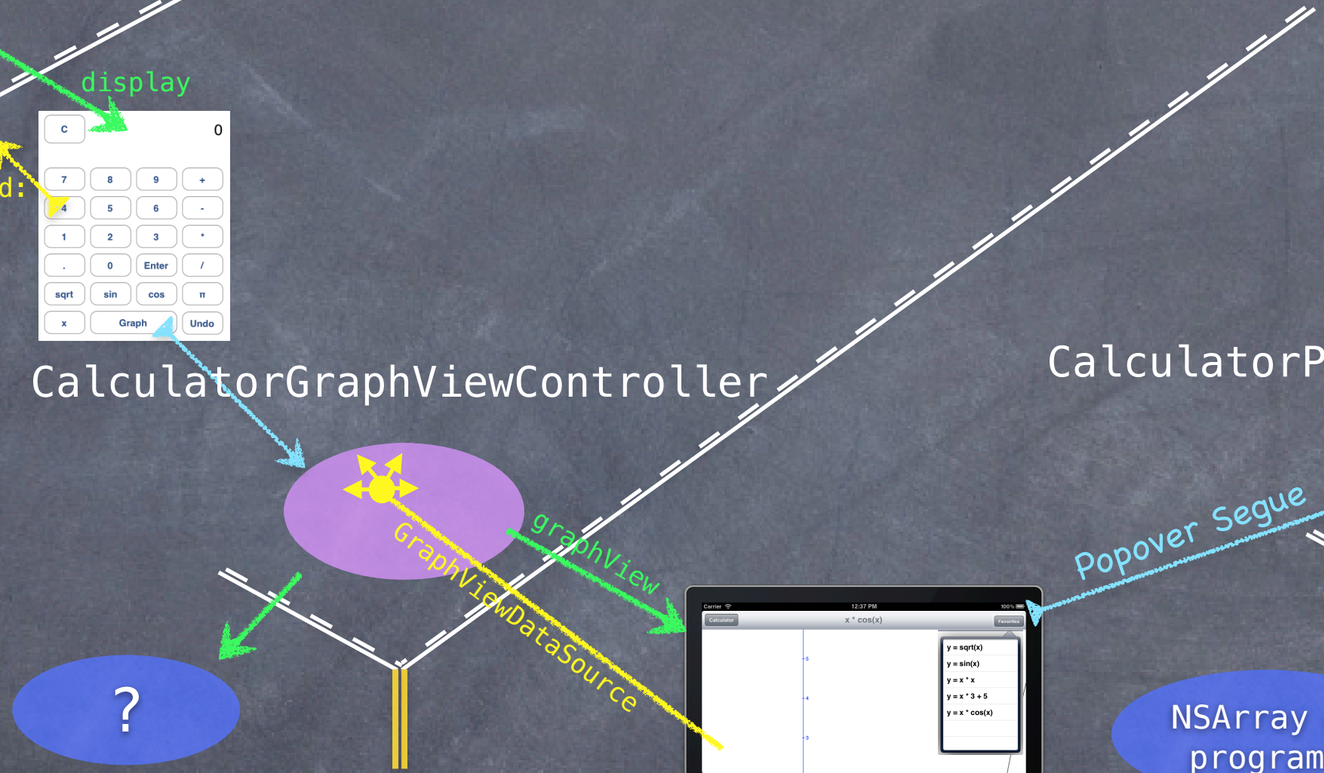


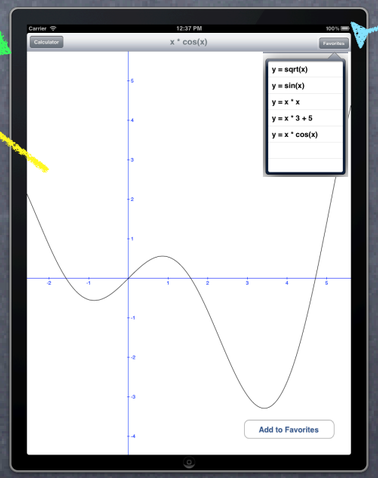
Calculator

CalculatorViewController

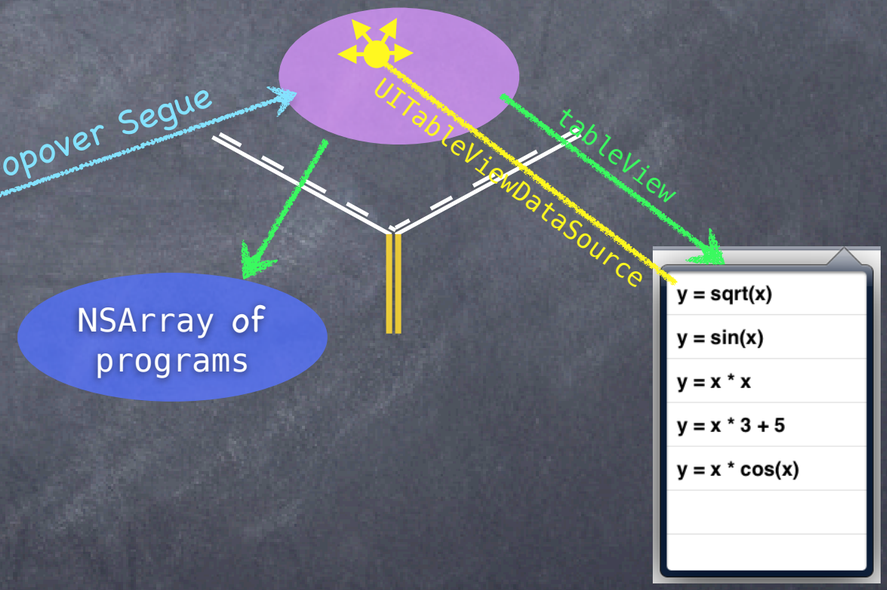


CalculatorGraphViewController



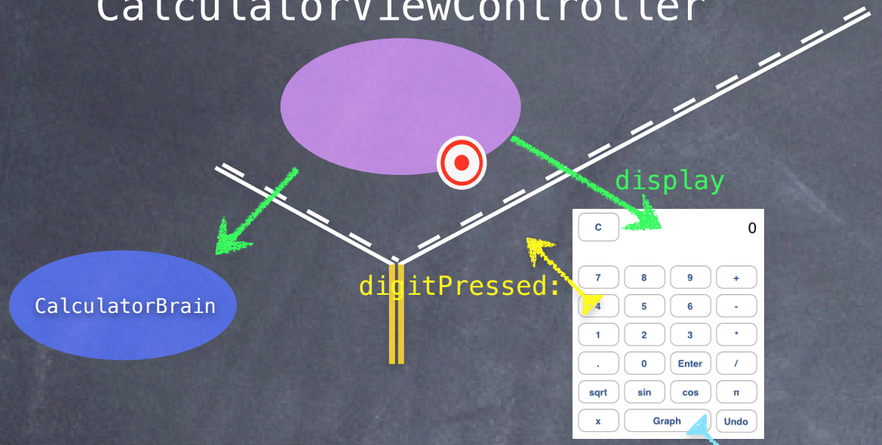


CalculatorProgramsTableViewController



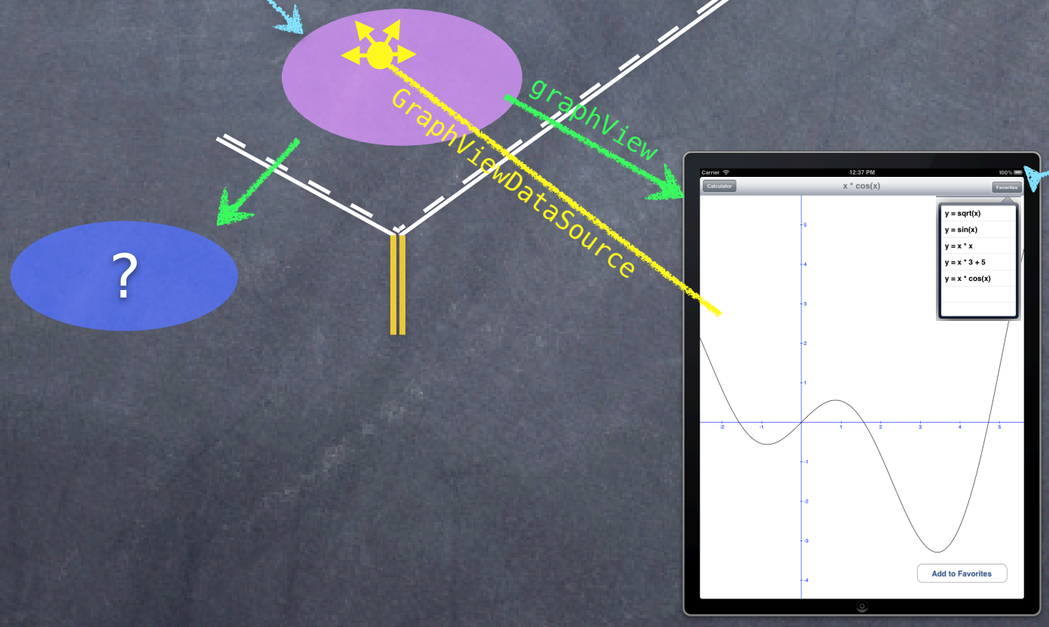
Calculator

CalculatorViewController

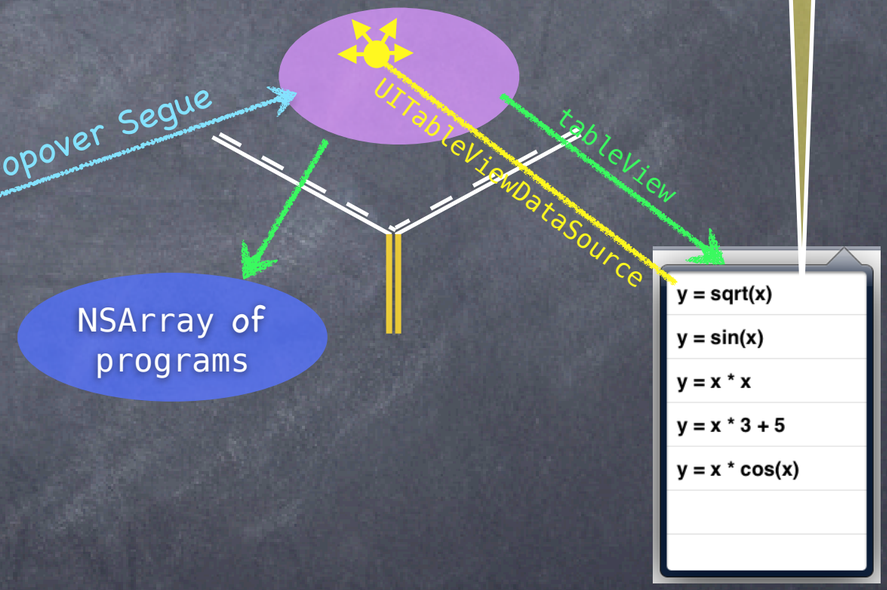


When someone clicks in this table, we want to update the graph. How can we do that?

CalculatorGraphViewController



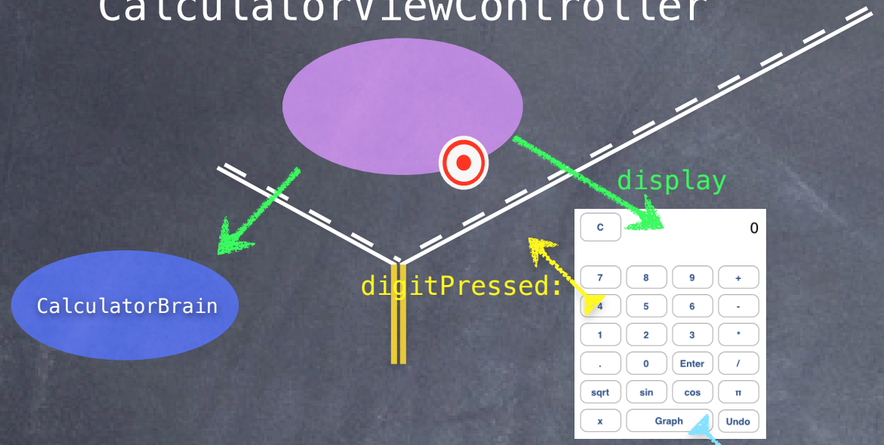
CalculatorProgramsTableViewController



Stanford CS193p
Fall 2011

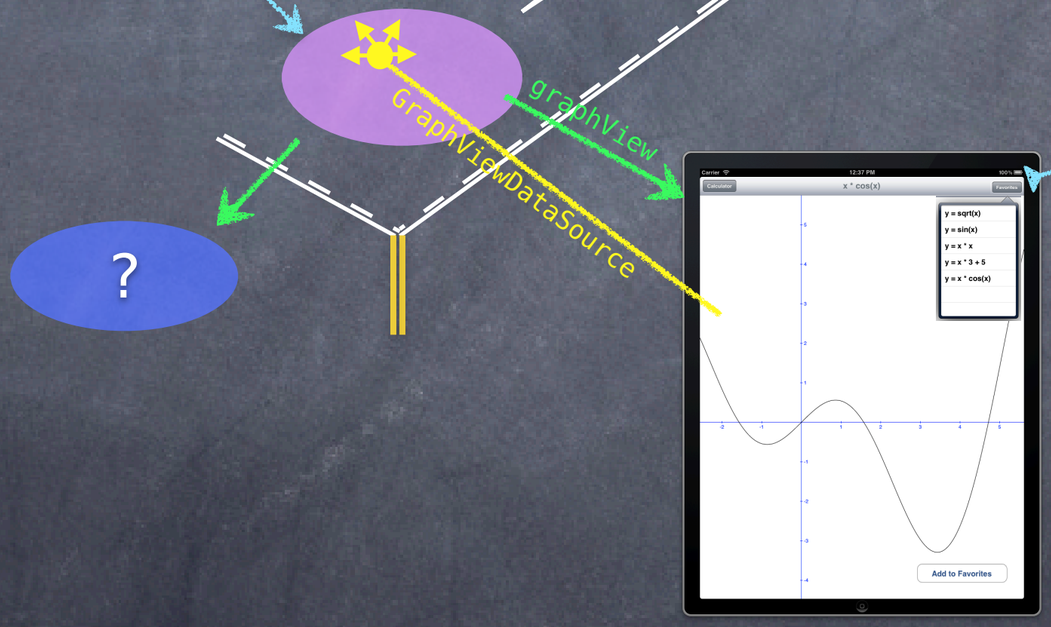
Calculator

CalculatorViewController

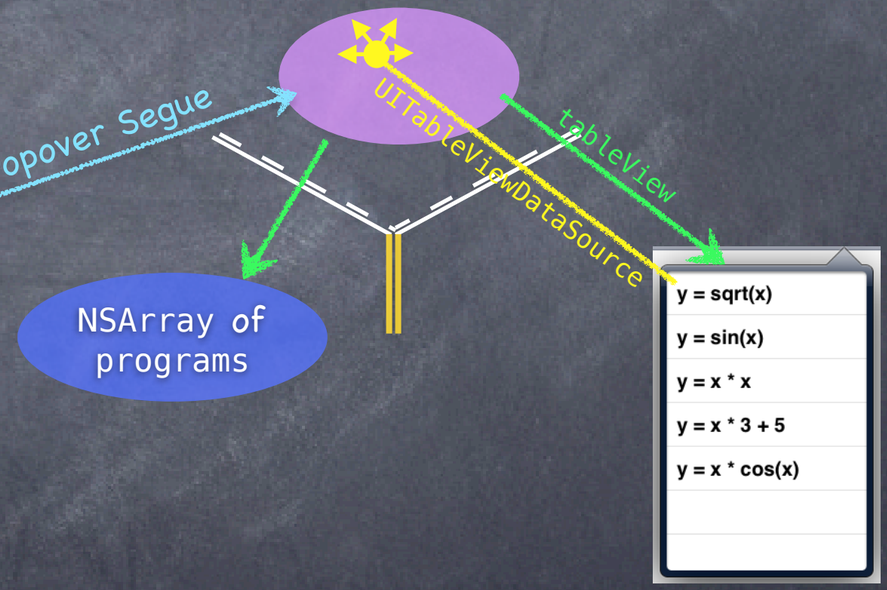


We CANNOT directly ask this Graph Controller to do it because we are (indirectly) part of that Controller's View.

CalculatorGraphViewController

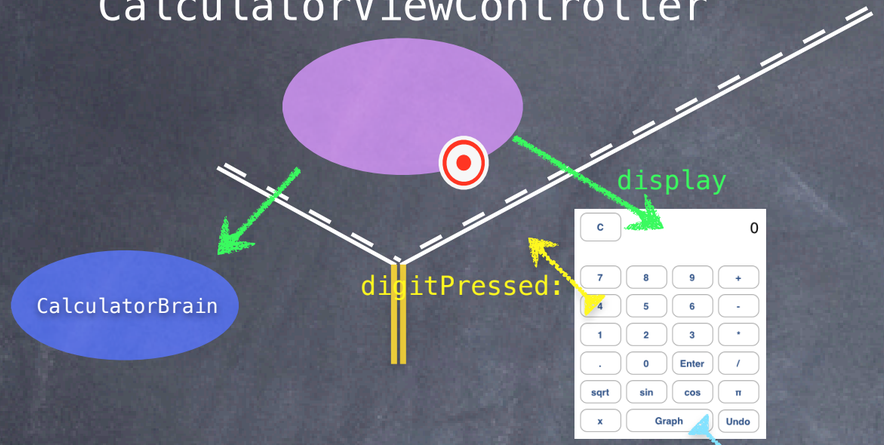


CalculatorProgramsTableViewController



Calculator

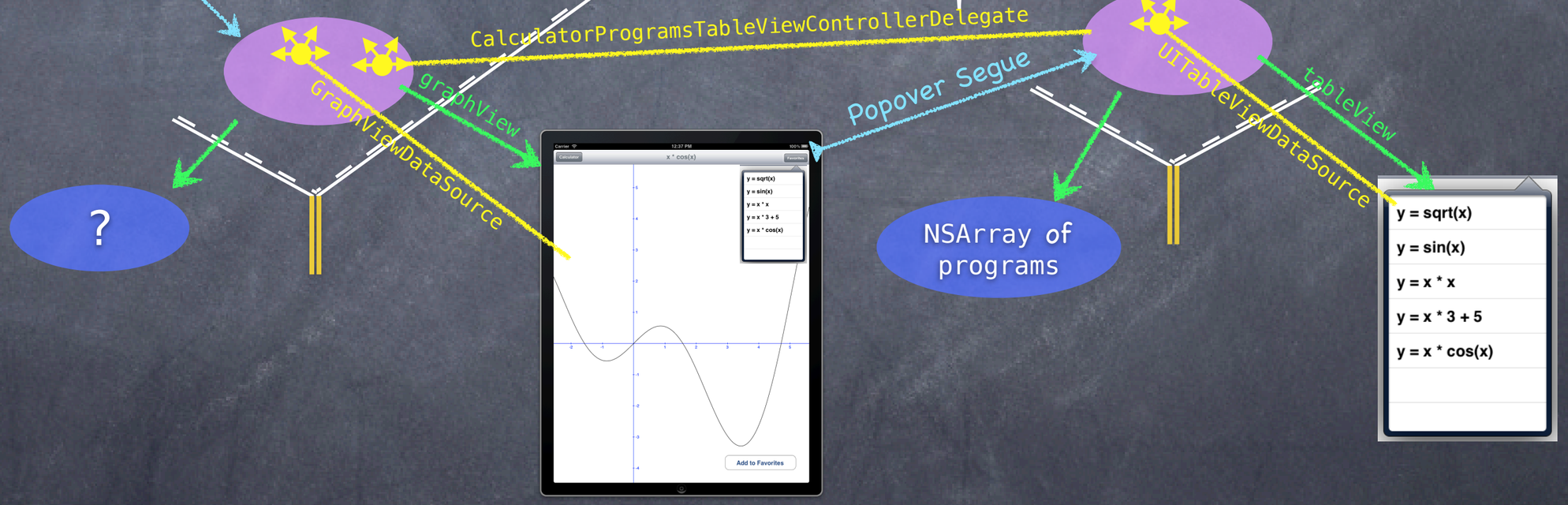
CalculatorViewController



We do it in the normal way a View can talk back to its Controller: delegation.

CalculatorGraphViewController

CalculatorProgramsTableViewController



Steps to Delegate

- 5 Steps
  1. Create the `@protocol`
  2. Add `delegate @property` to delegator's public `@interface`
  3. Use `delegate` property inside delegator's implementation
  4. Set the `delegate` property somewhere inside the delegate's `@implementation`
  5. Implement the protocol's method(s) in the delegate (include <> on `@interface`)

Coming Up

- Next Lecture
  Threads
  Persistence
- Section
  No section this week.