How to Make a Simple Mac App on OS X 10.7 Tutorial: Part 1/3

Ernesto García on August 27, 2012

This is a post by iOS Tutorial Team Member Ernesto García, a Mac and iOS developer founder of CocoaWithChurros.

It's a good time to be an iOS developer. Not only can you release your apps to both the iPhone and iPad App Stores, but you also have the foundational skills to become a Mac developer, since iOS development and Mac development are quite similar!

If you're an iOS developer and you're curious about learning the basics of becoming a Mac developer so you can start migrating your iOS apps to the desktop, this tutorial is for you.

In this tutorial, you're going to build your first Mac application, specifically a Mac version of the app we created in the How To Create A Simple iPhone App tutorial.

If you've followed that tutorial, you will be familiar with most of the steps on this one, and you will be able to see the main differences between iOS and Mac programming.

If you haven't followed it, don't worry. It's not required in order to read and understand this one – we'll guide you along the way step by step.

It's time to show these Scary Bugs in a Mac Application! :]

While making this app, you'll learn the following topics:

- How to create a Mac App in XCode
- Learn the basic structure of a Mac App
- Learn the main differences between OSX and iOS
- How to use Table Views – including adding and deleting rows
- How to use a text field, a button and an image view
- How to select an image from your hard drive, or capture a picture from your computer's camera
- How to handle window resizing

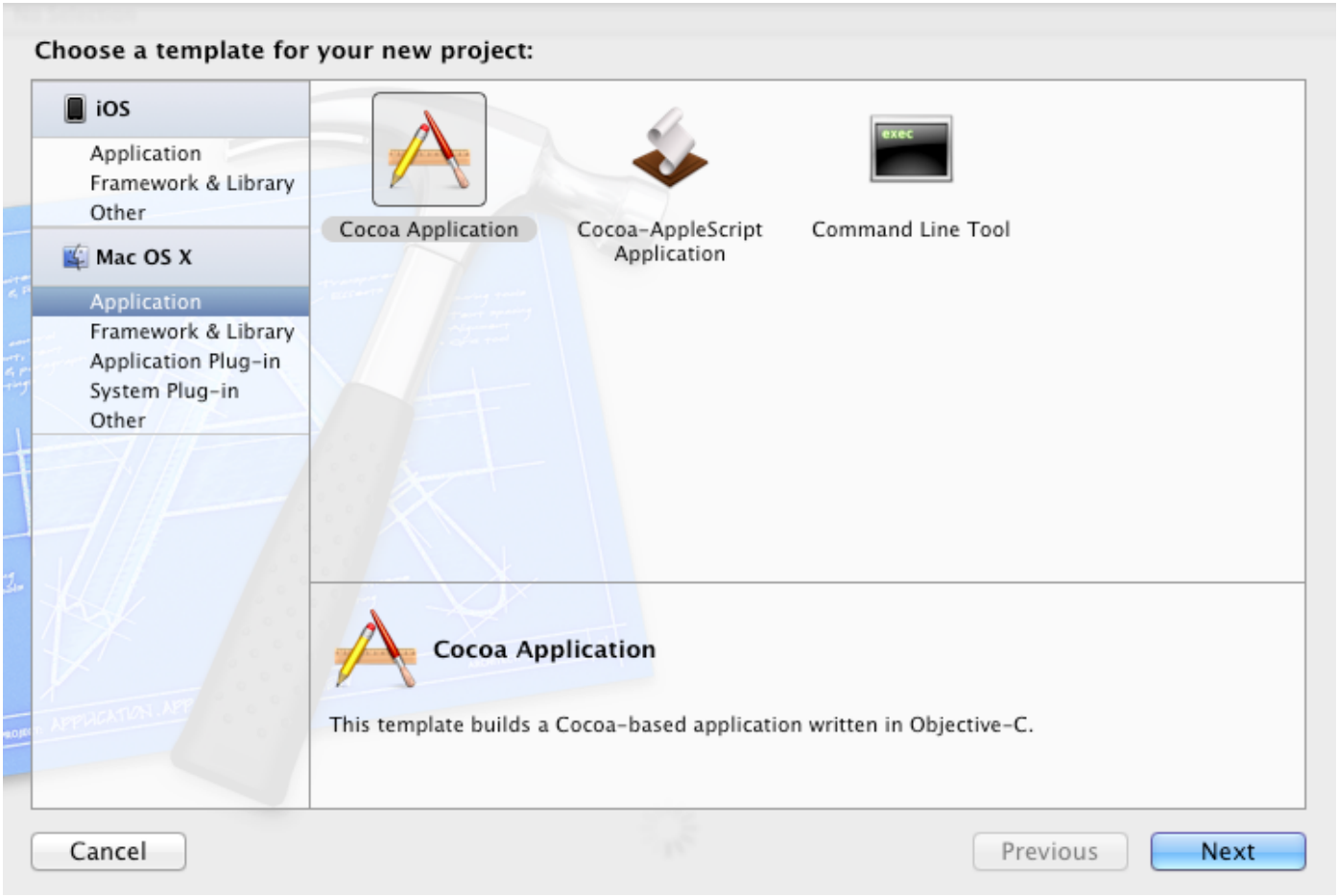
This tutorial is for beginner Mac Developers, but it assumes that you are familiar with Objective-C programming and with XCode. Knowledge of iOS programming is recommended to follow this tutorial, but not mandatory.

In this first part of this three-part series, we'll cover how to load your model with a list of bugs and display them in a table view. (Jump to Part two or Part three)

Getting Started

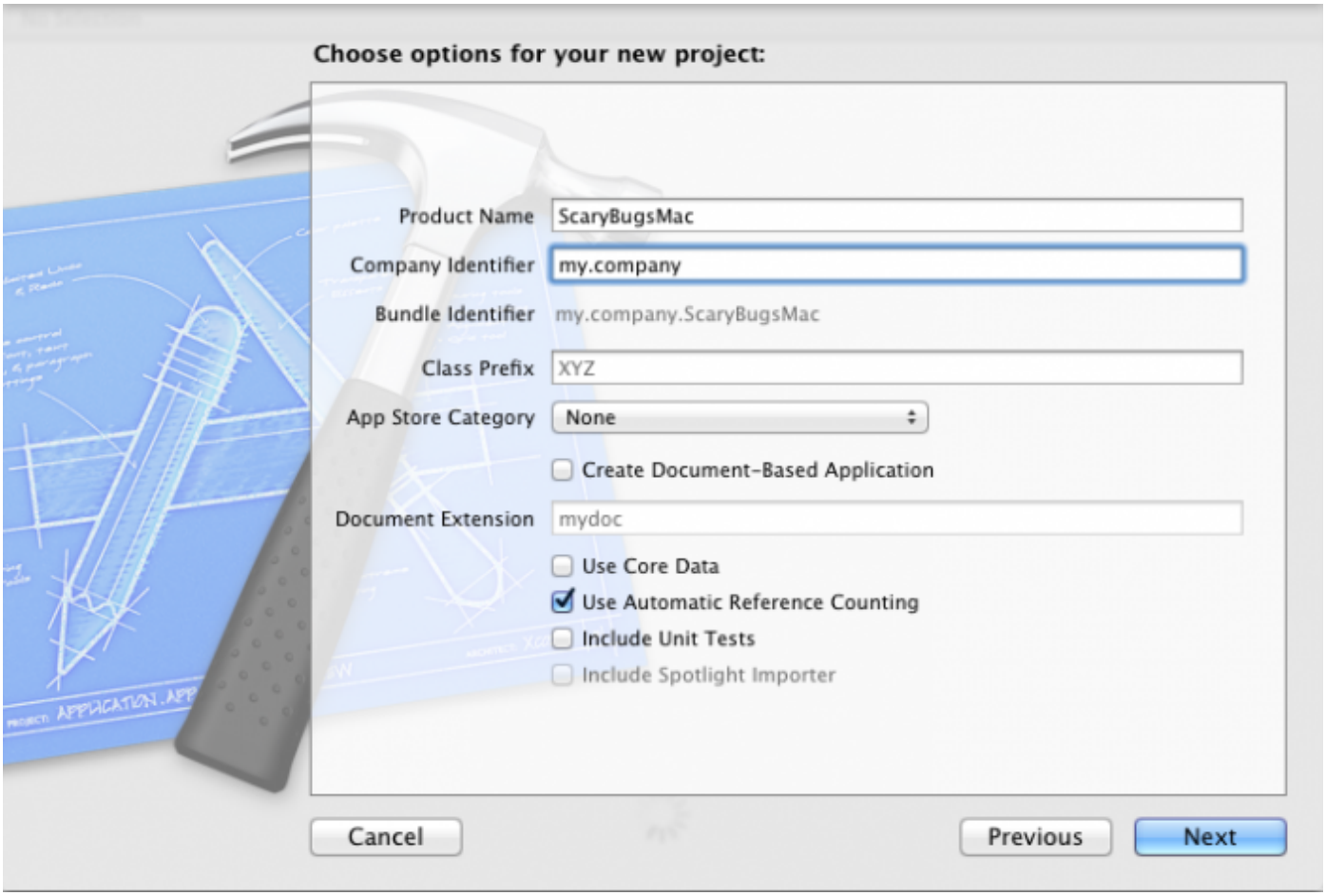
Creating a Mac project is very similar to creating an iOS project – it still uses Xcode, just a different template!

So start by going to File\New Project in XCode, and in the window that pops up, select “Application” in the “OS X” section. Then click Next.



On the Next Page, you will enter the application information. Type ScaryBugsMac in the product name and select a unique company identifier. Apple recommends using a reverse domain format. Leave the rest of the text fields blank.

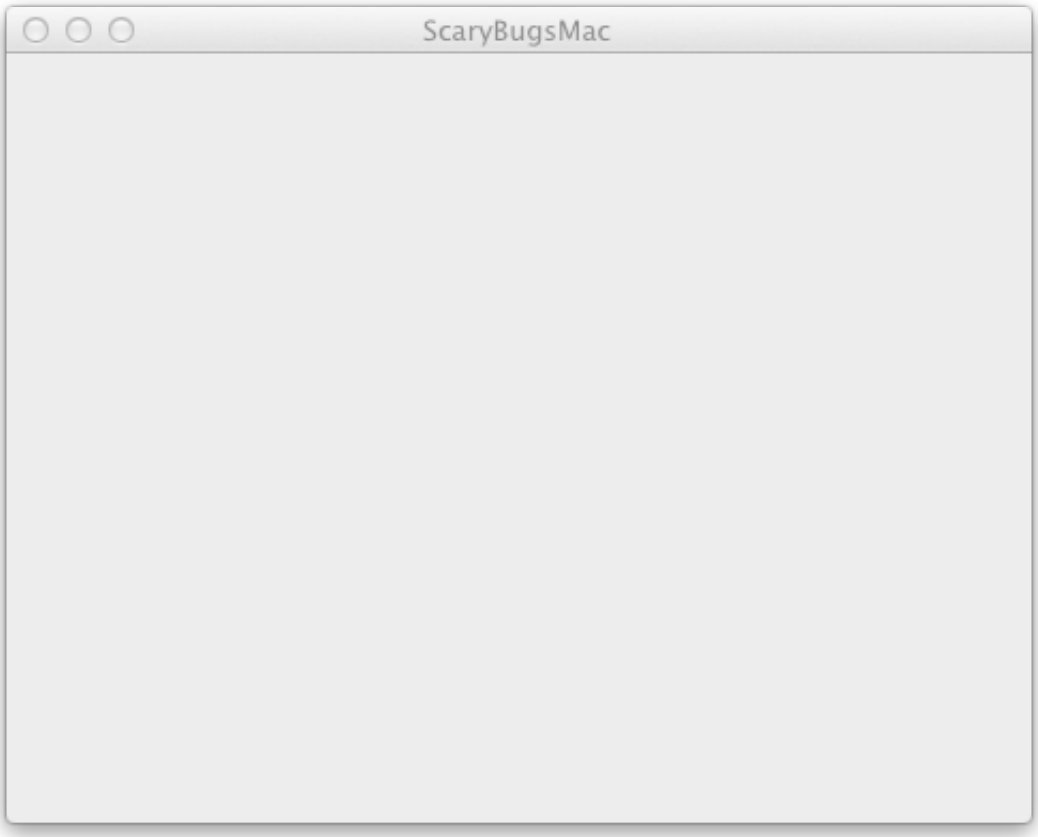
Finally, make sure that only “Use Automatic Reference Counting” is checked. The rest of the checks should not be marked. When you’re done, click Next.



Now XCode will ask you for a location to save to the project. Choose a folder in your computer and click “Create”.

The project is ready, and you should have a Mac Application with an single empty window. Let’s check out how it looks. Find the “Run” button, which is located in the left side of the toolbar at the top of XCode. Click it and XCode will begin to build the app.

When XCode finishes building the application, you should see the main window of your application.

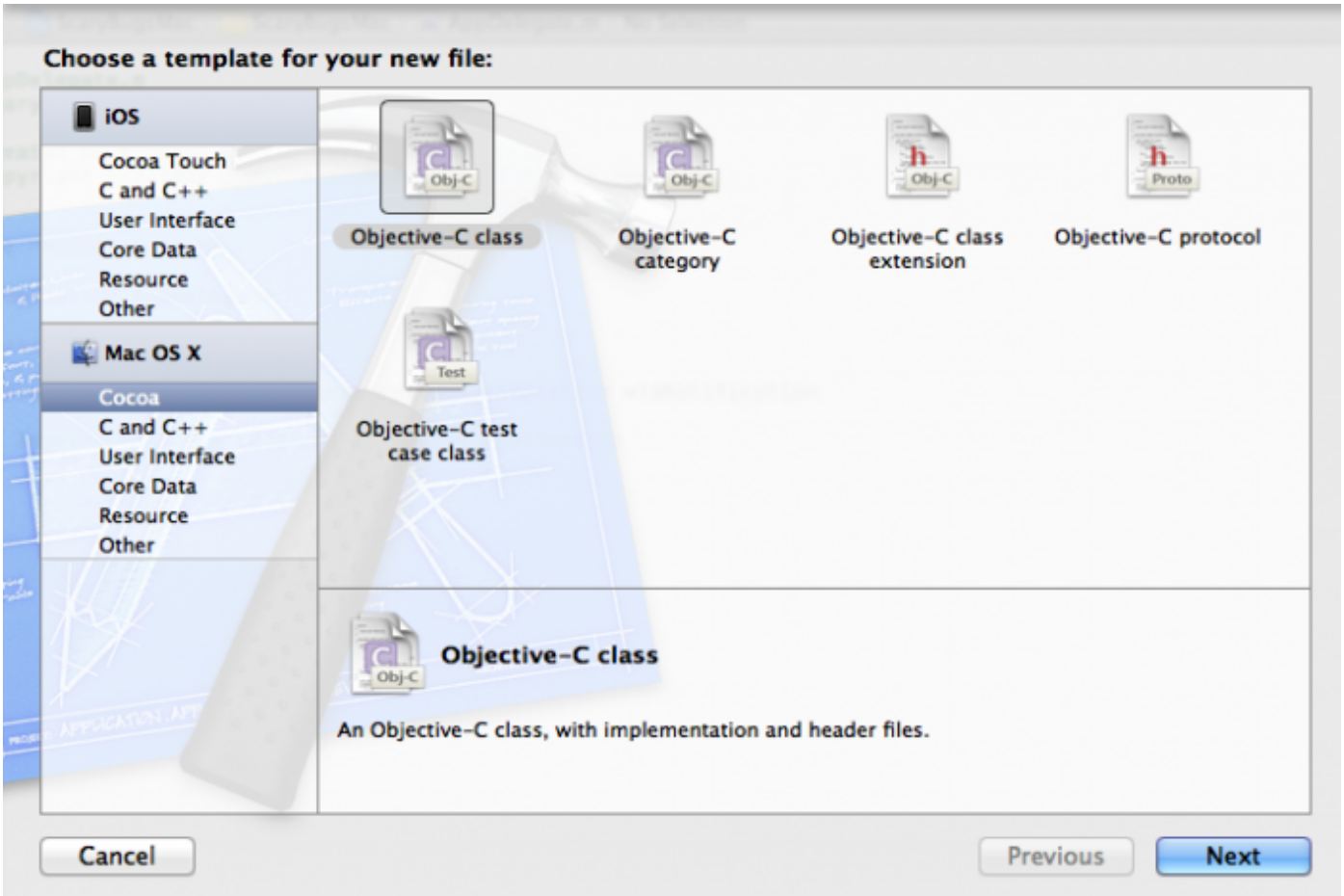


This shows you three things: first you chose the correct template and it works (yay!), second it is a good blank starting point to build upon, and third there are some big (and somewhat obvious) differences from developing for iOS:

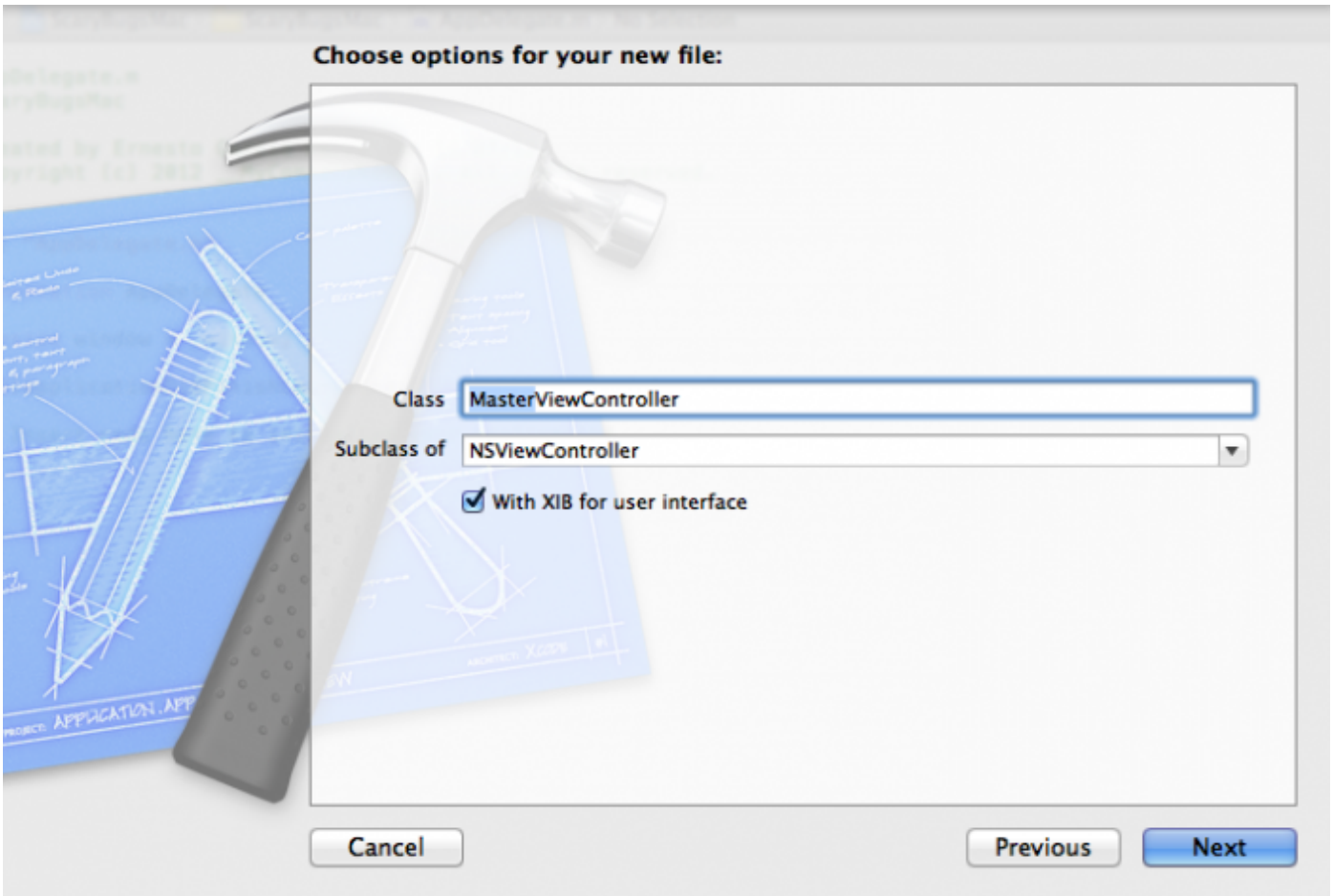
- The window doesn't have to be a particular fixed size such as the iPhone or iPad's screen size – it can be fully resizable!
- Mac apps can have more than one window, and you can minimize them, etc.

Let's do something with this window, and make it show some information about bugs. Just like in iOS, the first thing to do is to create a new View Controller. In this view, you will define the user interface of the main app.

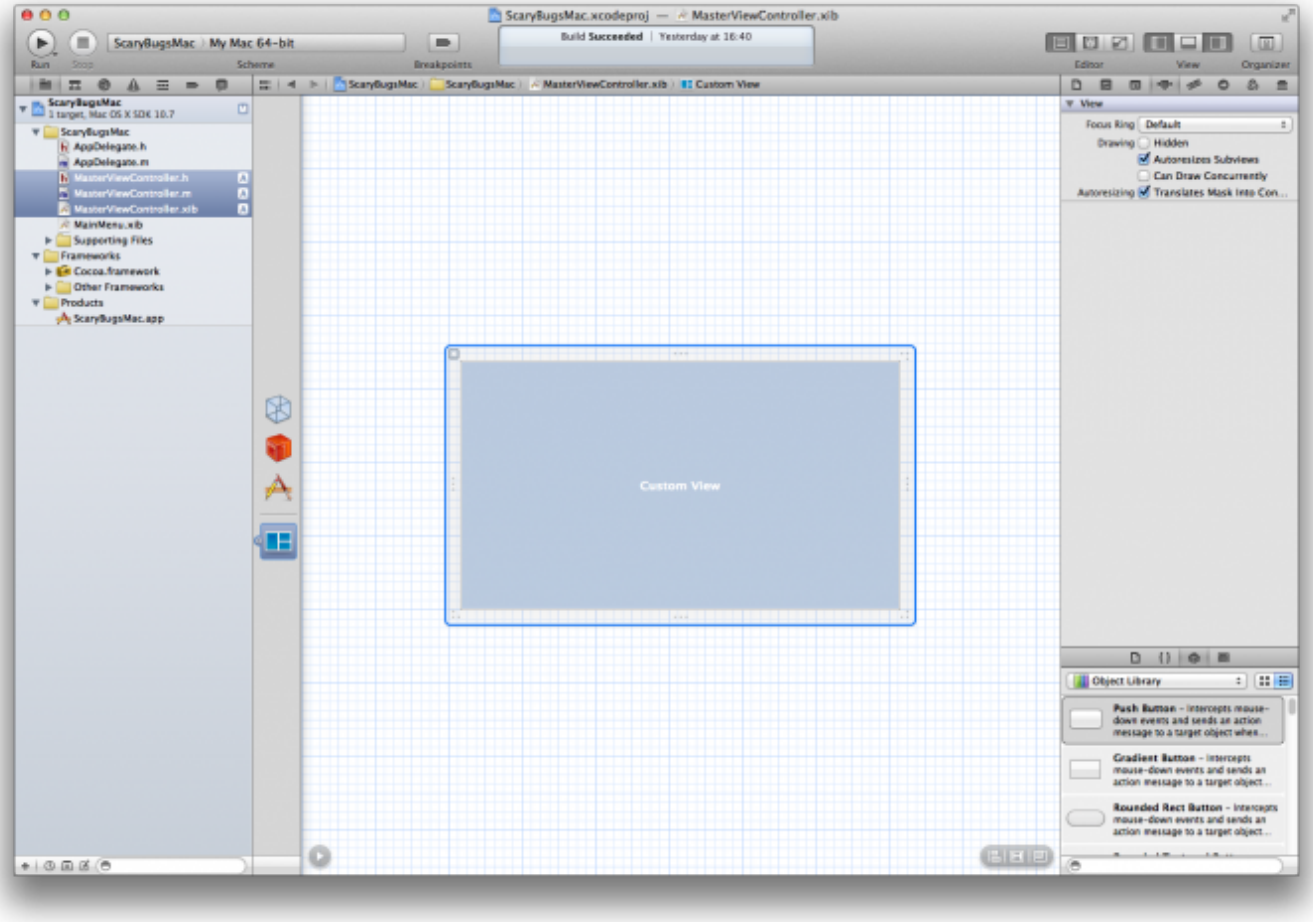
To Create a new View Controller, go to File\New\File⋯ , and in the window that pops up, choose OS X\Cocoa\Objective-C class, and click Next.



Name the class MasterViewController, and type NSViewController for “Subclass of”. Make sure that the option “With XIB for user Interface” is selected.
Click Next.



In the final popup, click Create again. Now your new view Controller is created and your Project Navigator should look similar to this:



Now that you've created the view controller, it's time to place the UI items on it. In the Project Navigator, click on **MasterViewController.xib**. That will load the visual representation of the view controller you just created in Interface Builder.

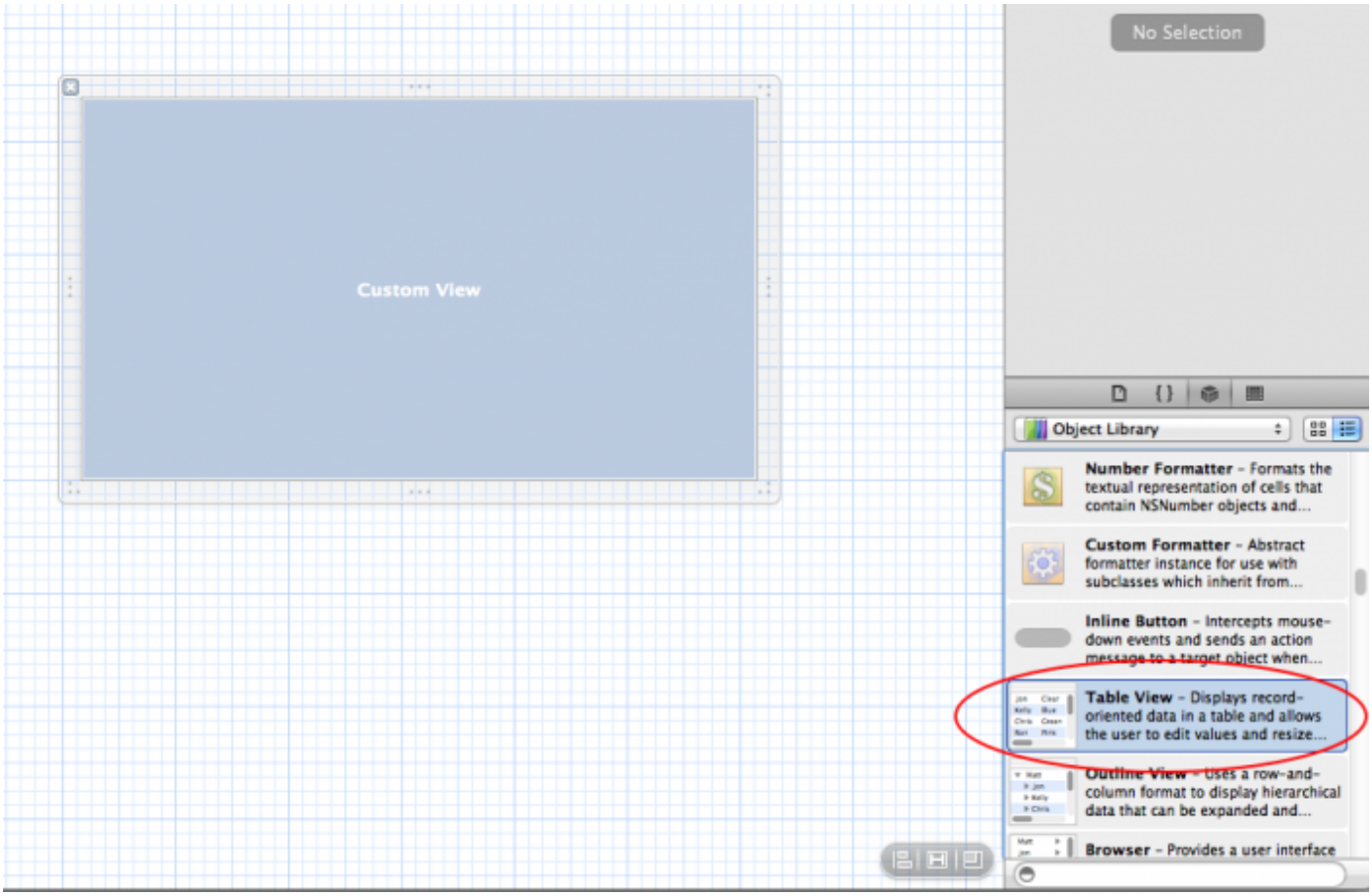
Interface Builder lets you build your user interfaces in a visual way. You just need to drag a component into your view and locate or resize it according to your application's needs.

The first thing your app needs do is to show a list with the Bugs. For that, you are going to need a table view. In OSX, the control is called NSTableView (similar to UITableView in iOS).

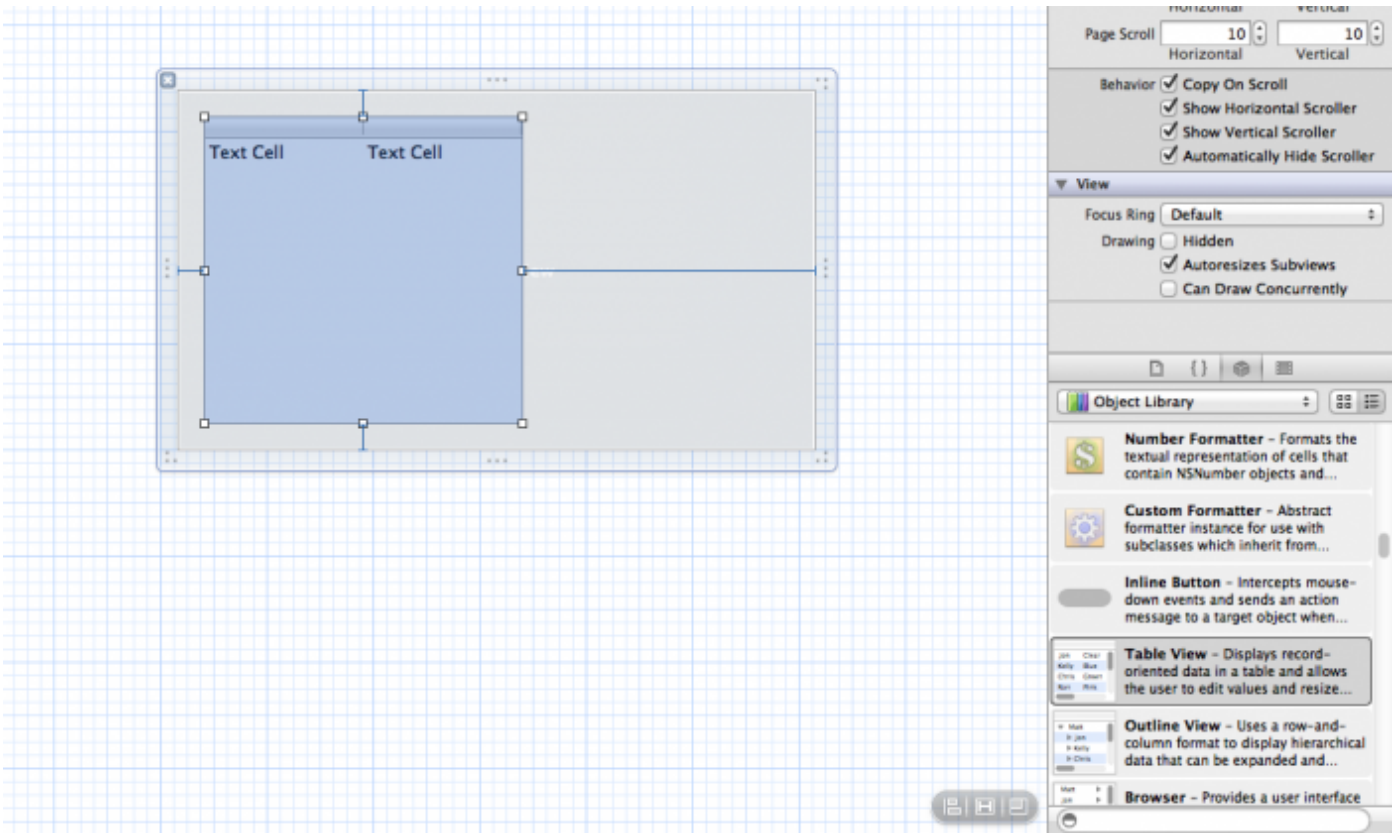
If you're familiar with iOS programming, you may be able to see a pattern here. Lots user interface classes in that are in UIKit were originally derived from classes that already existed in OSX's AppKit. So, some of them just changed the NS preffix used in Mac to the UI preffix used in iOS.

So, as a rule of thumb, if you are wondering if some iOS control you know and love may already exist in Mac, you can try and look for the same class with NS. You'll be surprised how many you'll find – NSScrollView, NSLabel, NSButton and more! Note that the APIs for these controls might be quite a bit different from the iOS variants in some cases though.

The user interface controls are located in the bottom right part of the screen. Make sure the third tab is selected (which is the tab with the UI controls) and find the NSTableView control. (you can scroll down the controls list until you find it, or you can type NSTableView in the panel's search field).



Drag the table view from the panel onto the view and position it near the top left corner. Don't worry now about the table size, you will take care of that later.



Now you have a view with a table on it, but you still haven't added the view controller to the main window so it won't show up. You'll do that in the Application Delegate, so in the Project Navigator select **AppDelegate.m**.

In order to use the new view controller, the Application Delegate must be aware that it exists, so the first thing you

need to do is to import the view controller header file. Add the following to **AppDelegate.m**, just below the line #import "AppDelegate.h" and before the line @implementation AppDelegate:

```
#include "MasterViewController.h"
```

Now you are going to create a property/instance variable for the view controller.
Add the following code just below the line you added before, and before the line @implementation AppDelegate.
Note that properties no longer need to be synthesized due to the new auto-synthesize feature, so you're set :]

```
@interface  AppDelegate()
@property (nonatomic,strong) IBOutlet MasterViewController *masterViewController;
@end
```

Now the Application Delegate has a MasterViewController property, but the view won't be shown on the application's screen yet. To do that you need to instantiate the variable to create a new view, and after that, you need to add the newly created view to the main window of the application.

This must be done when the application starts. The Application delegate has a method "applicationDidFinishLaunching", that is called by the Operative system when the application just started. That is the point where you should add all the initialization code that is meant to run only once when the application starts.

If you are familiar with iOS programming, this method is the equivalent in OSX to the method – (BOOL)application:didFinishLaunchingWithOptions:launchOptions in iOS.

Let's create the view controller and add it to the main window. Insert this code inside **applicationDidFinishLaunching**:

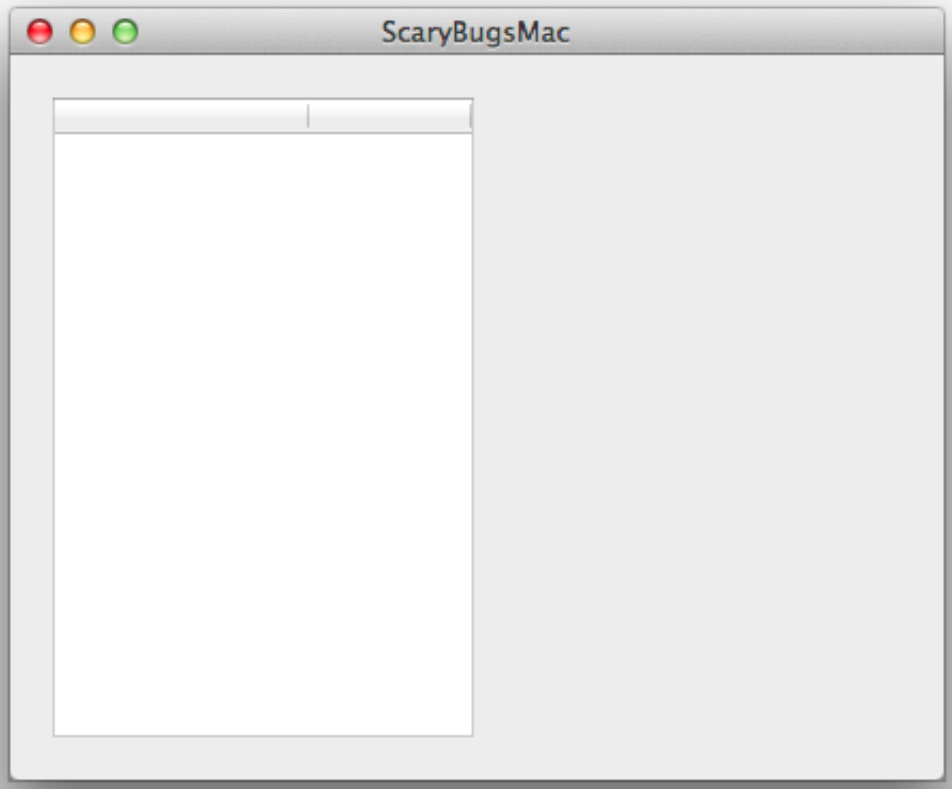
```
// 1. Create the master View Controller
self.masterViewController = [[MasterViewController alloc]
initWithNibName:@"MasterViewController" bundle:nil];

// 2. Add the view controller to the Window's content view
[self.window.contentView addSubview:self.masterViewController.view];
self.masterViewController.view.frame = ((NSView*)self.window.contentView).bounds;
```

The code above performs two actions. First, it creates a new MasterViewController from a nib file using initWithNibName:. Once it's created, it's added two the main window.

The windows in OSX (NSWindow class) are always created with a default view, called contentView that is automatically resized to the window's size. If you want to add your own views to a window, you will always need to add them to the contentView using addSubView.

The last line just sets the size of your view to match the initial size of the window. Comparing this again with iOS programming, it's a bit different. In iOS you would set the window's rootViewController. But rootViewController does not exist in OSX, so you need to add your view to the windows' content View.

Now, if you click Run, you will see that the main window now shows your view with your table view. Nice – now you're starting to get somewhere!



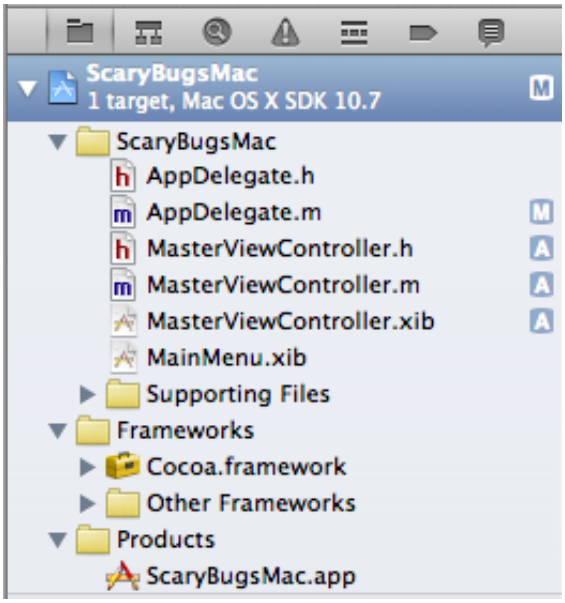
A Scary Data Model: Organization

So far you have a window with a nice table view on it. But it does not do anything at all. You want it to display some information about your scary bugs – but wait, you don't have any data to display either!

And having no data makes me a very sad panda. So in the next steps, you are going to create a data model for the application, but before that, you are going to show you a way to keep things organized in the Project Navigator.

> **Note:** This is an optional section that shows you how to organize files into groups. If you've followed the How To Create A Simple iPhone App on iOS 5 Tutorial or already know how to do that, you can skip to the next section.

This your current organization in the Project Navigator section of XCode:



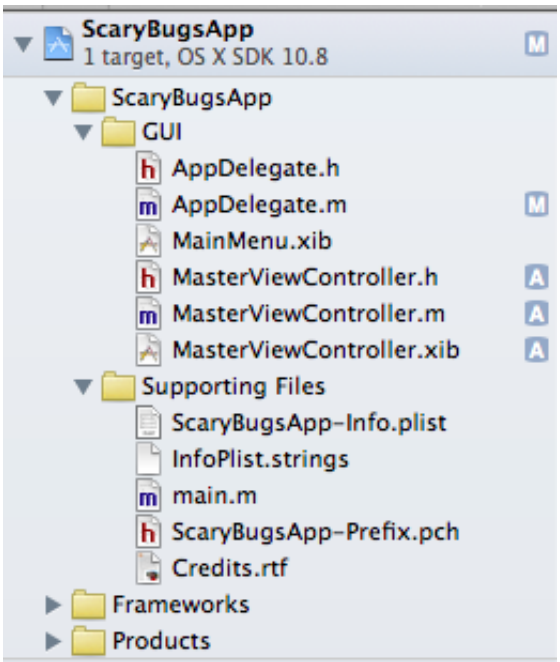
The default template creates a Group with the application's name, and a sub-group for the supporting files (plist, resources,etc). When your project grows up, and you have to deal with lots of files, it becomes more and more difficult to find the files you need.

In this section we're going to show a way to organize your files. This organization is quite subjective, so feel free to change it to any organization you feel comfortable with.

First, you are going to create a group to store your user interface file, and we're going to name it "GUI". To create it, control-Click (or click the mouse right button) the ScaryBugsMac group.

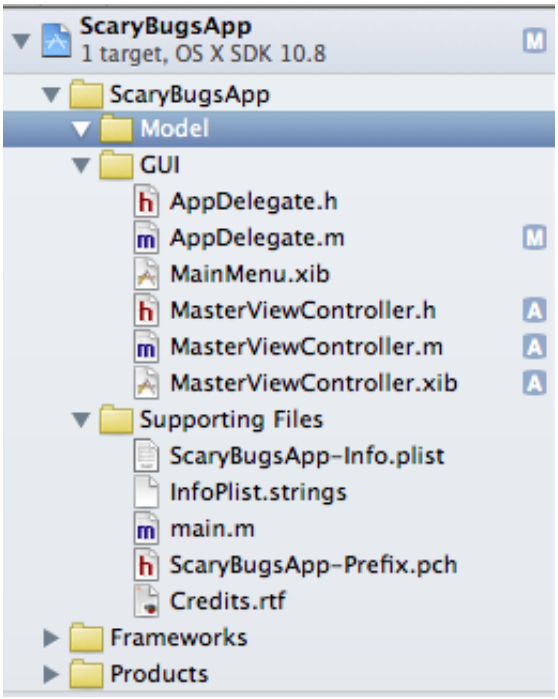
In the menu that pops up, choose "New Group". The group created is automatically selected and you can type de new name "GUI".

Now, drag the user interface files to that group (**AppDelegate.h/.m** , **MasterViewController.h/.m/.xib** and **MainMenu.xib**). After dragging them, your Project Navigator should look like this:



Now create a second group inside ScaryBugsMac, and name it "Model". In the next steps we're going to create the data model files for your application, and you will add those files to this group.

This is the way the Project Navigator looks after adding it.



Before we begin, let's talk about how we're going to organize things:

- **ScaryBugData**: Contains bug name and rating.
- **ScaryBugDoc**: Contains full size image, thumbnail image, ScaryBugData.

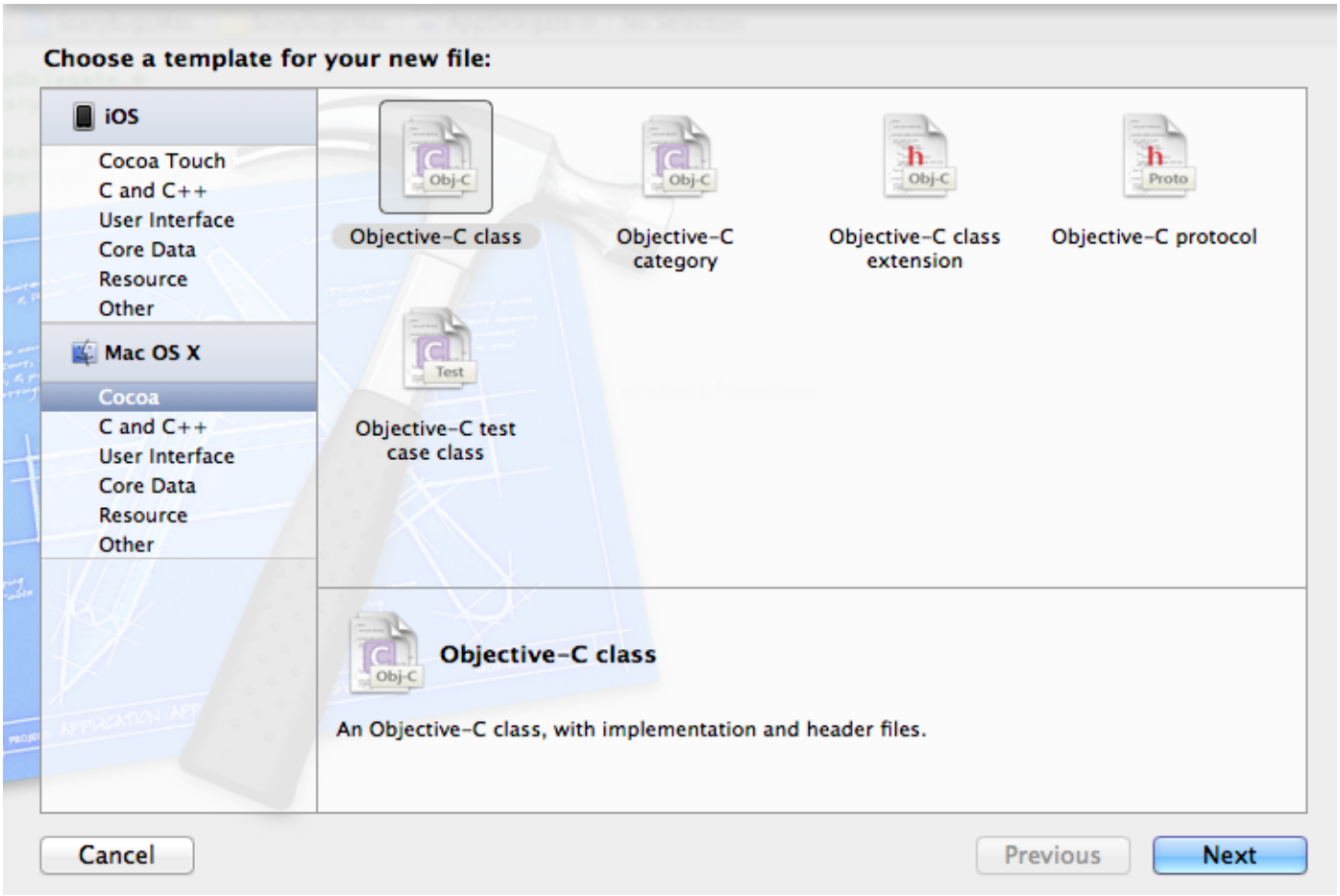
The reason we're setting things up like that is it will make things easier in the follow-up for this tutorial, where we're going to start saving our data to the disk.

A Scary Data Model: Implementation

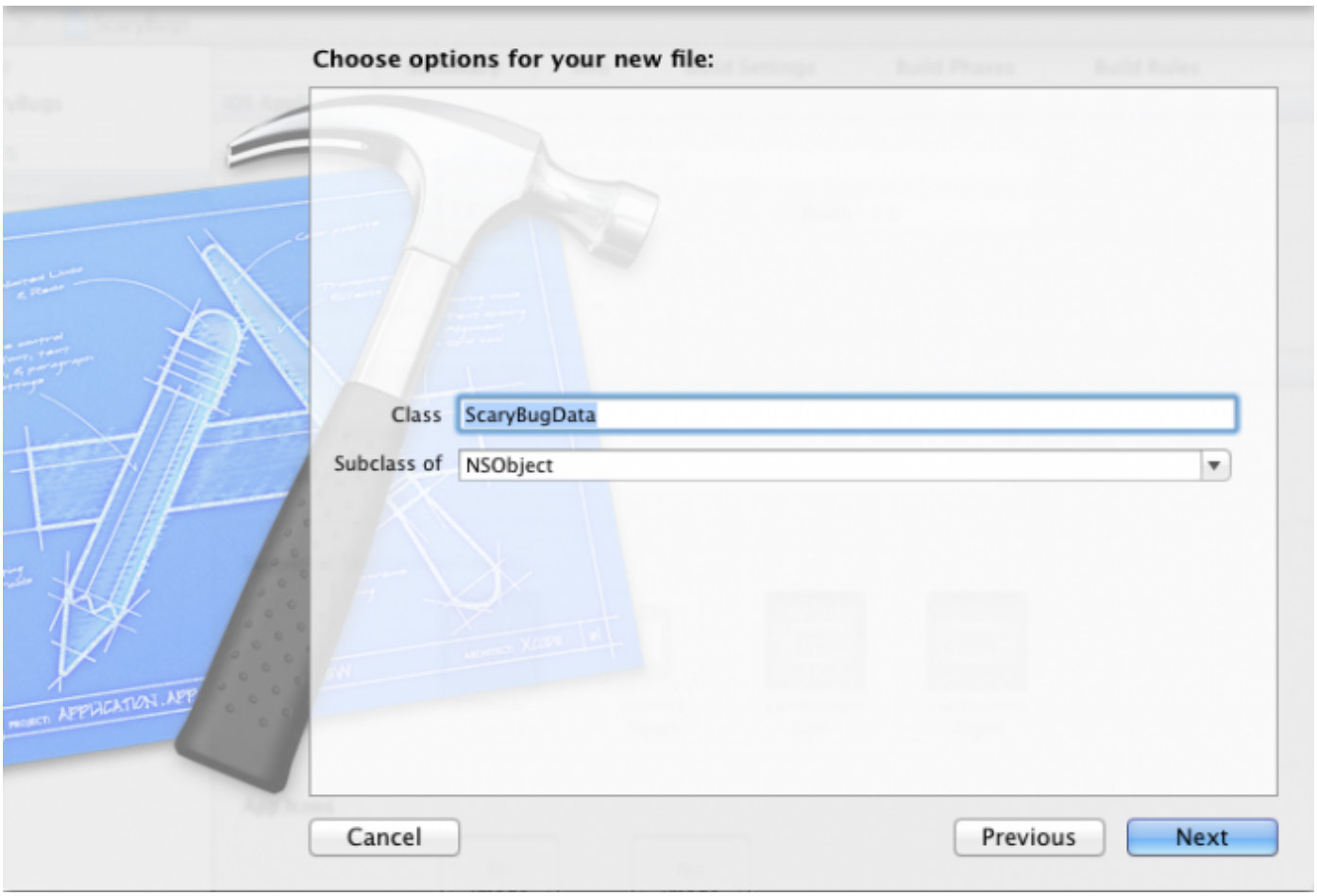
> **Note:** If you've followed the How To Create A Simple iPhone App on iOS 5 Tutorial, you will find that this section is (almost) identical to that. One of the good things about Mac/iOS programming is that they share most of the SDK, obviously, except the UI classes and some OS specific parts.
>
> So, when you're creating the model and classes that don't need user interface, you will find that most of your code will likely just work on Mac, or it will work with some minor changes.
>
> For instance, in this case, changing the ScaryBug model classes from iOS to Mac only required one change. UIImage does not exist in OSX, so you just needed to change it to OSX's image class, NSImage. And that was it!

Let's create the model. We'll begin creating the ScaryBugData file.

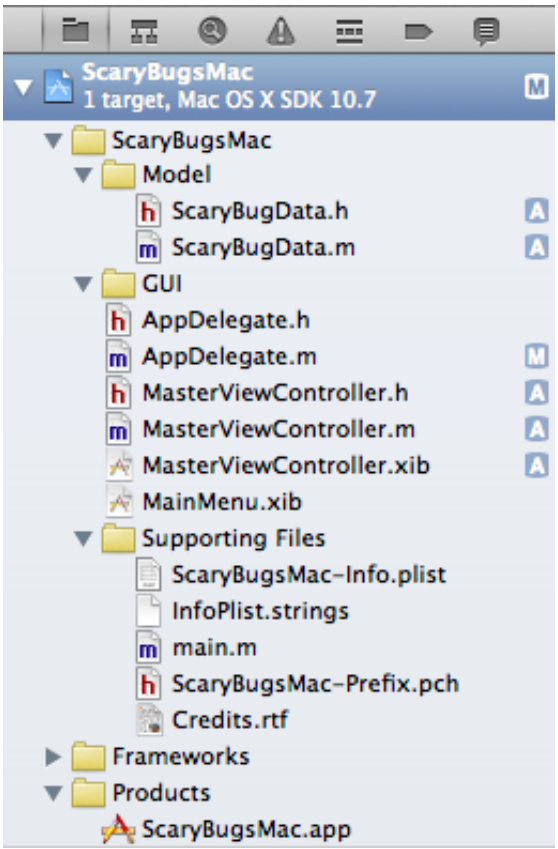
In the project navigator, Control-Click the Model Group you just created, and in the menu click "New File…". Select OS X\Cocoa\Objective-C class template, and click Next.



Name the class ScaryBugData, and enter NSObject for subclass. Click Next.



In the final popup, click Create again. If all went well, your Project Navigator should now look similar to this:



Now, we're going to add the ScaryBugData source code.

First, select the ScaryBugData.h file, and replace all its content with the following:

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

@interface ScaryBugData : NSObject

@property (strong) NSString *title;
@property (assign) float rating;

- (id)initWithTitle:(NSString*)title rating:(float)rating;

@end
```

This is pretty simple stuff – we're just declaring an object with two properties – a string for the name of the bug, and a float for how scary you rated it. you use two property attributes for these:

- **strong**: This specifies that the runtime should automatically keep a strong reference to the object. This is a fancy way of saying that the ARC runtime will keep the object in memory as long as there's a reference to it around, and deallocate it when no references remain. For more information, check out our Beginning ARC in iOS 5 Tutorial.
- **assign**: This means the property is set directly, with no memory management involved. This is what you usually set for primitive (non-object) types like a float.

You also define an initializer for the class, so you can set the title and rating when you create the bug. Switch over to **ScaryBugData.m** and replace it with the following:

```
#import "ScaryBugData.h"

@implementation ScaryBugData

- (id)initWithTitle:(NSString*)title rating:(float)rating {
    if ((self = [super init])) {
        self.title = title;
```

```
        self.rating = rating;
    }
    return self;
}

@end
```

Again, extremely simple stuff here. You reate your initializer to fill in your instance variables from the passed-in parameters. Note there is no need for dealloc, since you are using ARC, and no need to synthesize your properties due to auto-synthesis.

Ok that's it for ScaryBugData. Now follow the same steps you did above to create another subclass of NSObject, this time named **ScaryBugDoc**.

Replace **ScaryBugDoc.h** with the following:

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

@class ScaryBugData;

@interface ScaryBugDoc : NSObject

@property (strong) ScaryBugData *data;
@property (strong) NSImage *thumbImage;
@property (strong) NSImage *fullImage;

- (id)initWithTitle:(NSString*)title rating:(float)rating thumbImage:(NSImage
*)thumbImage fullImage:(NSImage *)fullImage;

@end
```

Nothing of particular note here – just creating some instance variables/properties and an initializer.
Replace **ScaryBugDoc.m** with the following:

```
#import "ScaryBugDoc.h"
#import "ScaryBugData.h"

@implementation ScaryBugDoc

- (id)initWithTitle:(NSString*)title rating:(float)rating thumbImage:(NSImage
*)thumbImage fullImage:(NSImage *)fullImage {
    if ((self = [super init])) {
        self.data = [[ScaryBugData alloc] initWithTitle:title rating:rating];
        self.thumbImage = thumbImage;
        self.fullImage = fullImage;
    }
    return self;
}

@end
```

And that's it – your data model is complete!

At this point you should compile and run your application to check that everything runs fine. You should expect to see the same empty list, because you haven't yet connected the data model with the UI.

Now you have a data model, but you don't have any data. you need to create a list of ScaryBugDocs, and you will store it in a NSMutableArray. We'll add a property to the MasterViewController to keep track of your list of Bugs.

Select **MasterViewController.h** and add the following line, between the @interface and @end lines:

```
@property (strong) NSMutableArray *bugs;
```

This will be the instance variable/property that we'll use to keep track of your list of bugs. Next let's hook it up!

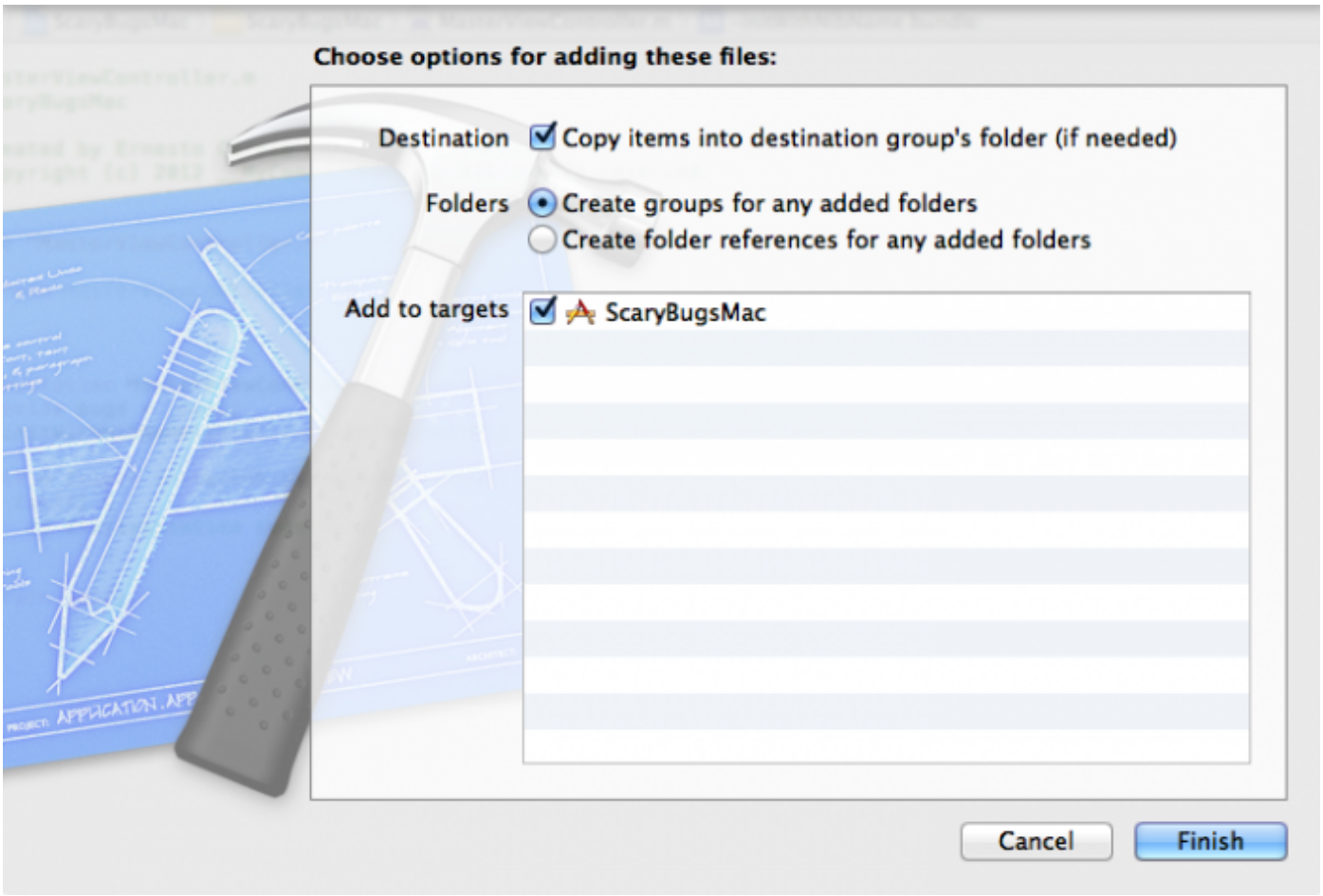
Scary bugs pictures and sample data

At this point, the MasterViewController class is ready to receive a list of Bugs, but then again, you don't have any data yet.

Before adding the data, we're going to need some scary pictures! You can download these pictures from the How To Create A Simple iPhone App on iOS 5 Tutorial or find your own favorite scary bugs on the Internet :]

Once you've downloaded the files or gotten your own, drag them all into the root of your Project Navigator tree. When the popup appears, make sure "Copy items into destination group's folder (if needed)" is checked, and click Add.

If you want to keep things more organized, you could create a Sub-group for the bugs pictures, and drag the files to that group.



Now, let's finally create the sample data. Select **AppDelegate.m** and add the following line at the top of the file, just bellow the #include "MasterViewController.h" line

```
#import "ScaryBugDoc.h"
```

In order to create the sample data and pass it over to the MainViewController, do the following changes in applicationDidFinishLaunching method, just above the line "[self.window.contentView addSubview:self.masterViewController.view];":

```
// Setup sample data
ScaryBugDoc *bug1 = [[ScaryBugDoc alloc] initWithTitle:@"Potato Bug" rating:4
thumbImage:[NSImage imageNamed:@"potatoBugThumb.jpg"] fullImage:[NSImage
imageNamed:@"potatoBug.jpg"]];
ScaryBugDoc *bug2 = [[ScaryBugDoc alloc] initWithTitle:@"House Centipede" rating:3
thumbImage:[NSImage imageNamed:@"centipedeThumb.jpg"] fullImage:[NSImage
imageNamed:@"centipede.jpg"]];
ScaryBugDoc *bug3 = [[ScaryBugDoc alloc] initWithTitle:@"Wolf Spider" rating:5
thumbImage:[NSImage imageNamed:@"wolfSpiderThumb.jpg"] fullImage:[NSImage
```

```
imageNamed:@"wolfSpider.jpg"]];
ScaryBugDoc *bug4 = [[ScaryBugDoc alloc] initWithTitle:@"Lady Bug" rating:1 thumbImage:
[NSImage imageNamed:@"ladybugThumb.jpg"] fullImage:[NSImage
imageNamed:@"ladybug.jpg"]];
NSMutableArray *bugs = [NSMutableArray arrayWithObjects:bug1, bug2, bug3, bug4, nil];

self.masterViewController.bugs = bugs;
```

Here you just use the ScaryBugDoc initializer to create four sample bugs, passing in the title, rating, and images for each. You add them all to a NSMutableArray, and pass them over to your masterViewController using the bugs property.

And finally you have some data! Compile and run your app, and make sure all works well without errors.
We still don't see anything in the user interface, but at this point the view controller has the data it needs, and we're able to begin the work in the user interface to finally show your Scary Bugs List.

A Different Kind of Bug List

In order to display your Bug List, you need to set up the table view to get the list from your model.

In OSX, the table view control is called NSTableView. It's similar to UITableView in that it displays lists of data, but one major difference is that in NSTableView each row can have multiple columns!

Just like UITableView, NSTableView has cells for each row. However, there have been some recent changes as to how these work:

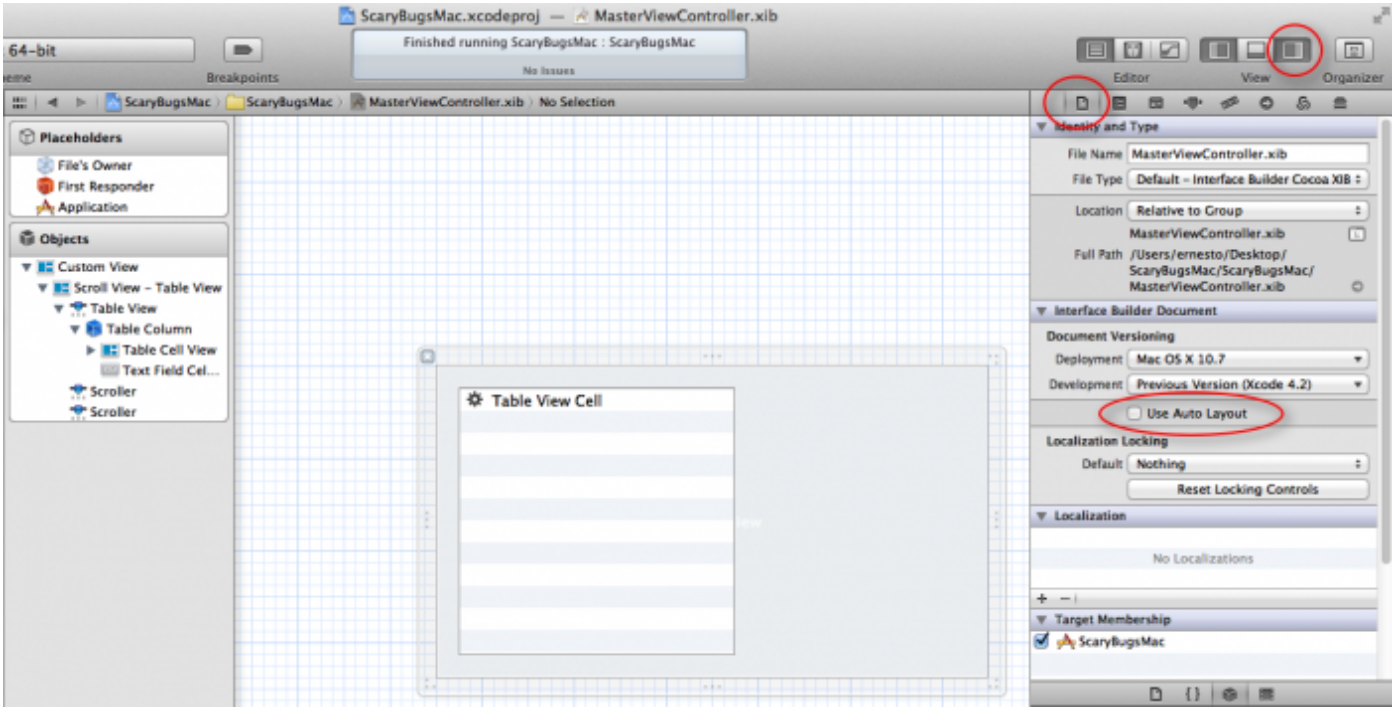
- **Before OSX 10.7 Lion**, table view cells were a special class derived from NSCell class. They were not based on views, and it was responsibility of the coder to handle the drawing and even mouse events (gah!)
- **From OSX 10.7 on**, there's a new type of table view – the View-Based table view. This table view works in a very similar way to UITableView. The cells are a special type of view (NSTableViewCell), and working with it is very similar to the way it works in iOS – which is much easier!

In this tutorial you are going to use the new View Based Table View. We'll cover the basics here, but if you want to learn more about NSTableView, you can read the Table View Programming Guide which does a great job explaining how the table views work.

Before setting up the user interface, you need to make a minor change in the nib files of the project – disable Auto Layout. Auto Layout is a new feature introduced in OSX 10.7 Lion, aimed to handle automatically the resizing of the User Interface controls based on a series of rules defined by the programmer.

Auto Layout is beyond of the scope of this tutorial and makes the explanation of some things a bit confusing, so you are going to disable it. When Auto Layout is disabled, the autoresizing can be configured and behaves exactly the same as in iOS 5 projects.

Select **MasterViewController.xib**. When the Interface Builder interface opens, in the Utilities panel on the right side of the window, make sure that the "File Inspector" tab is selected (it's the first one on the left in the tab bar). In the File Inspector Tab, uncheck "Use Auto Layout".



After that, you need to repeat the same operation with the main window. Select **MainMenu.xib** and select the Window. Disable auto layout in the same way you did with the Master View.

Now we're ready. Let's set up your table view so it can handle displaying a list of ScaryBugDocs. In the Project Navigator, select **MasterViewController.xib**.

Now, in the Interface Builder view, select your table view. Be aware that the table view is embedded in a scroll view, so first time you click it, you will select the scroll view.

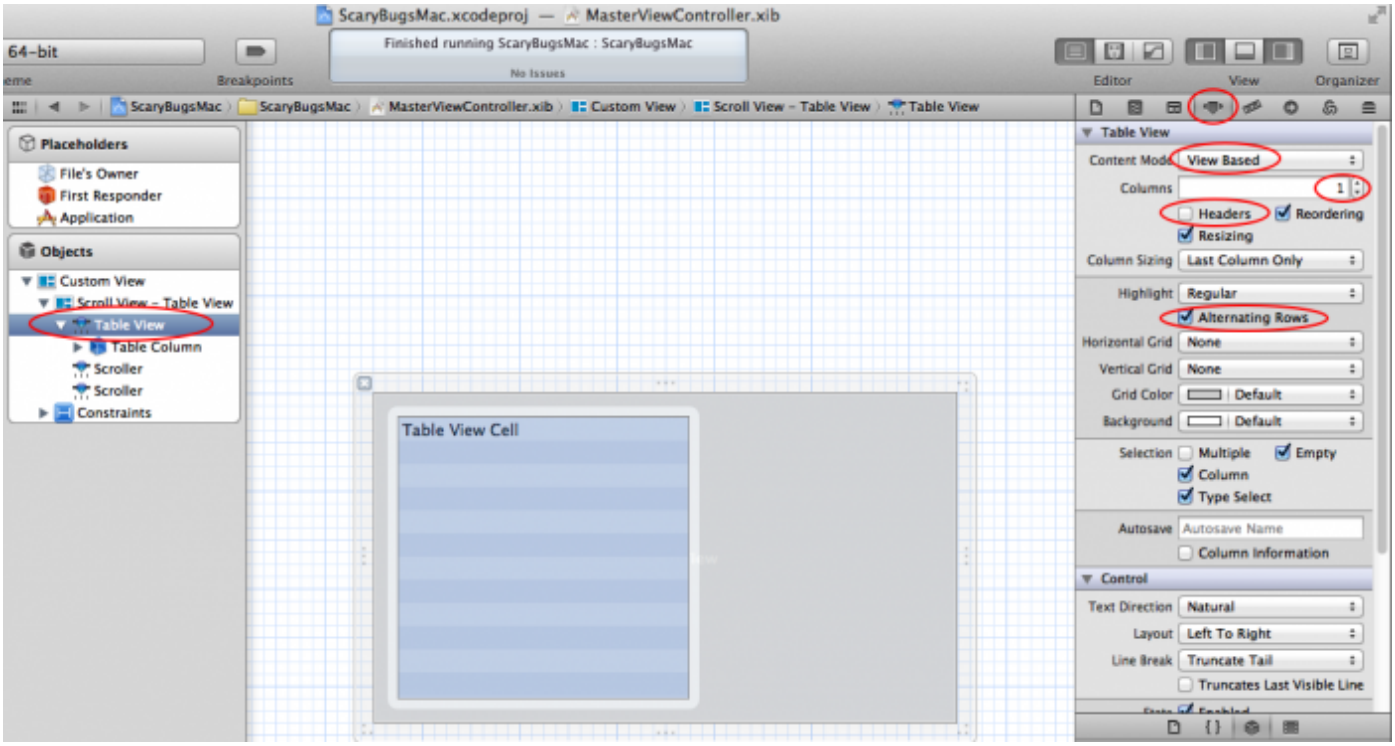
In order to select the table view, click on it a second time (not a double-click, but a second click a moment later). Another way to select it is to click directly on the table view on the "Objects" Panel on the right side.

Once you have it selected, the first thing you need to do is change the table view to "View based", because Interface builder creates "Cell based" table views by default.

To change it, make sure that you have selected the "Attributes Inspector Tab" on the properties panel on the right of the screen. And then, in "Content Mode" select "View Based". Your list does not need multiple columns, so change the Columns property to one.

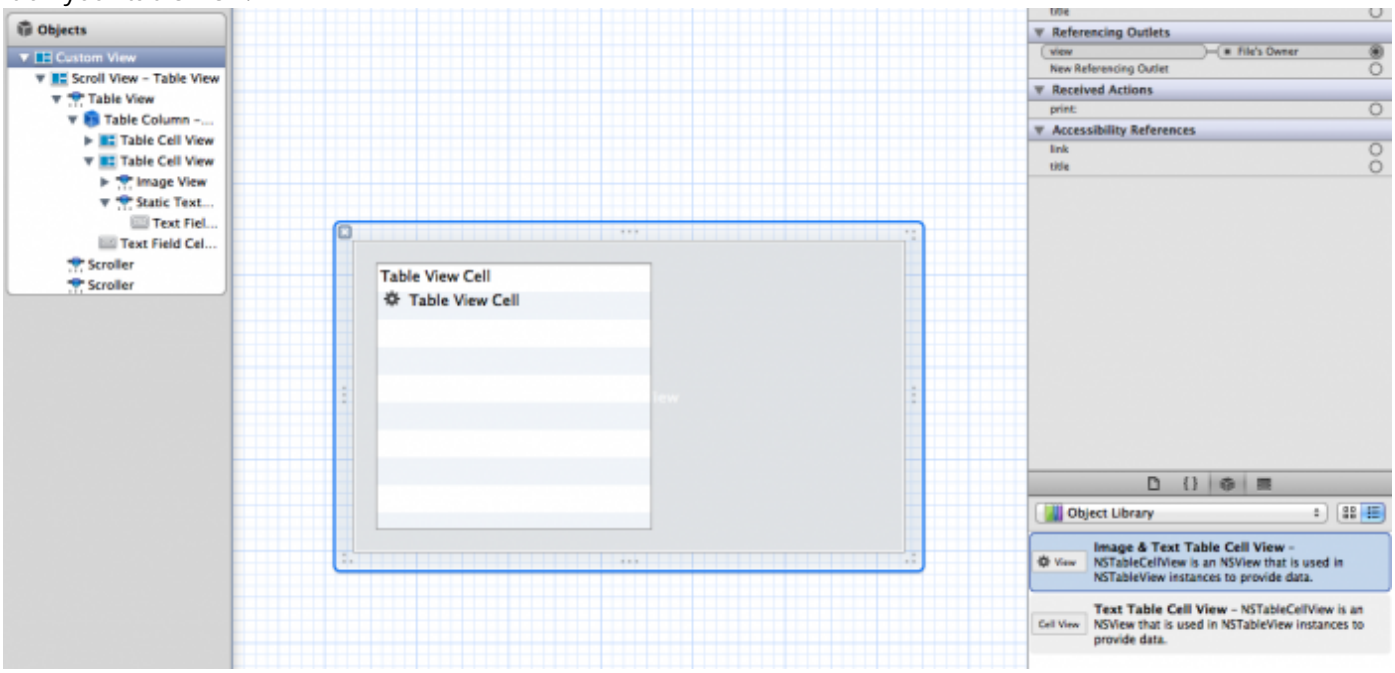
In order to customize the list a little bit, check the property "Alternating Rows", which will draw the rows in alternating white/blue colors, and uncheck the "Headers" property. This will remove the heading of the table, because you don't need it for the tutorial.

After removing the extra columns, the remaining column may be narrower that the table view. In order to resize it, just click on the table column (by clicking three times on the table view, or by using the Objects panel on the right) and resize it to the table's full width.



The next step is to configure the cell view that the table view will use. Your list needs to display the image of the Bug and its name. You need an image and a text field in your cell to show that information. Interface Builder has a preconfigured NSTableCellView that includes an image view and a text field, so you are going to use that one.

In the Object library panel on the bottom left side of the window, locate the “Image & Text Table Cell View”, and drop it on your table view.

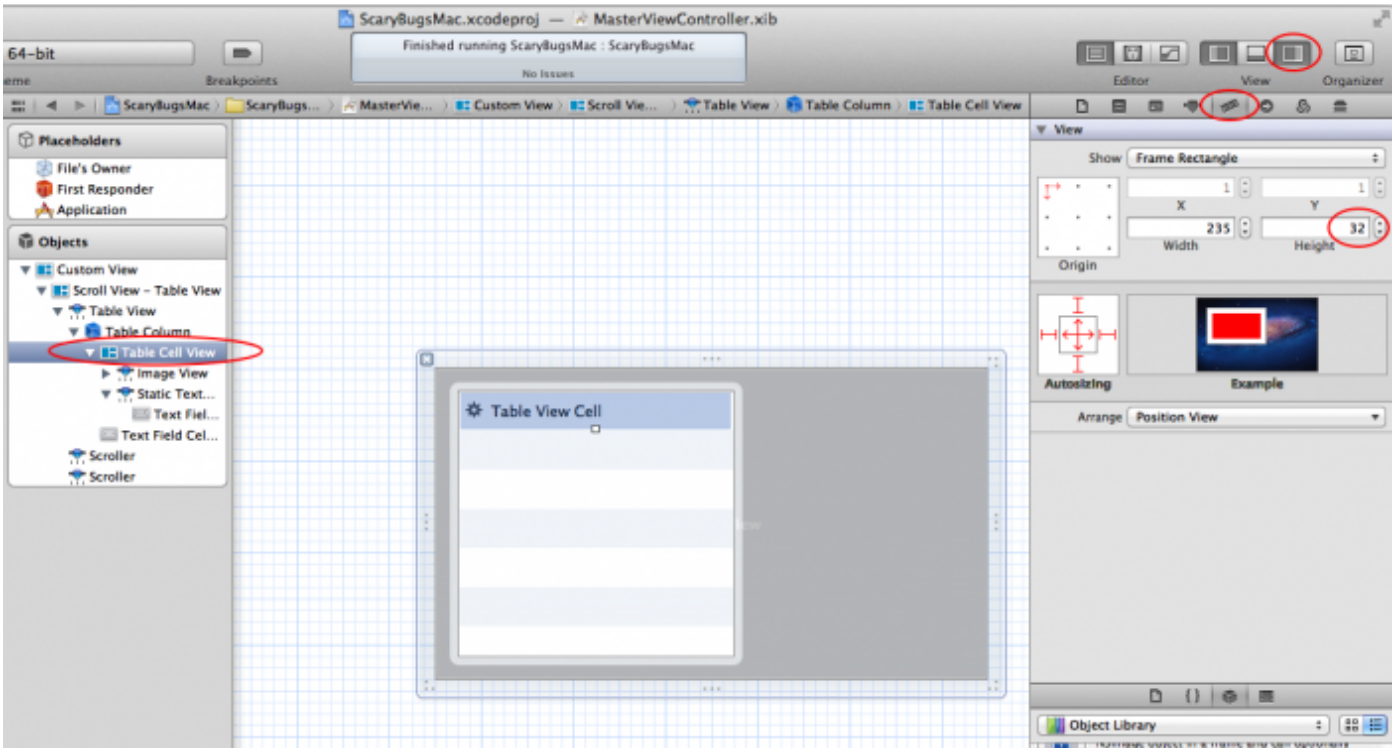


After doing that, your table has now two different types of cells. Remove the old cell type (the cell that does not have the gear icon on the left) by clicking on it and pressing the Delete Key on your keyboard.

The last step is changing the height of the cell, because now it’s too small to show the bug image. you want to set its height to 32. Select the cell by clicking on it, and then open the “Size Inspector” tab in the Utilities panel on the right side of XCode window. You can change the height of the cell to 32 in the Height panel.

Another way to do it is dragging the bottom border of the cell until you get to the desired height. After that, the image and the text fields are a bit misaligned. To fix that, click on then and move them until they are centered in the cell. You can also resize the image view and play with the text field font to fit it to your needs.

Now the table view design should look like this:

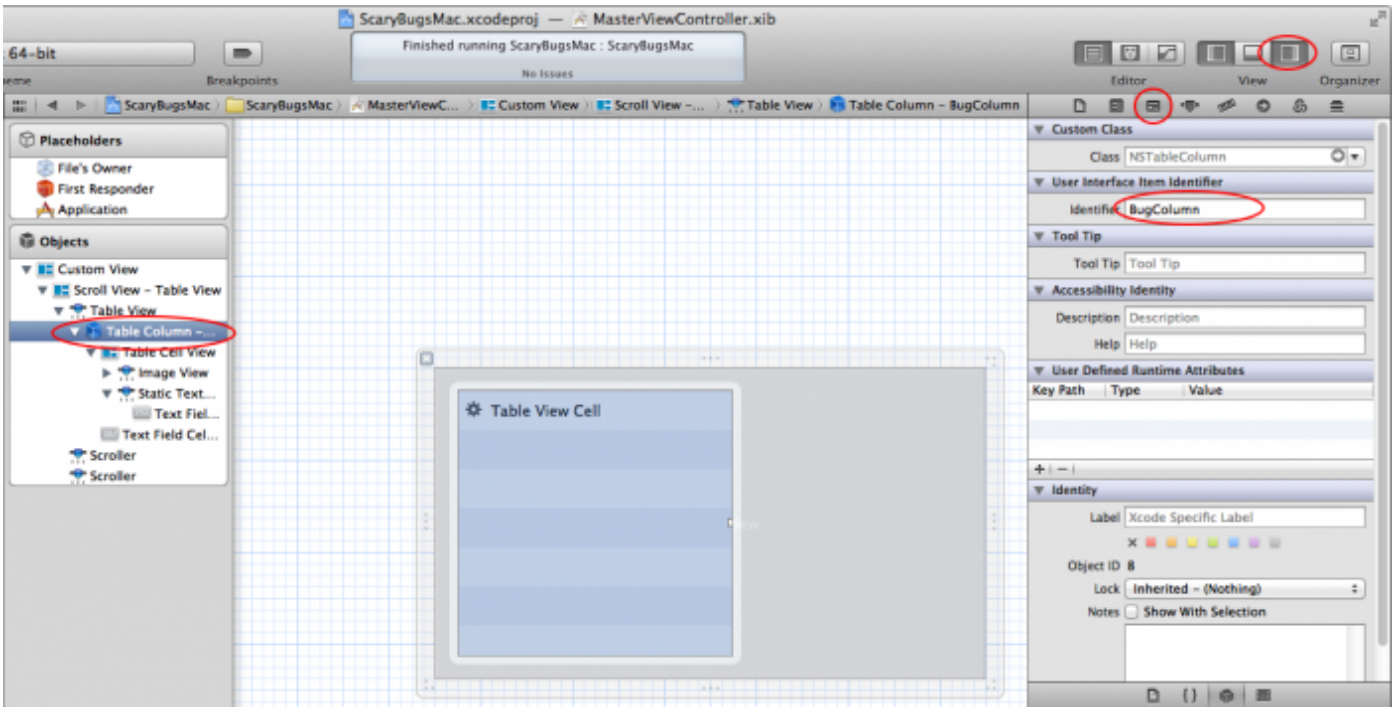


Now you need to set the column identifier. This is a name you give to every column of the table view, so that when you want to perform some actions or you receive some notification from a column, you are able to identify the column.

This may not be strictly necessary in this tutorial, because you only have one column, but it's a good practice to do it so that you don't find any issues when you want to create a multicolumn table view in other projects.

To do that, select the table column (remember, you may need to click three times in the table view to select it, or you can use the Objects panel on the left), and after that, open "Identity Inspector" tab in the Utilities panel.

There, change the Identifier from "Automatic" to "BugColumn".



That's it for the table UI configuration. Now, you need to connect the table view with the MasterViewController, so that they are aware of the existence of each other.

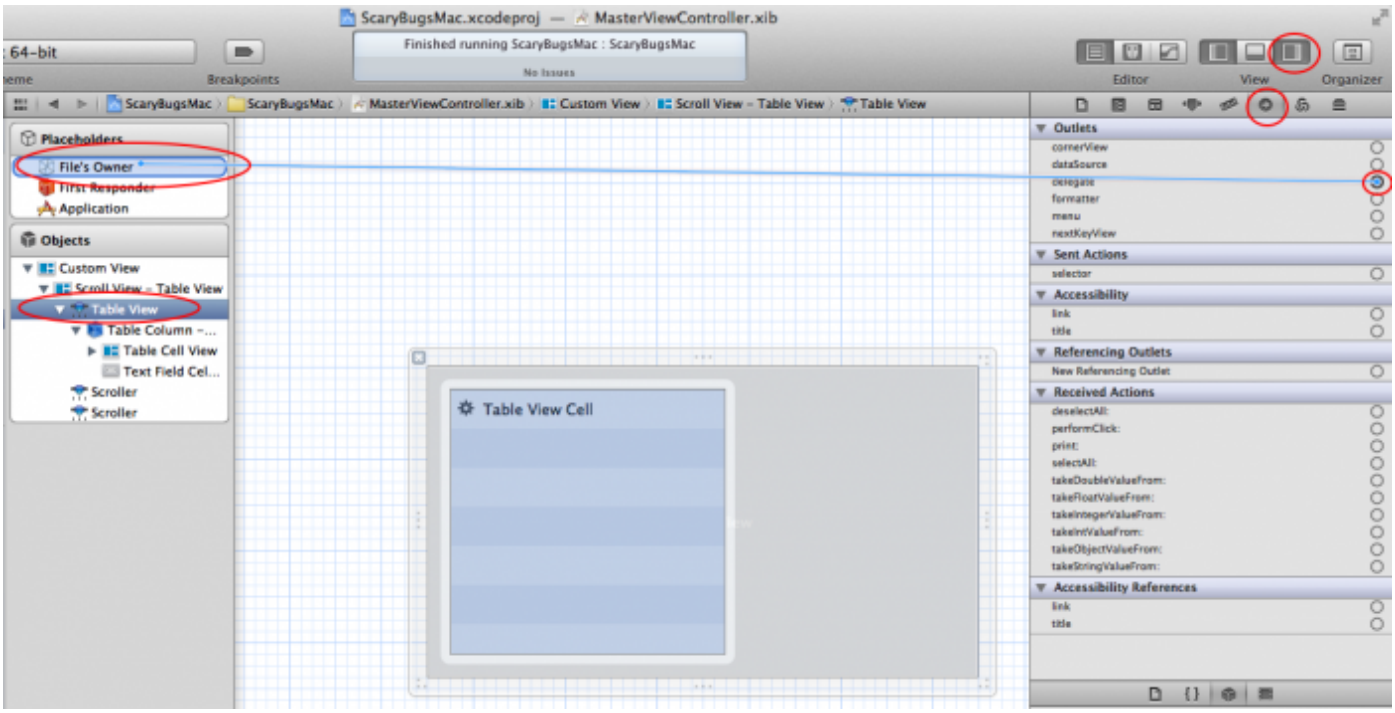
Like in iOS, the table view has two properties that are used to let the table and its controller communicate: the datasource and the delegate.

Basically, the datasource is the class that will tell the tableview what data it needs to show.
And the delegate is the one that controls how the data is displayed, and the one receives notifications from the table view, like for instance, when a cell is selected.

Usually (but now always) the delegate and the datasource is the same controller. In this case, the datasource and the delegate will be your MasterViewController class.
This can be done programmatically, but for this tutorial you are going to do that connection in Interface Builder.

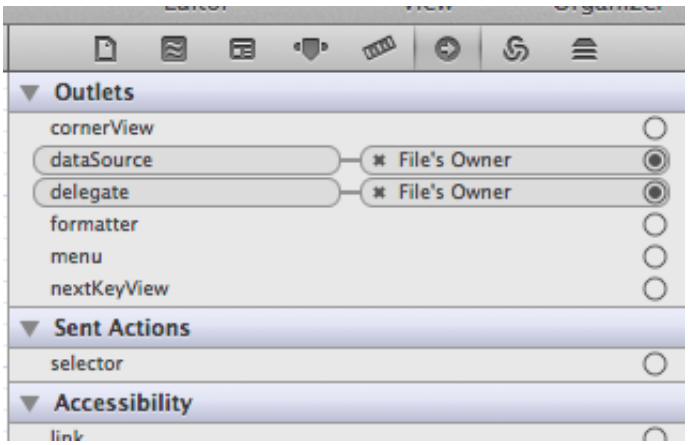
Select your table view, and in the Utilities Panel choose the “Connections Inspector” (the one with an arrow pointing to the right). There, in the “Outlets” section, you will see the delegate in the datasource. Let’s connect the delegate first.

To do that, click on the circle on the right of the delegate, and drag it to the File’s Owner (MasterViewController), located on the “PlaceHolders” panels on the left side.



Just with that you told the table view that its delegate is the MasterViewController. When you instantiate your view controller in your app, that connection will be automatically setup for us.

Now, repeat the same procedure for the datasource outlet. After doing it, you must see both connections pointing to the File’s Owner, like this.



That’s it, now we’re ready to add the necessary code in your view controller to show your Bugs List. Select

MasterViewController.m and add the following at the top of the file, just below the #import "MasterViewController.h" line:

```
#import "ScaryBugDoc.h"
#import "ScaryBugData.h"
```

Then, paste the following code at the end of the file, just before the @end line:

```
- (NSView *)tableView:(NSTableView *)tableView viewForTableColumn:(NSTableColumn
*)tableColumn row:(NSInteger)row {

    // Get a new ViewCell
    NSTableCellView *cellView = [tableView
makeViewWithIdentifier:tableColumn.identifier owner:self];

    // Since this is a single-column table view, this would not be necessary.
    // But it's a good practice to do it in order by remember it when a table is
multicolumn.
    if( [tableColumn.identifier isEqualToString:@"BugColumn"] )
    {
        ScaryBugDoc *bugDoc = [self.bugs objectAtIndex:row];
        cellView.imageView.image = bugDoc.thumbImage;
        cellView.textField.stringValue = bugDoc.data.title;
        return cellView;
    }
    return cellView;
}


- (NSInteger)numberOfRowsInTableView:(NSTableView *)tableView {
    return [self.bugs count];
}
```

Ok, let's have a look at what we're doing here. In order to show data in a table view, you need, to implement, at least two methods.

One is the dataSource's numberOfRowsInTableView:. This is called by the OS to ask to the datasource (MasterViewController in this case) "How many rows do I need to show?" You respond by simply giving the list of bugs in your array.

With that method, the table view knows how many rows to display, but still does not know anything about which cells to display in every row, and what information those cells should have.

That is done in tableView:viewForTableColumn:row. This method will be called by the OS for every row and column of the table view, and there you have to create the proper cell, and fill it with the information you need.

If you have experience with iOS, you will find this is quite similar to the way UITableView works.

numberOfRowsInTableView: is the equivalent of numberOfRowsInSection: in iOS.
And viewForTableColumn:row is the same as cellForRowAtIndexPath: in iOS. The difference here is that in iOS you have to setup a cell based on its section and row, and in OSX the cell setup is based on row and column.

cellForRowAtIndexPath: is a very important method, so let's look this method in detail:

```
- (NSView *)tableView:(NSTableView *)tableView viewForTableColumn:(NSTableColumn
*)tableColumn row:(NSInteger)row {

    // Get a new ViewCell
    NSTableCellView *cellView = [tableView
makeViewWithIdentifier:tableColumn.identifier owner:self];

    // Since this is a single-column table view, this would not be necessary.
    // But it's a good practice to do it in order by remember it when a table is
multicolumn.
    if( [tableColumn.identifier isEqualToString:@"BugColumn"] )
    {
```

```
        ScaryBugDoc *bugDoc = [self.bugs objectAtIndex:row];
        cellView.imageView.image = bugDoc.thumbImage;
        cellView.textField.stringValue = bugDoc.data.title;
        return cellView;
    }
    return cellView;
}
```

The first thing you do is getting a cellView by calling makeViewWithIdentifier:. This method will create (or reuse) the proper cell for a column based on the identifier (which is the one you setup in Interface Builder).

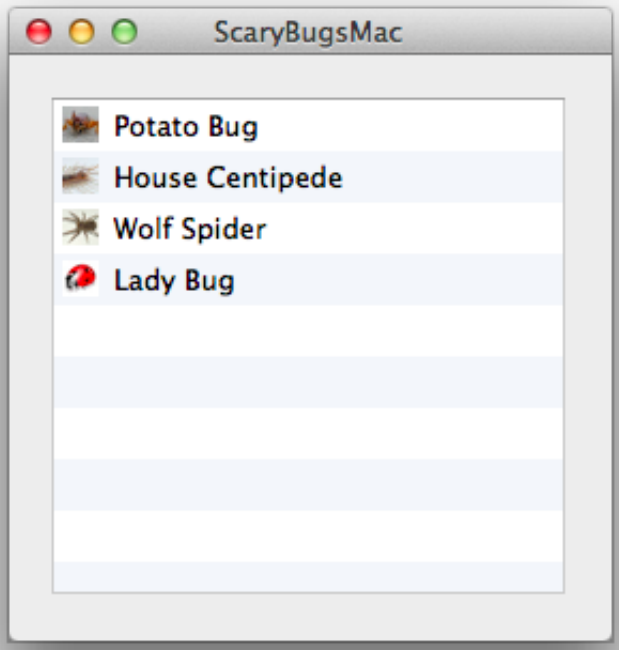
Once you have the cell, is time for us to fill it with information. Well have to do it differently depending of the column. That's why you check for the "BugColumn" identifier. If the column identifier is "BugColumn", then you set the image and text from the ScaryBugDoc.

In this case, this table only has one type of column so this check isn't absolutely needed. However it's good to know how to handle the multicolumn case, since you might need that for your own apps.

The last step is setting the cell information. Based on the row, you get from your bugs array the proper ScaryBugDoc, and then fill the image and the text field with the name of the bug.

That's all you need to display information in a table view. It's just a matter of defining the properties and connections in Interface Builder and implementing only two methods in your view controller.

Now, it's time to compile and Run the application. If everything went fine, you should see the table view with all the Scary Bugs in your list!



Where to go from here?

Here is a sample project with all of the code we've developed so far in this tutorial series.

Next in the Series, you will learn how to add a detail view, and how to add/edit/delete bugs from your list. you will also cover how to polish the User Interface and handle the window resizing to make your app look great at any size!

This is a post by iOS Tutorial Team Member Ernesto García, a Mac and iOS developer founder of CocoaWithChurros.

Ernesto García

Ernesto is a Mac and iOS developer from Spain. After 16+ years of experience developing Enterprise applications, now he is an indie developer who focuses on creating great mobile and desktop applications. He's the founder of CocoaWithChurros where he develops Apps for clients as well as his own. You can also find him on Twitter or Github.