FREE eBook

LEARNING Xamarin.iOS

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.io

S

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xamarin-ios

It is an unofficial and free Xamarin.iOS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Xamarin.iOS.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Xamarin.iOS

Remarks

Xamarin.iOS allows you to create native iOS applications using the same UI controls you would in Objective-C and Xcode, but with the flexibility and elegance of a modern language (C#), the power of the .NET Base Class Library (BCL), and two first-class IDEs - Xamarin Studio and Visual Studio - at your fingertips.

For more information on installing Xamarin.iOS on your Mac or Windows machine, refer to the Getting Started guides on the Xamarin developer center

Versions

Version	Release Date
1.0	2009-09-14
2.0	2010-04-05
3.0	2010-04-16
4.0	2011-04-06
5.0	2011-10-12
6.0	2012-09-19
7.0	2013-09-18
8.0	2014-09-10
9.0	2015-09-17
9.2	2015-11-17
9.4	2015-12-09
9.6	2016-03-22

Detailed info for each release can be found here: https://developer.xamarin.com/releases/ios/

Examples

Get Started in Xamarin Studio

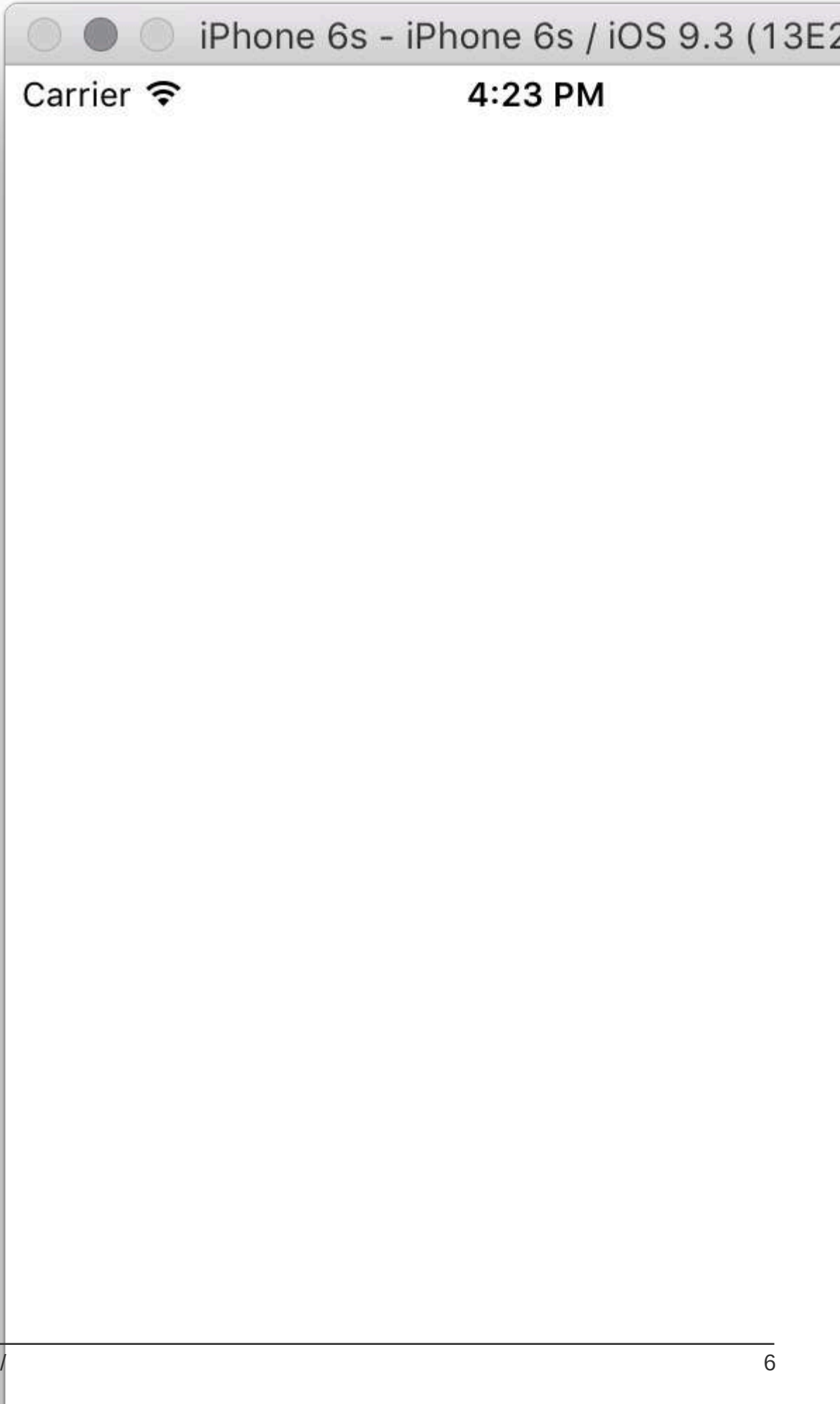
1. Browse to **File > New > Solution** to bring you up the new project dialog
2. Select **Single View App** and press **Next**
3. Configure your app by setting your app name and organization ID, and press **Next**:


Configure your iOS app
App Name:
HelloApp
Organization Identifier:
com.xamarin
Bundle Identifier:
com.xamarin.helloapp
Devices:
iPad
iPhone
Select the minimum iOS version yo
support.

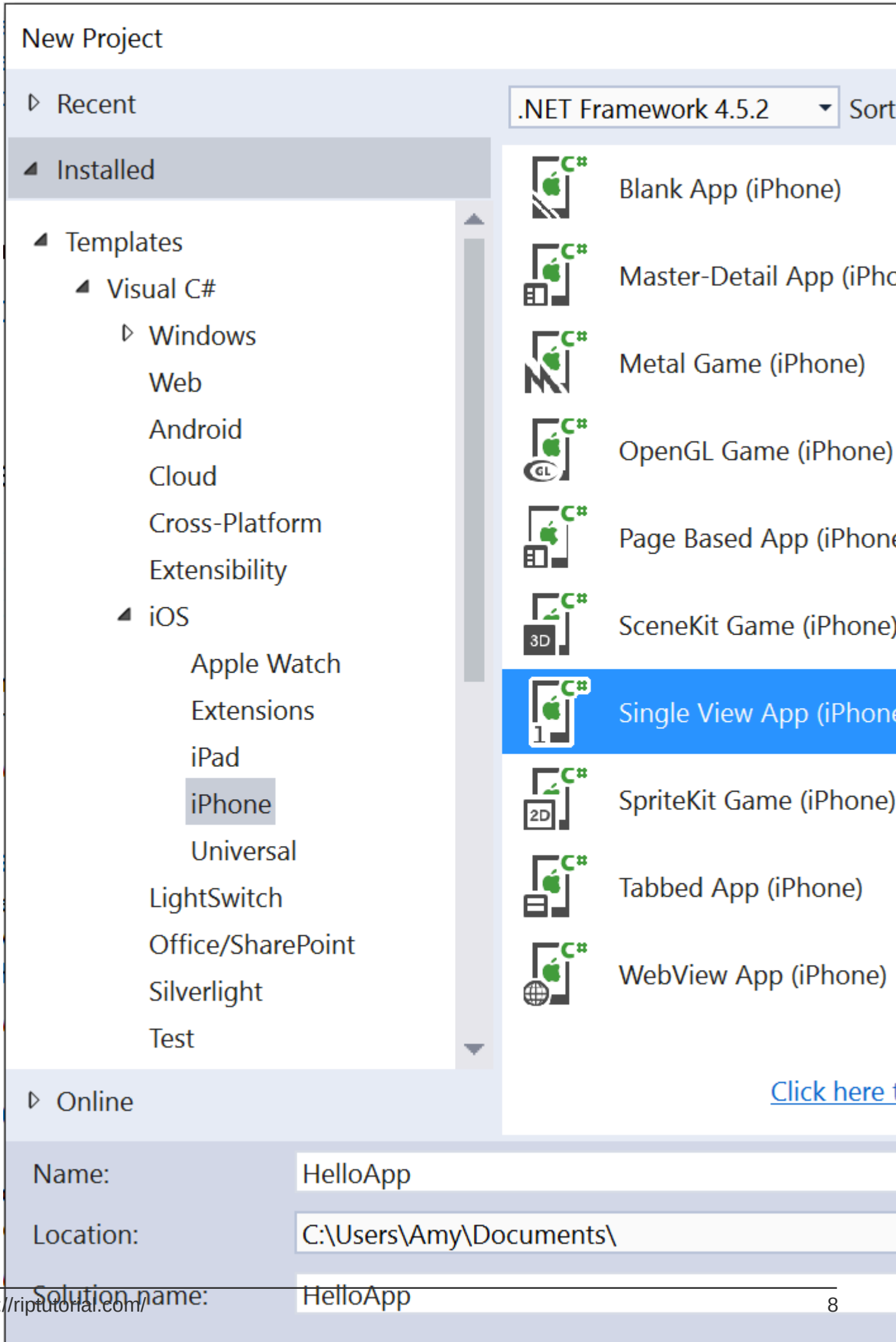
to create your project.

5. To run your application, select the Debug | iPhone 6s iOS 9.x configuration, and press the **Play** button:

6. This will launch the iOS Simulator, and will display your empty application:

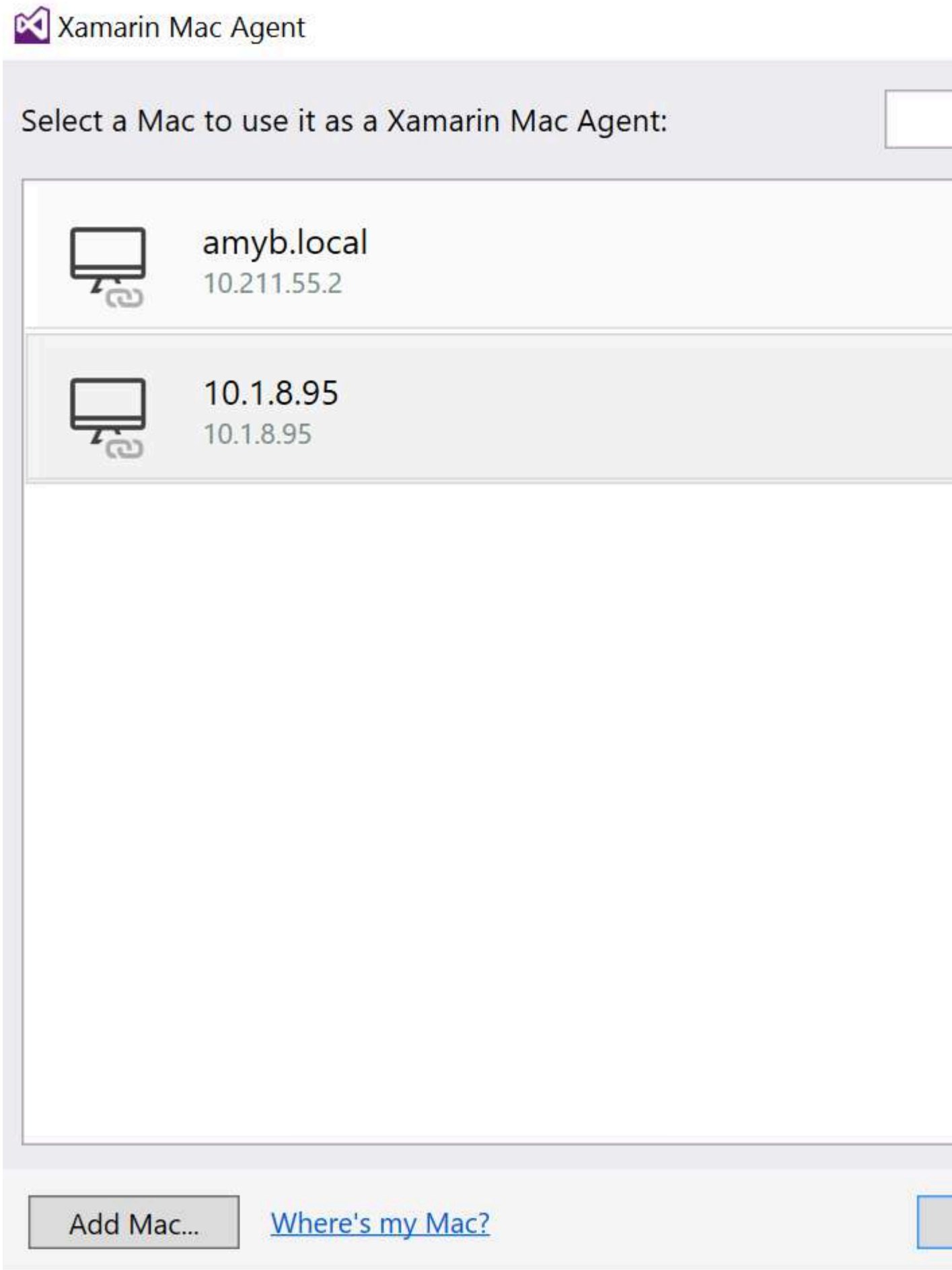

iPhone 6s - iPhone 6s / iOS 9.3 (13E2
Carrier
4:23 PM

2. Navigate to Visual C# > iOS > iPhone and select Single View App:


New Project
Recent
.NET Framework 4.5.2
Sort
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Cross-Platform
Extensibility
iOS
Apple Watch
Extensions
iPad
iPhone
Universal
LightSwitch
Office/SharePoint
Silverlight
Test
Online
Blank App (iPhone)
Master-Detail App (iPho
Metal Game (iPhone)
OpenGL Game (iPhone)
Page Based App (iPhone
SceneKit Game (iPhone)
Single View App (iPhone
SpriteKit Game (iPhone)
Tabbed App (iPhone)
WebView App (iPhone)
Click here
Name:
HelloApp
Location:
C:\Users\Amy\Documents\
Solution name:
HelloApp

3. Give your app a **Name** and press **OK** to create your project.
4. Select the Mac Agent icon from the toolbar, as illustrated below:

5. Select the Mac that will build your application from the list (make sure you Mac is set up to receive the connection!), and press **Connect**:

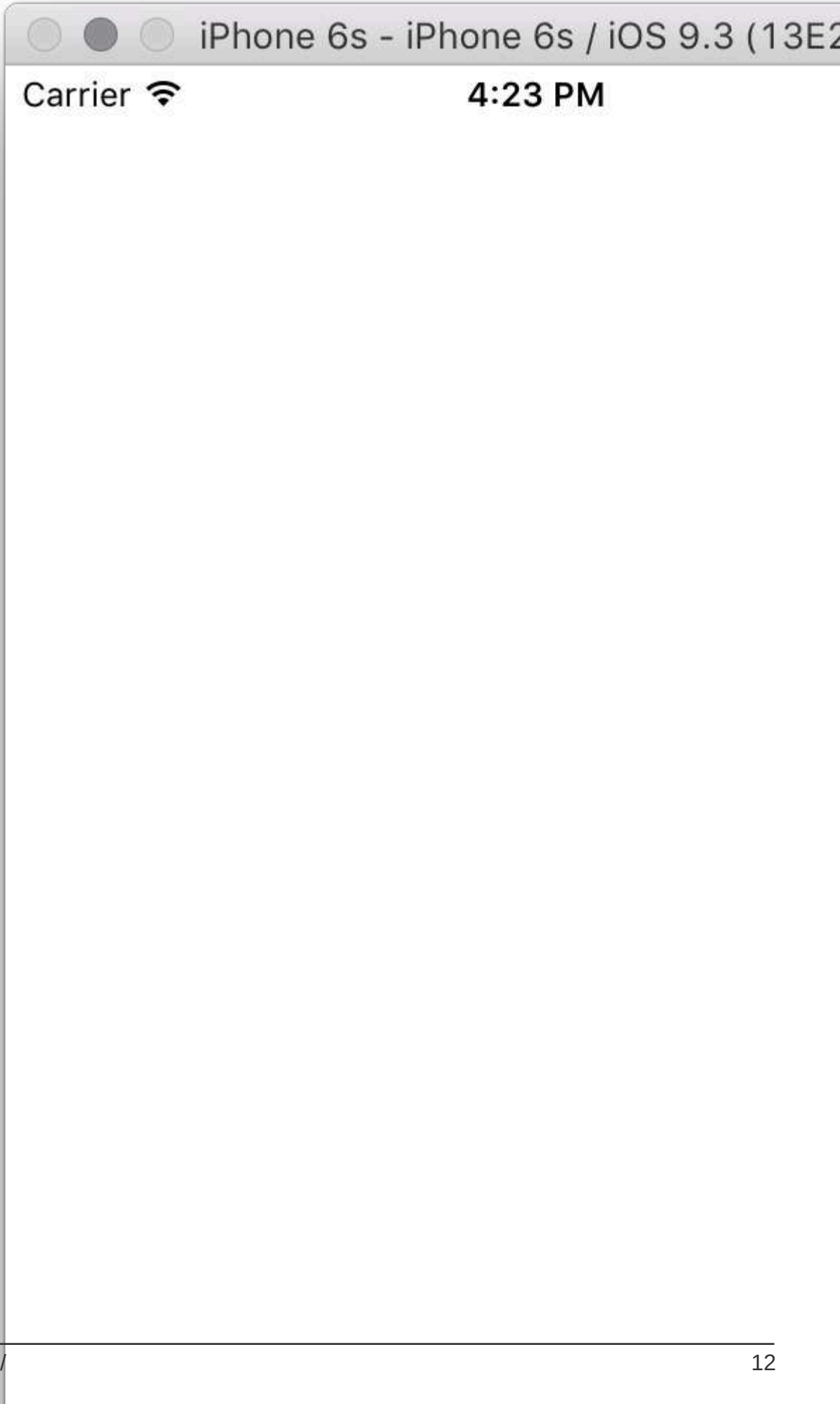


6. To run your application, select the **Debug | iPhoneSimulator** configuration, and press the Play button:

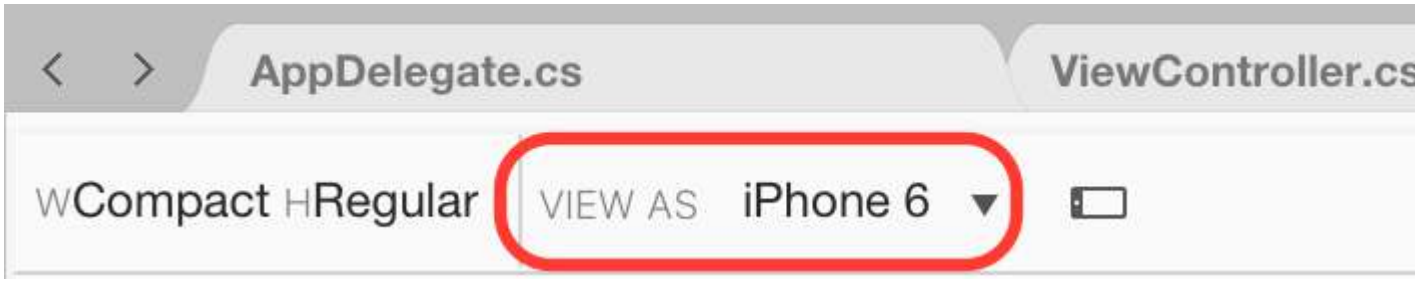
configuration, and press the Play button:

configuration, and press the Play button:

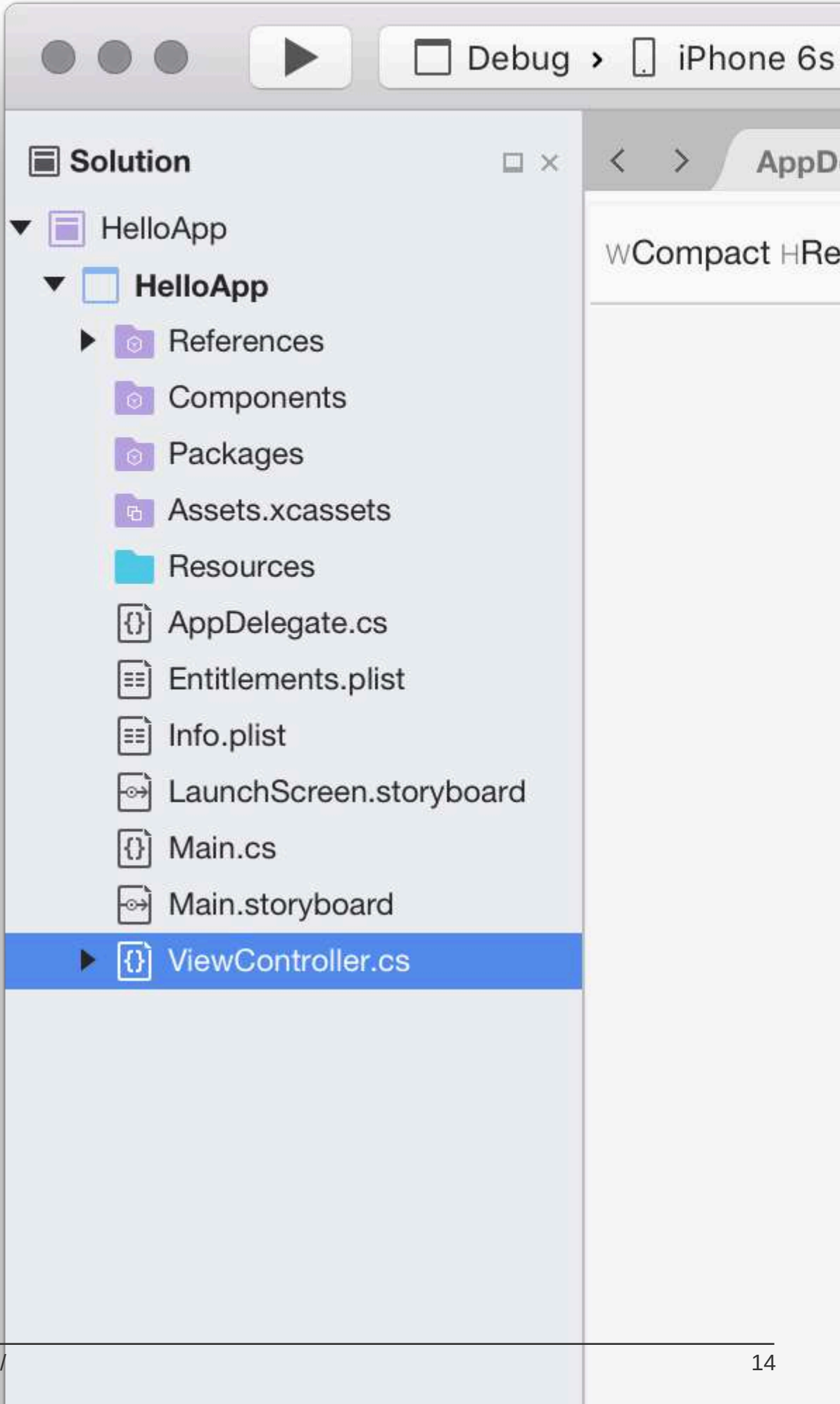
7. This will launch the iOS Simulator on the Mac, and will display your empty application:


iPhone 6s - iPhone 6s / iOS 9.3 (13E2
Carrier
4:23 PM

2. Set **View As** to iPhone 6:



3. Drag a label and a button from the Toolbox to the design surface so that it looks like the image below:


Debug › iPhone 6s
Solution
HelloApp
HelloApp
References
Components
Packages
Assets.xcassets
Resources
AppDelegate.cs
Entitlements.plist
Info.plist
LaunchScreen.storyboard
Main.cs
Main.storyboard
ViewController.cs
AppD
wCompact HRe

4. In the Properties pad, give the label and button the following properties:

nothing	Name	Title
Label	lblClicks	[blank]
Button	clickMe	Click Me!

5. Add the following code to the **ViewDidLoad** method inside the **ViewController** class:

```
clickMe.TouchUpInside += (sender, e) =>
{
    totalClicks++;
    if (totalClicks == 1)
    {
        lblClicks.Text = totalClicks + " Click";
    }

   else {
       lblClicks.Text = totalClicks + " Clicks";
   }
};
```

6. Run the application

Read Getting started with Xamarin.iOS online: https://riptutorial.com/xamarin-ios/topic/402/getting-started-with-xamarin-ios

Chapter 2: Add PullToRefresh to UITableView

Remarks

Object References:

UITableView table;
TableSource tableSource;
bool useRefreshControl = false;
UIRefreshControl RefreshControl;
List tableItems;

TableSource and TableItem are user-defined classes

For the complete sample you can fork: https://github.com/adiiaditya/Xamarin.iOS-Samples/tree/master/PullToRefresh

Examples

Adding UIRefreshControl to UITableView

```
public override async void ViewDidLoad(){
base.ViewDidLoad();
// Perform any additional setup after loading the view, typically from a nib.

Title = "Pull to Refresh Sample";
table = new UITableView(new CGRect(0, 20, View.Bounds.Width, View.Bounds.Height - 20));
//table.AutoresizingMask = UIViewAutoresizing.All;
tableItems = new List<TableItem>();
tableItems.Add(new TableItem("Vegetables") { ImageName = "Vegetables.jpg" });
tableItems.Add(new TableItem("Fruits") { ImageName = "Fruits.jpg" });
tableItems.Add(new TableItem("Flower Buds") { ImageName = "Flower Buds.jpg" });
tableItems.Add(new TableItem("Legumes") { ImageName = "Legumes.jpg" });
tableItems.Add(new TableItem("Tubers") { ImageName = "Tubers.jpg" });
tableSource = new TableSource(tableItems);
table.Source = tableSource;

await RefreshAsync();

AddRefreshControl();

Add(table);
table.Add(RefreshControl);
}

async Task RefreshAsync()
{
// only activate the refresh control if the feature is available
if (useRefreshControl)
    RefreshControl.BeginRefreshing();

if (useRefreshControl)
```

```
    RefreshControl.EndRefreshing();

    table.ReloadData();
}

#region * iOS Specific Code
// This method will add the UIRefreshControl to the table view if
// it is available, ie, we are running on iOS 6+
void AddRefreshControl()
{
if (UIDevice.CurrentDevice.CheckSystemVersion(6, 0))
{
    // the refresh control is available, let's add it
    RefreshControl = new UIRefreshControl();
    RefreshControl.ValueChanged += async (sender, e) =>
    {
        tableItems.Add(new TableItem("Bulbs") { ImageName = "Bulbs.jpg" });
        await RefreshAsync();
    };
    useRefreshControl = true;
   }
}
#endregion
```

Read Add PullToRefresh to UITableView online: https://riptutorial.com/xamarin-ios/topic/6565/add-pulltorefresh-to-uitableview

Chapter 3: Add Search Bar to UITableView

Remarks

Object References:

UITableView table;
TableSource tableSource;
List tableItems;
UISearchBar searchBar;

To fork the complete sample: https://github.com/adiiaditya/Xamarin.iOS-Samples/tree/master/SearchBarWithTableView

Examples

Add UISearchBar to UITableView

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();
    // Perform any additional setup after loading the view, typically from a nib.

    //Declare the search bar and add it to the header of the table
    searchBar = new UISearchBar();
    searchBar.SizeToFit();
    searchBar.AutocorrectionType = UITextAutocorrectionType.No;
    searchBar.AutocapitalizationType = UITextAutocapitalizationType.None;
    searchBar.TextChanged += (sender, e) =>
    {
        //this is the method that is called when the user searches
        searchTable();
    };

    Title = "SearchBarWithTableView Sample";
    table = new UITableView(new CGRect(0, 20, View.Bounds.Width, View.Bounds.Height - 20));
    //table.AutoresizingMask = UIViewAutoresizing.All;
    tableItems = new List<TableItem>();

    tableItems.Add(new TableItem("Vegetables") { ImageName = "Vegetables.jpg" });
    tableItems.Add(new TableItem("Fruits") { ImageName = "Fruits.jpg" });
    tableItems.Add(new TableItem("Flower Buds") { ImageName = "Flower Buds.jpg" });
    tableItems.Add(new TableItem("Legumes") { ImageName = "Legumes.jpg" });
    tableItems.Add(new TableItem("Tubers") { ImageName = "Tubers.jpg" });
    tableSource = new TableSource(tableItems);
    table.Source = tableSource;
    table.TableHeaderView = searchBar;
    Add(table);
}

private void searchTable()
{
    //perform the search, and refresh the table with the results
```

```
    tableSource.PerformSearch(searchBar.Text);
    table.ReloadData();
}
```

The TableSource class will look like this:

```
public class TableSource : UITableViewSource
{
    private List<TableItem> tableItems = new List<TableItem>();
    private List<TableItem> searchItems = new List<TableItem>();
    protected string cellIdentifier = "TableCell";

    public TableSource(List<TableItem> items)
    {
        this.tableItems = items;
        this.searchItems = items;
    }

    public override nint RowsInSection(UITableView tableview, nint section)
    {
        return searchItems.Count;
    }

    public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
    {
        // request a recycled cell to save memory
        UITableViewCell cell = tableView.DequeueReusableCell(cellIdentifier);


        var cellStyle = UITableViewCellStyle.Default;

        // if there are no cells to reuse, create a new one
        if (cell == null)
        {
            cell = new UITableViewCell(cellStyle, cellIdentifier);
        }

        cell.TextLabel.Text = searchItems[indexPath.Row].Title;
        cell.ImageView.Image = UIImage.FromFile("Images/" +
searchItems[indexPath.Row].ImageName);

        return cell;
    }

    public override nint NumberOfSections(UITableView tableView)
    {
        return 1;
    }

    public void PerformSearch(string searchText)
    {
        searchText = searchText.ToLower();
        this.searchItems = tableItems.Where(x =>
x.Title.ToLower().Contains(searchText)).ToList();
    }
}
```

Read Add Search Bar to UITableView online: https://riptutorial.com/xamarin-ios/topic/6540/add-search-bar-to-uitableview

Chapter 4: Adding UIRefreshControl to a table view

Examples

Adding a UIRefreshControl to a TableView

Assumptions:

TableView - reference to the TableView

DataSource - is a class which inherits UITableViewSource

DataSource.Objects - is a public List< object >(), accessible to the UIViewController

```
private UIRefreshControl refreshControl;

public override void ViewDidLoad()
{
    base.ViewDidLoad();

    // Set the DataSource for the TableView
    TableView.Source = dataSource = new DataSource(this);

    // Create the UIRefreshControl
    refreshControl = new UIRefreshControl();

    // Handle the pullDownToRefresh event
    refreshControl.ValueChanged += refreshTable;

    // Add the UIRefreshControl to the TableView
    TableView.AddSubview(refreshControl);
}

private void refreshTable(object sender, EventArgs e)
{
    fetchData();
    refreshControl.EndRefreshing();
    TableView.ReloadData();
}

private void fetchData()
{
    var objects = new List<object>();
    // fetch data and store in objects.
    dataSource.Objects = objects;
}
```

Read Adding UIRefreshControl to a table view online: https://riptutorial.com/xamarin-ios/topic/4642/adding-uirefreshcontrol-to-a-table-view

Chapter 5: Adding UIRefreshControl to a table view

Examples

Addind a simple UIRefreshControl to a UIScrollView

We assume a fully working `UIScrollview` named `_scrollView`;

Note that `UITableView`, `UICollectionView` are also scrollviews, hence the following examples would work on those UI elements.

First, creation & allocation

```
UIRefreshControl refreshControl = new UIRefreshControl();
```

Second, connecting the refresh event to a method. There are different ways to do that.

Style 1:

```
refreshControl.ValueChanged += (object sender, EventArgs e) => MyMethodCall();
```

Style 2:

```
refreshControl.ValueChanged += (object sender, EventArgs e) =>
{
    //Write code here
};
```

Style 3:

```
refreshControl.ValueChanged += HandleRefreshValueChanged;

void HandleRefreshValueChanged(object sender, EventArgs e)
{
    //Write code here
}
```

Third and last, adding the refresh control itself to our scrollview.

```
_scrollView.AddSubview(refreshControl);
```

Read Adding UIRefreshControl to a table view online: https://riptutorial.com/xamarin-ios/topic/8371/adding-uirefreshcontrol-to-a-table-view

Chapter 6: Alerts

Examples

Display an Alert

For Alerts since iOS 8, you would use a `UIAlertController` but for versions before, you would have used a `UIAlertView`, which is now deprecated.

8.0

```
var alert = UIAlertController.Create(title, message, UIAlertControllerStyle.Alert);
alert.AddAction(UIAlertAction.Create(otherTitle, UIAlertActionStyle.Destructive, (action) => {
    // otherTitle();
}));
alert.AddAction(UIAlertAction.Create(cancelTitle, UIAlertActionStyle.Cancel, null));
this.PresentViewController(alert, true, null);
```

8.0

```
var alert = new UIAlertView (title, message, null, cancelTitle, otherTitle);
alert.Clicked += (object sender, UIButtonEventArgs e) => {
    if(e.ButtonIndex == 1)
       // otherTitle();
};
alert.Show ();
```

Display a login alert

The following code is for iOS 8 and lower for creating a login alert.

```
// Create the UIAlertView
var loginAlertView = new UIAlertView(title, message, null, cancelTitle, okTitle);

// Setting the AlertViewStyle to UIAlertViewStyle.LoginAndPasswordInput
loginAlertView.AlertViewStyle = UIAlertViewStyle.LoginAndPasswordInput;

// Getting the fields Username and Password
var usernameTextField = loginAlertView.GetTextField(0);
var passwordTextField = loginAlertView.GetTextField(1);

// Setting a placeholder
usernameTextField.Placeholder = "user@stackoverflow.com";
passwordTextField.Placeholder = "Password";

// Adding the button click handler.
loginAlertView.Clicked += (alertViewSender, buttonArguments) =>
{

    // Check if cancel button is pressed
    if (buttonArguments.ButtonIndex == loginAlertView.CancelButtonIndex)
    {
        // code
```

```
    }

    // In our case loginAlertView.FirstOtherButtonIndex is equal to the OK button
    if (buttonArguments.ButtonIndex == loginAlertView.FirstOtherButtonIndex)
    {
        // code
    }
};

// Show the login alert dialog
loginAlertView.Show();
```

Display an Action Sheet

The `UIAlertController` available since iOS8 allows you to use the same alert object for either Action sheets or more classic alerts. The only difference is the `UIAlertControllerStyle` passed as a parameter when creating.

This line changes from an AlertView to an ActionSheet, compared to some other examples available here :

```
var alert = UIAlertController.Create(title, message, UIAlertControllerStyle.ActionSheet);
```

The way you add actions to the controller is still the same :

```
alert.AddAction(UIAlertAction.Create(otherTitle, UIAlertActionStyle.Destructive, (action) => {
    // ExecuteSomeAction();
}));
alert.AddAction(UIAlertAction.Create(cancelTitle, UIAlertActionStyle.Cancel, null));

//Add additional actions if necessary
```

Note that if you have a parameterless void method, you can use it as the last parameter of the `.AddAction()`.

For example, let's assume I want the code of `private void DoStuff(){...}` to be executed when I press "OK" :

```
UIAlertAction action = UIAlertAction.Create("OK", UIAlertActionStyle.Cancel, DoStuff);
alert.AddAction(action);
```

Notice I'm not using the () after DoStuff in the creation of the action.

The way you present the controller is done the same way as any other controller :

```
this.PresentViewController(alert, true, null);
```

Display Modal Alert Dialog

It was common practice to use `NSRunLoop` to show modal `UIAlertView` to block code execution until user input is processed in iOS; until Apple released the iOS7, it broke few existing apps.

Fortunately, there is a better way of implementing it with C#'s async/await.

Here's the new code taking advantage of async/await pattern to show modal UIAlertView:

```
Task ShowModalAletViewAsync (string title, string message, params string[] buttons)
{
    var alertView = new UIAlertView (title, message, null, null, buttons);
    alertView.Show ();
    var tsc = new TaskCompletionSource ();

    alertView.Clicked += (sender, buttonArgs) => {
        Console.WriteLine ("User clicked on {0}", buttonArgs.ButtonIndex);
        tsc.TrySetResult(buttonArgs.ButtonIndex);
    };
    return tsc.Task;
}

//Usage
async Task PromptUser() {
    var result = await ShowModalAletViewAsync
                ("Alert", "Do you want to continue?", "Yes", "No"); //process the result
}
```

Read Alerts online: https://riptutorial.com/xamarin-ios/topic/433/alerts

Chapter 7: Auto Layout in Xamarin.iOS

Examples

Adding Constraints With iOS 9+ Layout Anchors

9.0

```
// Since the anchor system simply returns constraints, you still need to add them somewhere.
View.AddConstraints(
    new[] {
        someLabel.TopAnchor.ConstraintEqualTo(TopLayoutGuide.GetBottomAnchor()),
        anotherLabel.TopAnchor.ConstraintEqualTo(someLabel.BottomAnchor, 6),
        oneMoreLabel.TopAnchor.ConstraintEqualTo(anotherLabel.BottomAnchor, 6),

oneMoreLabel.BottomAnchor.ConstraintGreaterThanOrEqualTo(BottomLayoutGuide.GetTopAnchor(), -
10),
    }
);
```

Adding Constraints Using Visual Format Language (VFL)

```
// Using Visual Format Language requires a special look-up dictionary of names<->views.
var views = new NSDictionary(
    nameof(someLabel), someLabel,
    nameof(anotherLabel), anotherLabel,
    nameof(oneMoreLabel), oneMoreLabel
);
// It can also take a look-up dictionary for metrics (such as size values).
// Since we are hard-coding those values in this example, we can give it a `null` or empty
dictionary.
var metrics = (NSDictionary)null;

// Add the vertical constraints to stack everything together.
// `V:` = vertical
// `|…|` = constrain to super view (`View` for this example)
// `-10-` = connection with a gap of 10 pixels (could also be a named parameter from the
metrics dictionary)
// `-[viewName]-` = connection with a control by name looked up in views dictionary (using C#
6 `nameof` for refactoring support)
var verticalConstraints = NSLayoutConstraint.FromVisualFormat(
    $"V:|-20-[{nameof(someLabel)}]-6-[{nameof(anotherLabel)}]-6-[{nameof(oneMoreLabel)}]->=10-
|",
    NSLayoutFormatOptions.AlignAllCenterX,
    metrics,
    views
);
View.AddConstraints(verticalConstraints);
```

You may find some constraint types, like aspect ratios, cannot be conveyed in Visual Format Language (VFL) syntax and must call the appropriate methods directly.

Using Cirrious.FluentLayout

Using NuGet

```
Install-Package Cirrious.FluentLayout
```

An expanded example based on the starter example at the GitHub Page, a simple first name, last name labels and fields all stacked one on top of the other:

```
public override void ViewDidLoad()
{
    //create our labels and fields
    var firstNameLabel = new UILabel();
    var lastNameLabel = new UILabel();
    var firstNameField = new UITextField();
    var lastNameField = new UITextField();

    //add them to the View
    View.AddSubviews(firstNameLabel, lastNameLabel, firstNameField, lastNameField);

    //create constants that we can tweak if we do not like the final layout
    const int vSmallMargin = 5;
    const int vMargin = 20;
    const int hMargin = 10;

    //add our constraints
    View.SubviewsDoNotTranslateAutoresizingMaskIntoConstraints();
    View.AddConstraints(
        firstNameLabel.WithSameTop(View).Plus(vMargin),
        firstNameLabel.AtLeftOf(View).Plus(hMargin),
        firstNameLabel.WithSameWidthOf(View),

        firstNameField.WithSameWidth(firstNameLabel),
        firstNameField.WithSameLeft(firstNameLabel),
        firstNameField.Below(firstNameLabel).Plus(vSmallMargin),

        lastNameLabel.Below(firstNameField).Plus(vMargin),
        lastNameLabel.WithSameLeft(firstNameField),
        lastNameLabel.WithSameWidth(firstNameField),

        lastNameField.Below(lastNameLabel).Plus(vSmallMargin),
        lastNameField.WithSameWidth(lastNameLabel),
        lastNameField.WithSameLeft(lastNameLabel));
}
```

Adding Constraints with Masonry

Masonry is a library for objective-c but xamarin have created a binding for it and created it as a nuget package https://www.nuget.org/packages/Masonry/.

Nuget install

```
Install-Package Masonry
```

This centers a button 100 points below the centre point of the containing view and sets a width between 200 and 400 points

```
this.loginBtn.MakeConstraints(make =>
{
    make.Width.GreaterThanOrEqualTo(new NSNumber(200));
    make.Width.LessThanOrEqualTo(new NSNumber(400));
    make.Center.EqualTo(this.View).CenterOffset(new CGPoint(0, 100));
});
```

This sets a scaled image 100 points above the centre point of the containing view then sets the width to the width of the containing view with a muliplier of 0.5 which means 50% of the width. It then sets the height to the width multiplied by the aspect ratio which causes the image to scale but maintain its correct aspect ratio

```
this.logo.MakeConstraints(make =>
{
    make.Center.EqualTo(this.View).CenterOffset(new CGPoint(0, -100));
    make.Width.EqualTo(this.View).MultipliedBy(0.5f);
    make.Height.EqualTo(this.logo.Width()).MultipliedBy(0.71f);
});
```

Read Auto Layout in Xamarin.iOS online: https://riptutorial.com/xamarin-ios/topic/1317/auto-layout-in-xamarin-ios

Chapter 8: Best practices for migrating from UILocalNotification to User Notifications framework

Examples

UserNotifications

1. You will have to import UserNotifications

```
@import UserNotifications;
```

2.Request authorization for localNotification

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization([.alert, .sound]) { (granted, error) in
    // Enable or disable features based on authorization.
}
```

3. Now we will update the application icon badge number

```
@IBAction  func triggerNotification(){
let content = UNMutableNotificationContent()
content.title = NSString.localizedUserNotificationString(forKey: "Tom said:", arguments:
nil)
content.body = NSString.localizedUserNotificationString(forKey: "Hello Mike Let's go.",
arguments: nil)
content.sound = UNNotificationSound.default()
content.badge = UIApplication.shared().applicationIconBadgeNumber + 1;
content.categoryIdentifier = "com.mike.localNotification"
//Deliver the notification in two seconds.
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 1.0, repeats: true)
let request = UNNotificationRequest.init(identifier: "TwoSecond", content: content,
trigger: trigger)

//Schedule the notification.
let center = UNUserNotificationCenter.current()
center.add(request)
}

@IBAction func stopNotification(_ sender: AnyObject) {
let center = UNUserNotificationCenter.current()
center.removeAllPendingNotificationRequests()
}
```

Read Best practices for migrating from UILocalNotification to User Notifications framework online: https://riptutorial.com/xamarin-ios/topic/6382/best-practices-for-migrating-from-uilocalnotification-to-user-notifications-framework

Chapter 9: Binding Swift Libraries

Introduction

An easy to follow guide that will lead you through the process of binding Swift .framework files for use in a Xamarin project.

Remarks

1. When building a library in Xcode it has an option to include the swift libraries. Don't! They will be included in your final app as NAME.app/Frameworks/LIBRARY.framework/Frameworks/libswift*.dylib but they must be included as NAME.app/Frameworks/libswift*.dylib

2. You can find this information elsewhere, but it's worth mention: Don't include Bitcode in the library. As of right now Xamarin don't include Bitcode for iOS and Apple requires all libraries to support the same architectures.

Examples

Binding a Swift Library in Xamarin.iOS

Binding a Swift Library in Xamarin.iOS follows the same process for Objective-C as shown in https://developer.xamarin.com/guides/ios/advanced_topics/binding_objective-c/, but with some caveats.

1. A swift class must inherit from NSObject to be binded.
2. Swift compiler will translate class and protocol names into something else unless you use the @objc annotation (e.g. @objc(MyClass)) in your swift classes to specify the explicit objective c name.
3. In runtime your APP must include some swift core libraries alongside your binded framework in a folder called Frameworks;
4. When the App is pushed to AppStore it must include a SwiftSupport folder alongside your Payload folder. Those are inside the IPA file.

Here you can find a simple sample binding:
https://github.com/Flash3001/Xamarin.BindingSwiftLibrarySample

And a full binding sample: https://github.com/Flash3001/iOSCharts.Xamarin

Please find steps below:

1.1 Prepare the Swift classes you want to

export

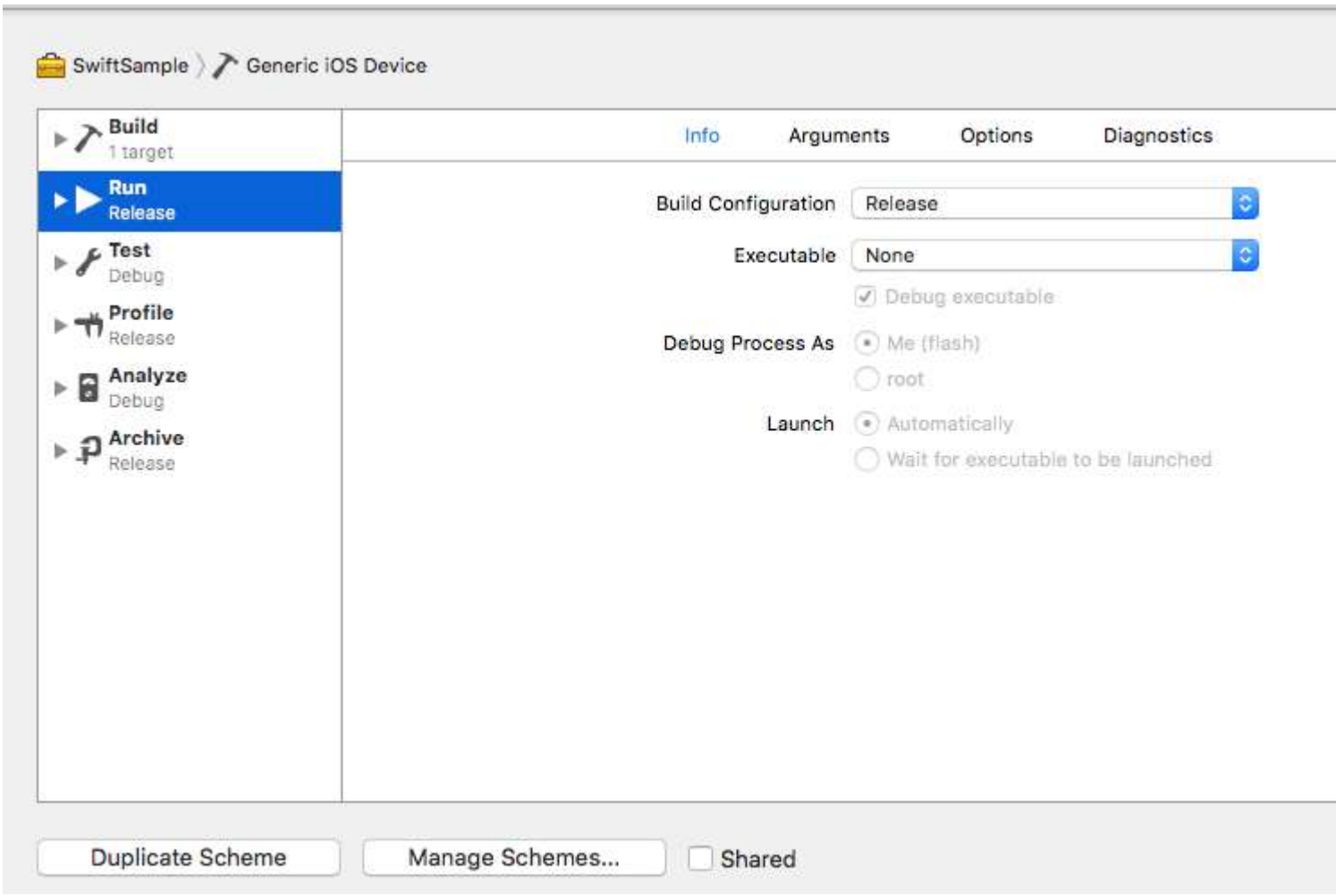
For any Swift class you want to use you either have to inherit from NSObject and make the Objective-C name explicit using the objc annotation. Otherwise the Swift compiler will generate different names. Below is an example code of how a Swift class could look like. Note that it doesn't matter which class it inherits as long as the root class inherits from NSObject.

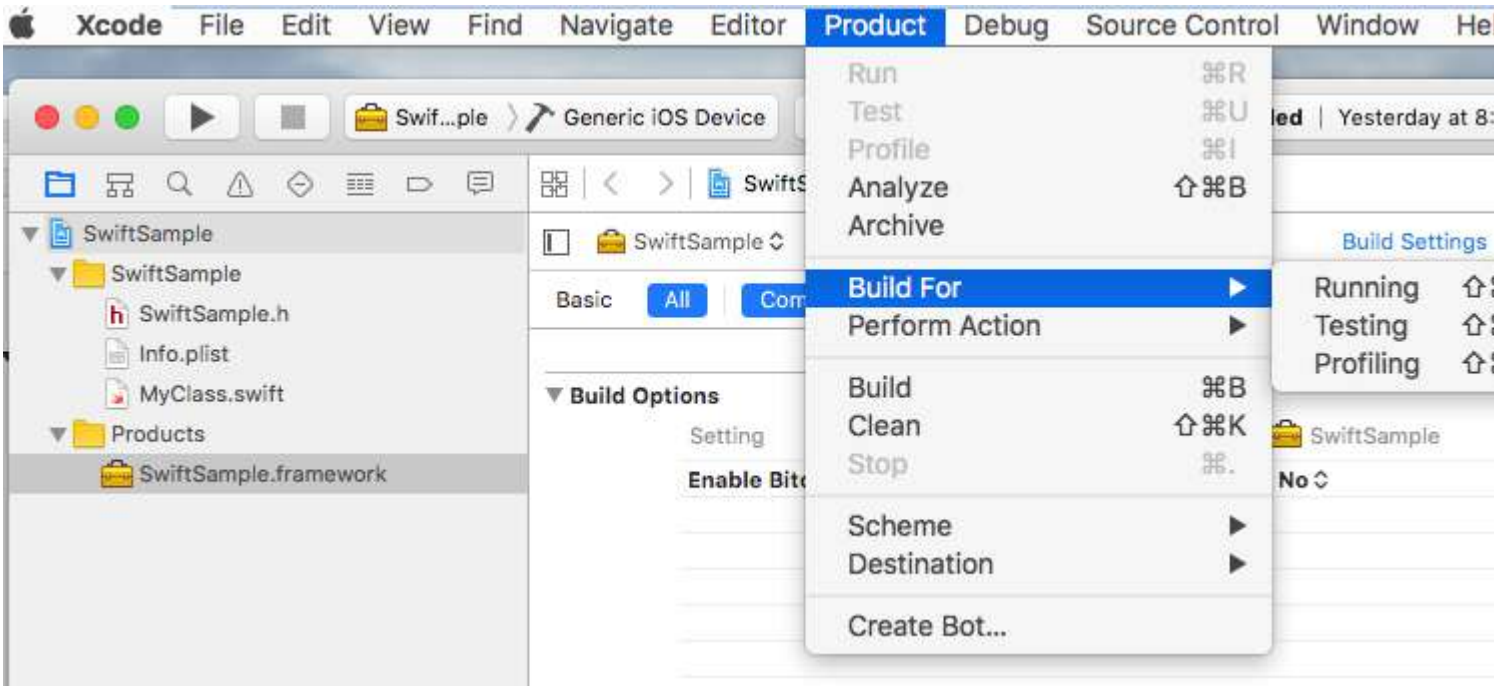
```
//Add this to specify explicit objective c name
@objc(MyClass)
open class MyClass: NSObject {
    open func getValue() -> String
    {
        return "Value came from MyClass.swift!";
    }
}
```

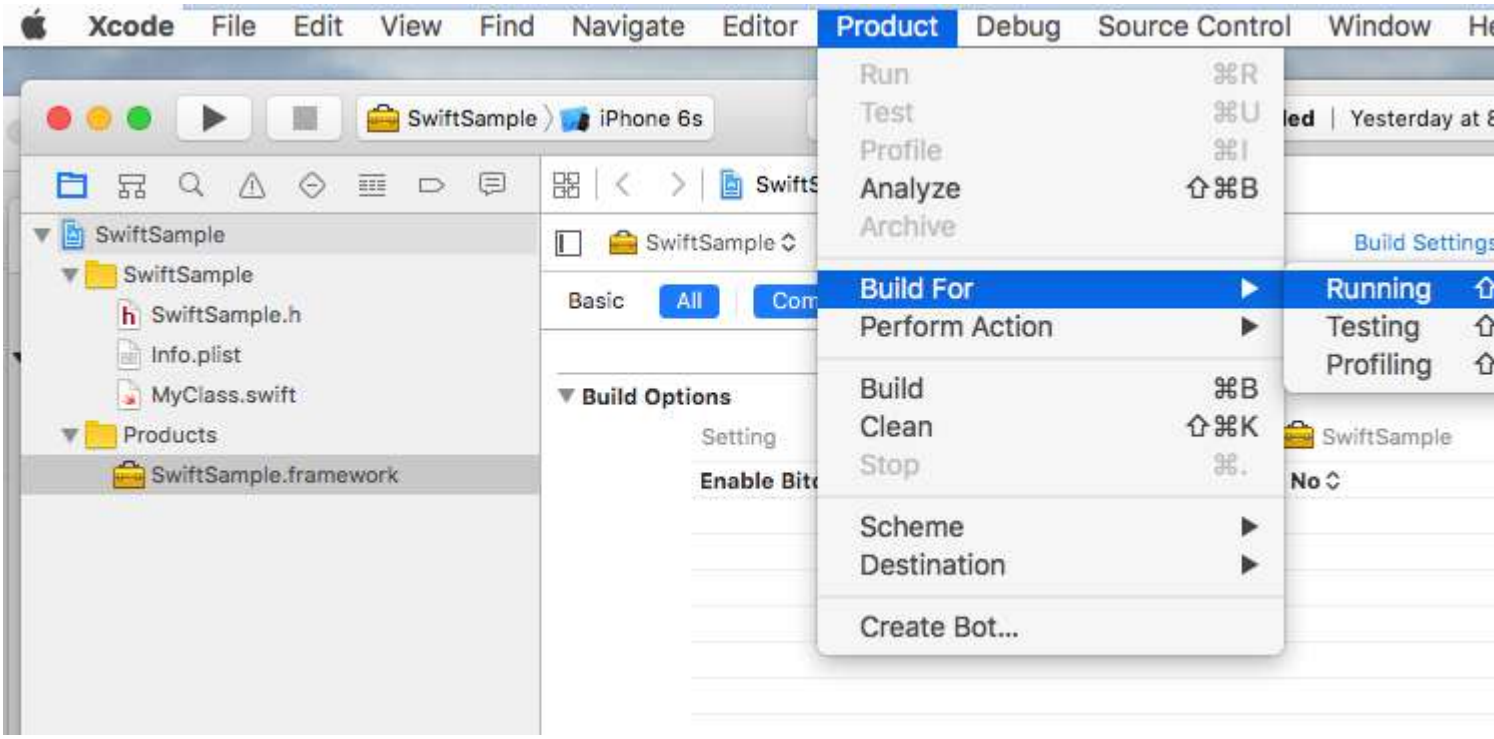
1.2 Build the framework

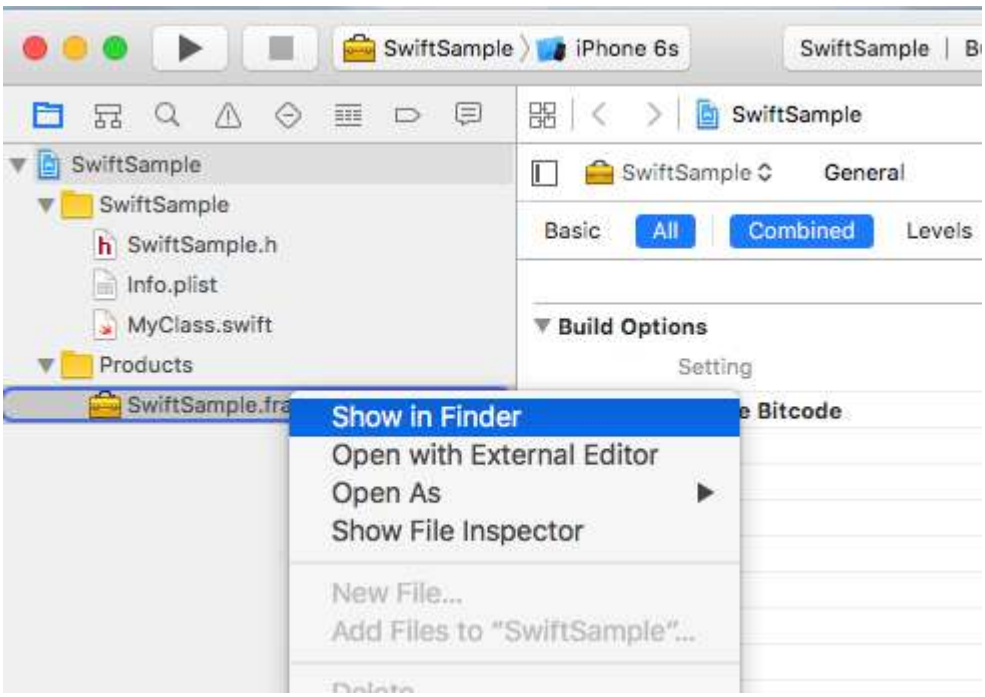
Disable Bitcode. *

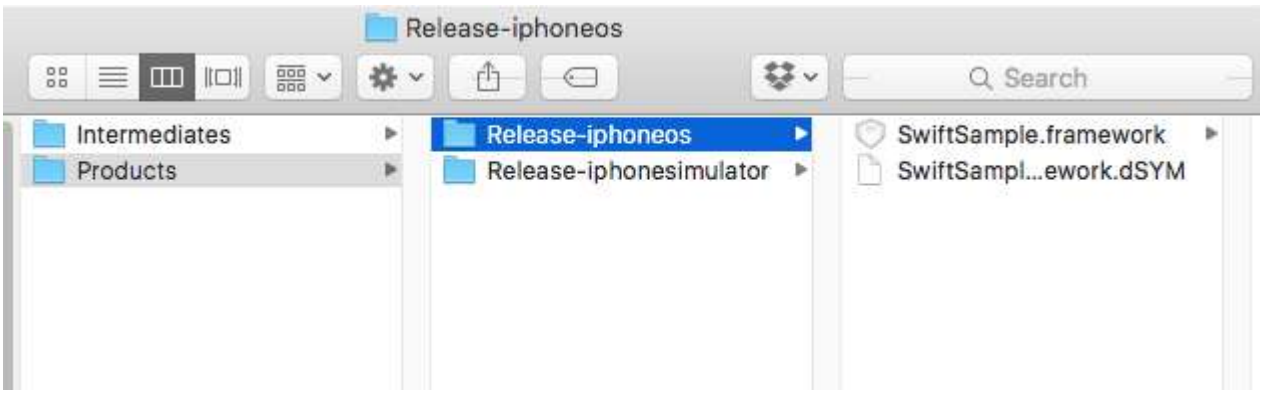
Build for release for Device and Simulator. *


SwiftSample
Generic iOS Device
Build
1 target
Run
Release
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Info
Arguments
Options
Diagnostics
Build Configuration
Release
Executable
None
Debug executable
Debug Process As
Me (flash)
root
Launch
Automatically
Wait for executable to be launched
Duplicate Scheme
Manage Schemes...
Shared


Xcode
File
Edit
View
Find
Navigate
Editor
Product
Debug
Source Control
Window
Run ⌘R
Test ⌘U
Profile ⌘I
Analyze ⇧⌘B
Archive
Build For
Perform Action
Build ⌘B
Clean ⇧⌘K
Stop ⌘.
Scheme
Destination
Create Bot...
SwiftSample
SwiftSample
SwiftSample.h
Info.plist
MyClass.swift
Products
SwiftSample.framework
Basic
All
Build Options
Build Settings
Running
Testing
Profiling





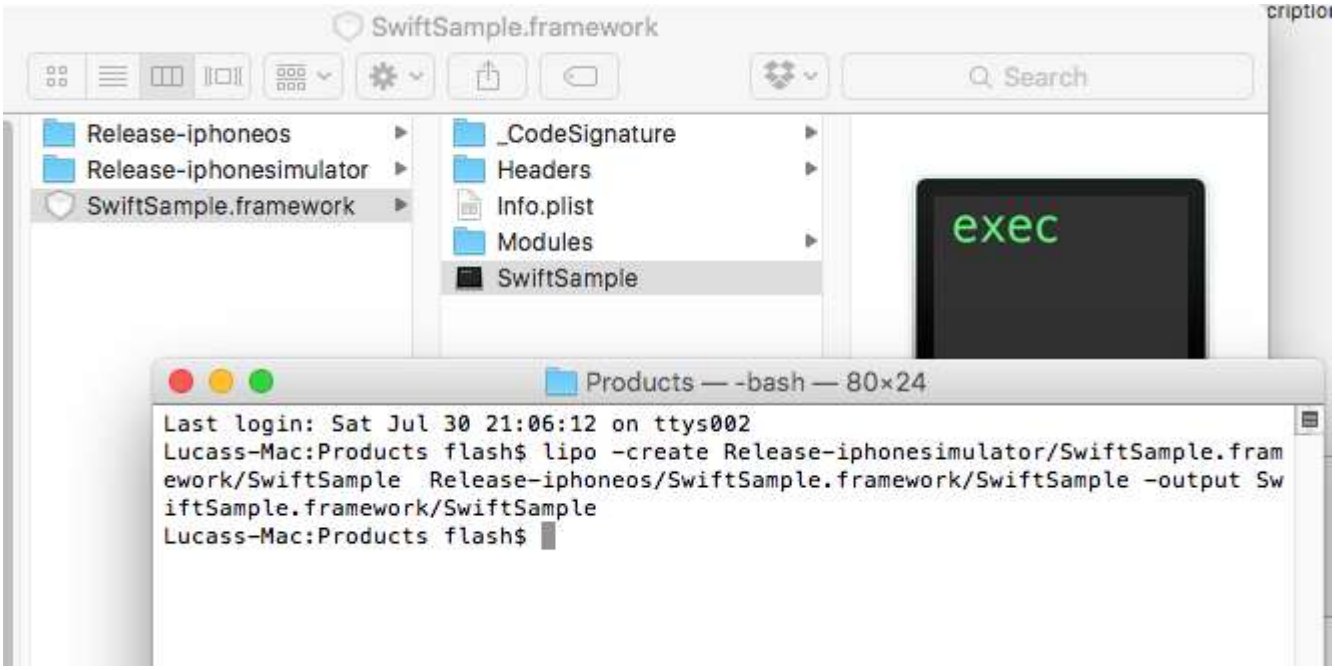


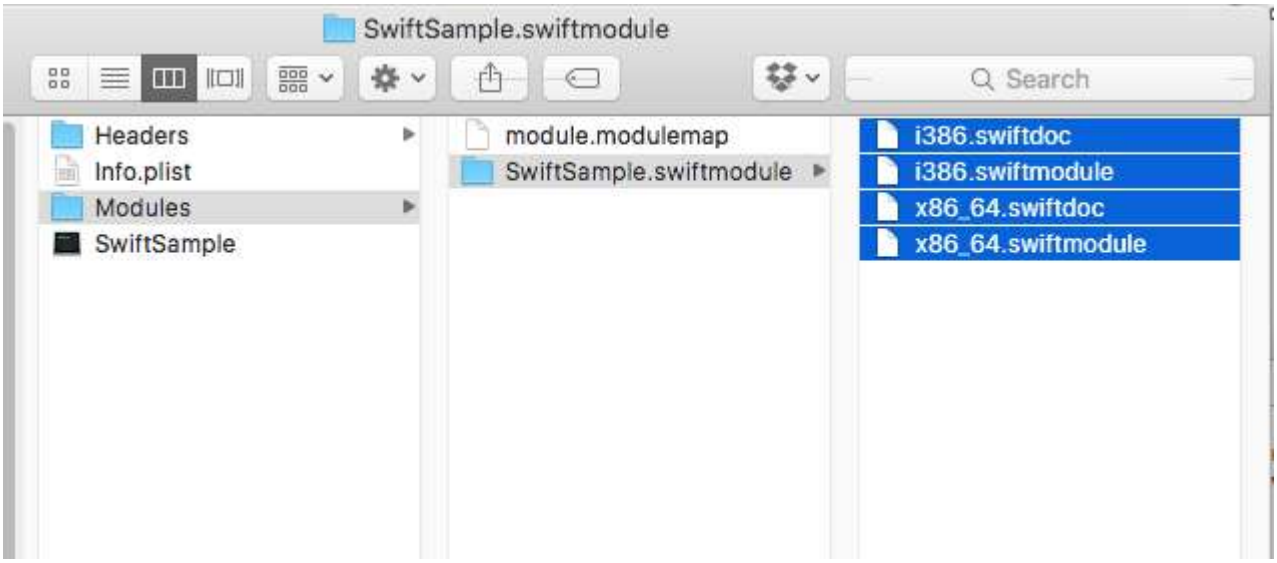
- Not related only to Swift binding.

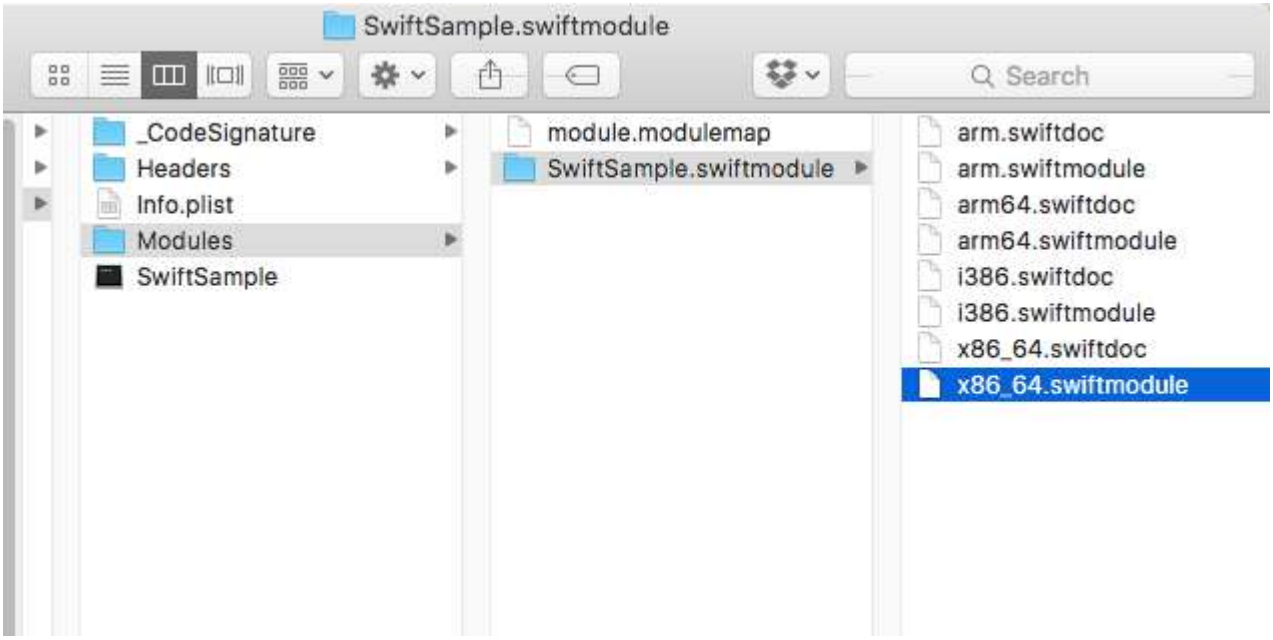
2. Create a fat library

A framework contains several files, the one that needs to eat a little is NAME.framework/NAME (without extension).

- Copy Release-iphoneos/NAME.framework to NAME.framework
- Create the FAT library using:
    - **lipo -create Release-iphonesimulator/NAME.framework/NAME Release-iphoneos/NAME.framework/NAME -output NAME.framework/NAME**
- Copy the files in Release-iphonesimulator/NAME.framework/Modules/NAME.swiftmodule to NAME.framework/Modules/NAME.swiftmodule (until now it only contained files from the iphoneos)



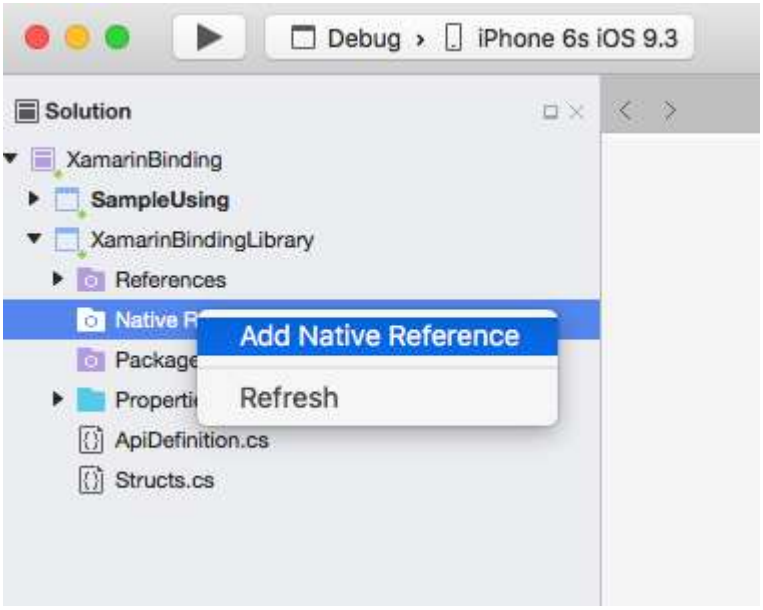


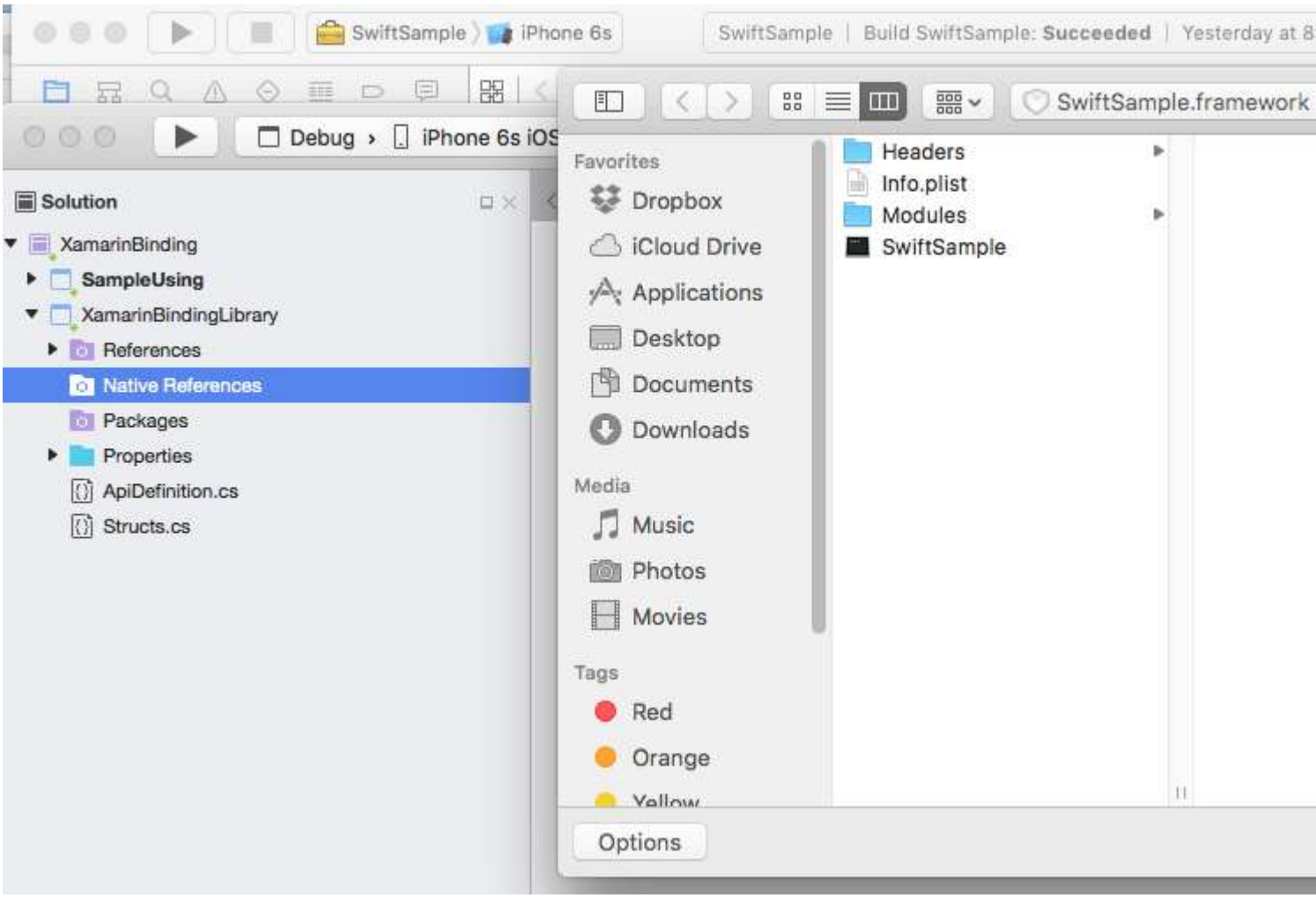


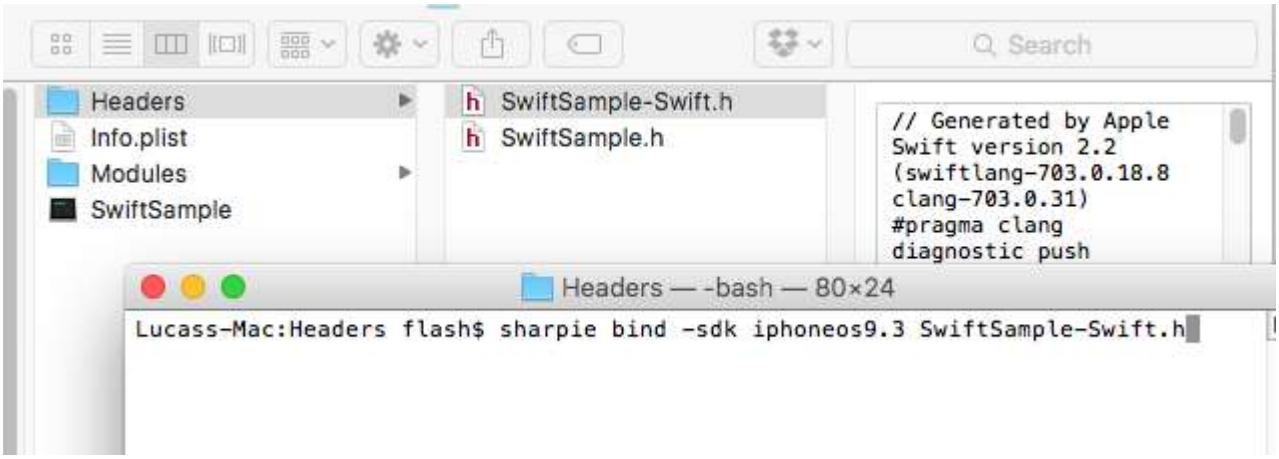
3. Import the library

I'll assume you already created the Binding project in File -> New -> iOS -> Binding Library.

Xamarin support importing .frameworks. Just right click 'Native References' and click in 'Add native reference'. Find the newly created fat framework and add it.







4. Create the ApiDefinition based on LIBRARY-Swift.h file inside headers.

You can do it manually, but it won't be nice. You can use Objetive Sharpie. The tool Xamarin uses to bind its own libraries.

How to use it on https://developer.xamarin.com/guides/cross-platform/macios/binding/objective-sharpie/

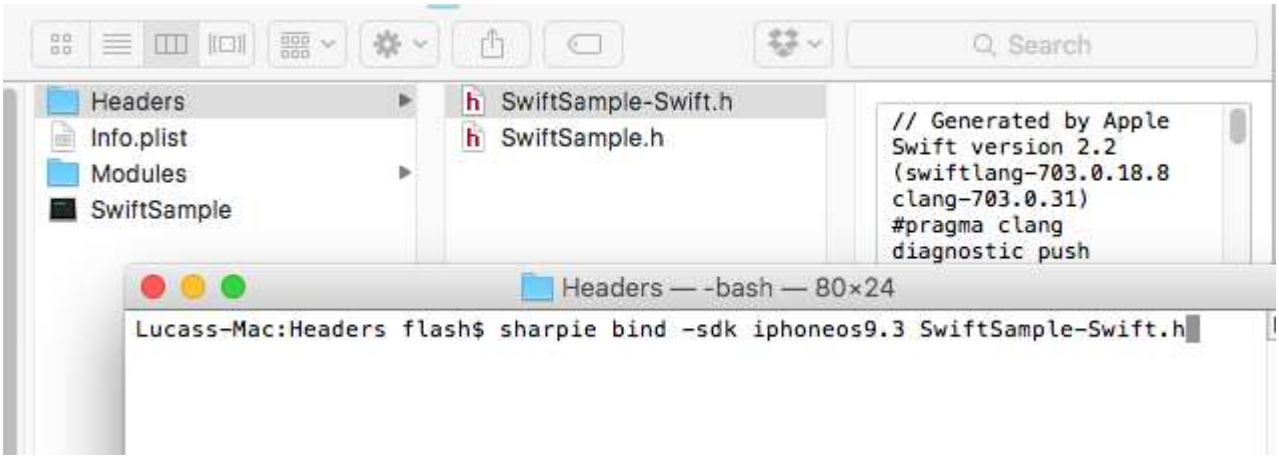
The basic command will be something like: ***sharpie bind -sdk iphoneos9.3 NAME-Swift.h***

If you get a `System.Reflection.TargetInvocationException` it is probably because you have a different SDK version installed. Run the following command to check with iPhone OS SDK you have installed:

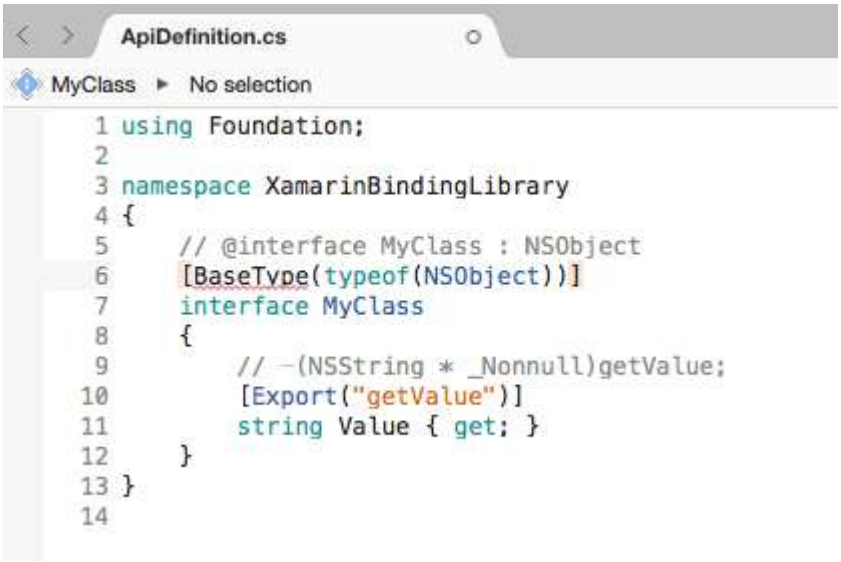
```
sharpie xcode -sdks
```

The file ***NAME-Swift.h*** is located in ***NAME.framework/Headers/NAME-Swift.h***

Note: The swift classes must inherit from "NSObject", otherwise ***NAME-Swift.h*** won't import your classes and Objetive Sharpie will convert nothing.



Replace the contents of your binding project ApiDefinition.cs with the newly created one.



5. Change all [Protocol] and [BaseType] to include the class' name in Objective-C runtime.

In case the original Swift class or protocol doesn't include the @objc(MyClass) annotation as specified in step 1.1 they will have its internal Objective-C names changed, so you need to map it to the right one.

All names are available in the file NAME-Swift.h in the following format:

```
SWIFT_CLASS("_TtC11SwiftSample7MyClass")
@interface MyClass : NSObject
```

And

```
SWIFT_PROTOCOL("_TtP6Charts17ChartDataProvider_")
@protocol ChartDataProvider
```

To set the name you use BaseTypeAttribute.Name https://developer.xamarin.com/guides/cross-platform/macios/binding/binding-types-reference/#BaseType.Name property for classes and ProcotolAttribute.Name https://developer.xamarin.com/api/property/MonoTouch.Foundation.ProtocolAttribute.Name/ for protocols.

```
[BaseType(typeof(NSObject), Name = "_TtC11SwiftSample7MyClass")]
interface MyClass
```

Doing it manually is not cool. You can use this tool https://github.com/Flash3001/SwiftClassify to insert all the names. (It’s work in progress. But it’s quite simple, just by looking at the code you will get how it works).

6.1 Include all Swift dependencies to run.

If you try to consume the library in an App and try to run it right now it will crash. The error is due to the lack of libswiftCore.dylib

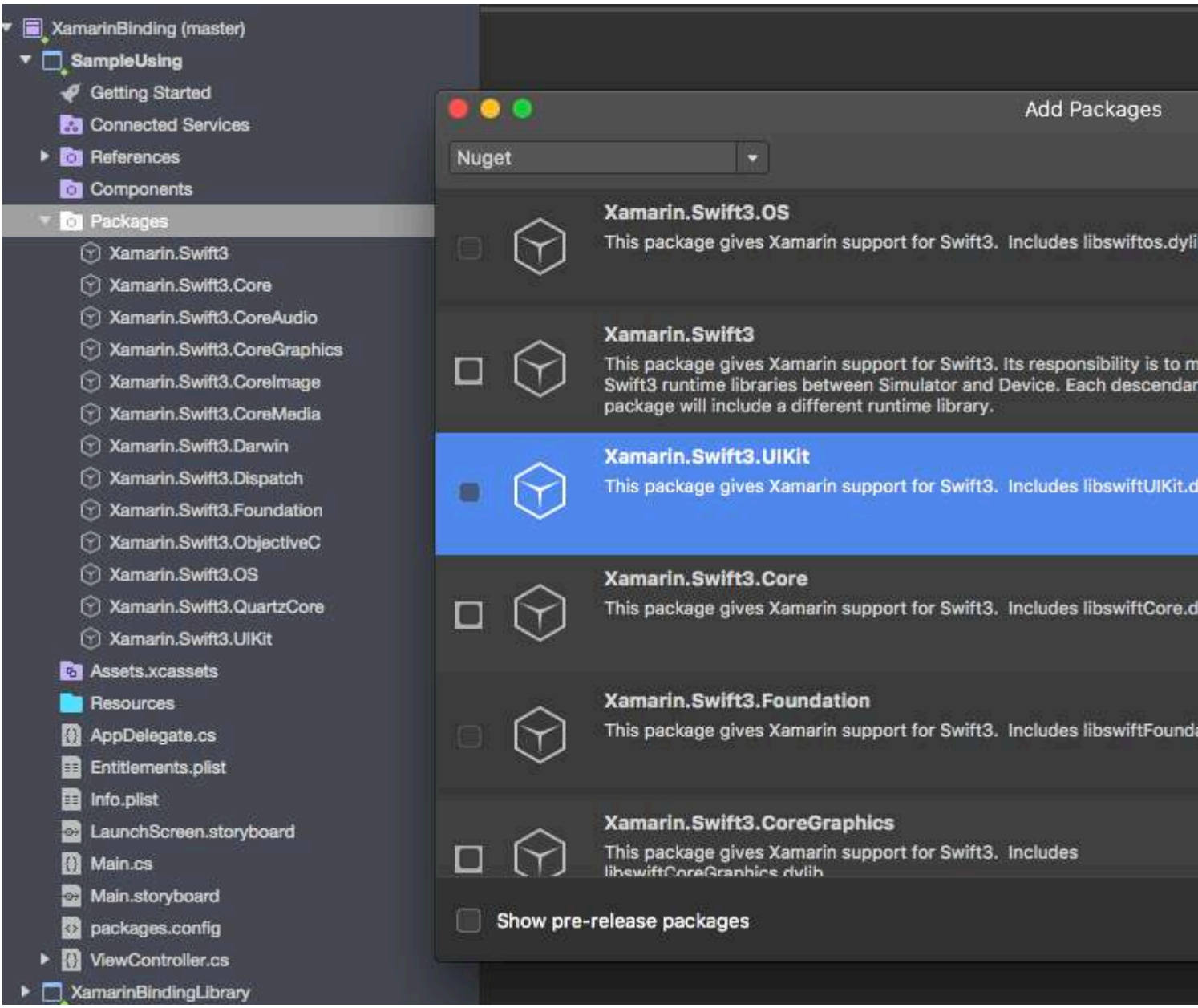
Something like this:

```
Dyld Error Message:
  Library not loaded: @rpath/libswiftCore.dylib
  Referenced from: /Users/USER/Library/Developer/CoreSimulator/Devices/AC440891-C819-4050-
8CAB-CE15AB4B3830/data/Containers/Bundle/Application/27D2EC87-5042-4FA7-9B80-
A24A8971FB48/SampleUsing.app/Frameworks/SwiftSample.framework/SwiftSample
  Reason: image not found
```

Xamarin.iOS doesn’t give official support for binding a Swift library. So you must manually include the swift core libraries in the Frameworks and SwiftSupport folders. The files for the Frameworks folder are different for Simulator and Device. They can be found in /Applications/Xcode.app/Contents/Developer//XcodeDefault.xctoolchain/usr/lib/swift.Toolchains

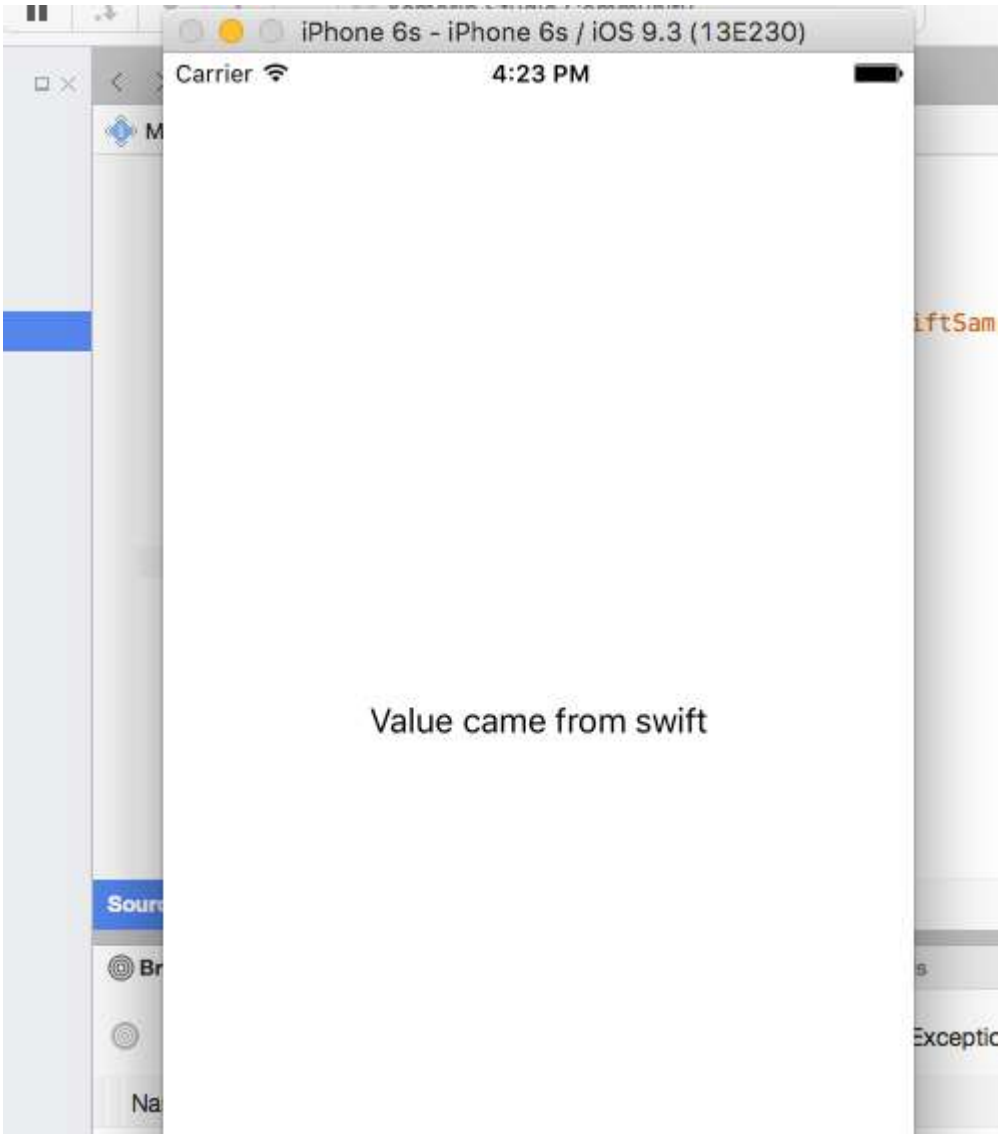
Instead of manually copying the files inside the Framework folder you can use this library https://github.com/Flash3001/Xamarin.Swift3.Support . It includes every single dependency Swift

3.1 needs, each one in a single NuGet package.



As you can see, the Nuget Package is included in the consumer App, not in the binding itself. If you try to include it in the binding you will get compile errors.

If you are building a Nuget Package you can instruct Nuget to include it as a dependency.



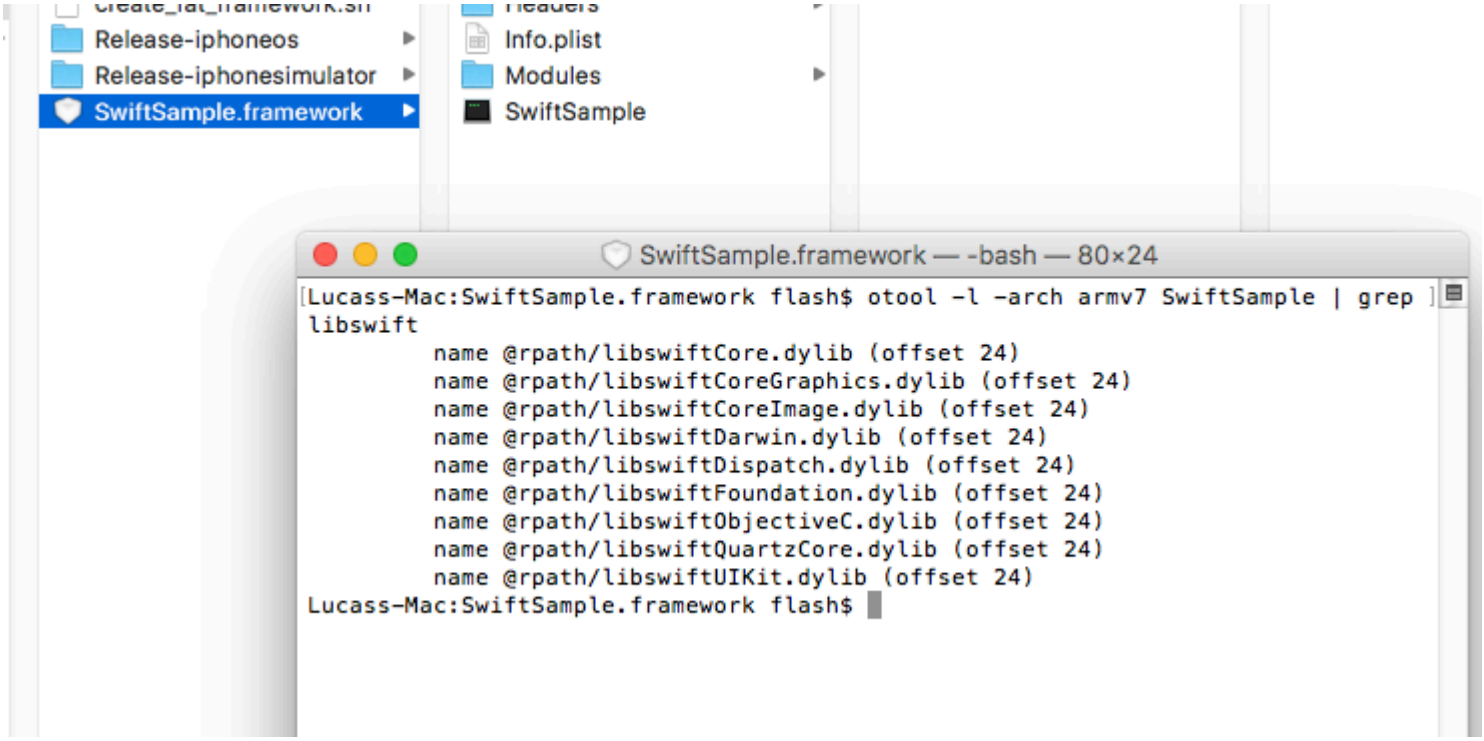
6.2. Finding out which Swift dependencies to include.

An important thing to do is to figure out each package you need to include in your project. A simple binding will usually need:

```
    libswiftCore.dylib
    libswiftCoreGraphics.dylib
    libswiftCoreImage.dylib
    libswiftDarwin.dylib
    libswiftDispatch.dylib
    libswiftFoundation.dylib
    libswiftObjectiveC.dylib
    libswiftQuartzCore.dylib
    libswiftUIKit.dylib
```

To list each dependency you can run the following command inside your LibraryName.framework

```
otool -l -arch armv7 LibraryName | grep libswift
```



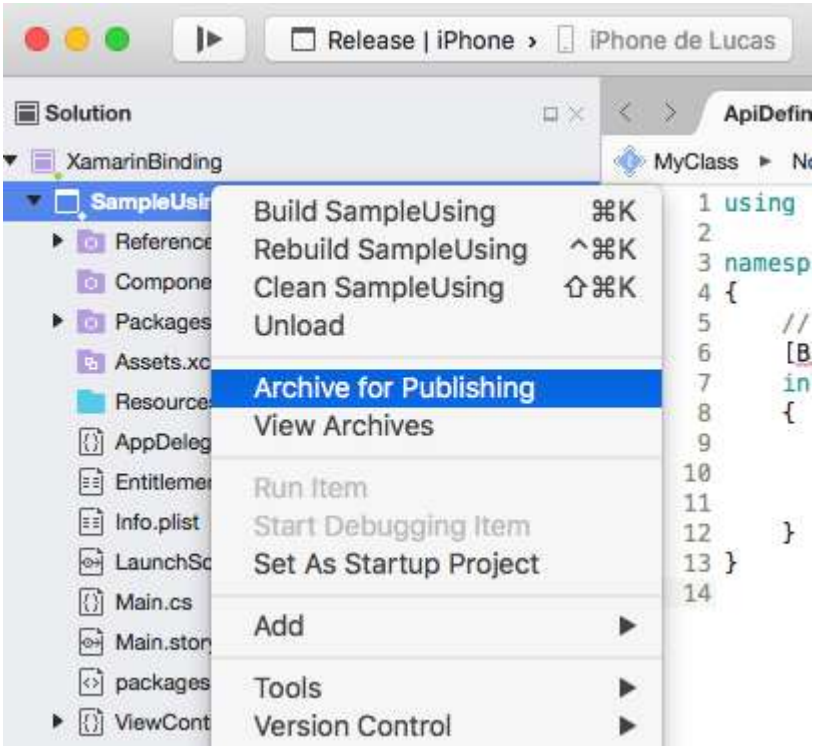
Do not include every package available in NuGet for Swift3 as they might grow your app size.

7. Include SwiftSupport to push App to AppStore.

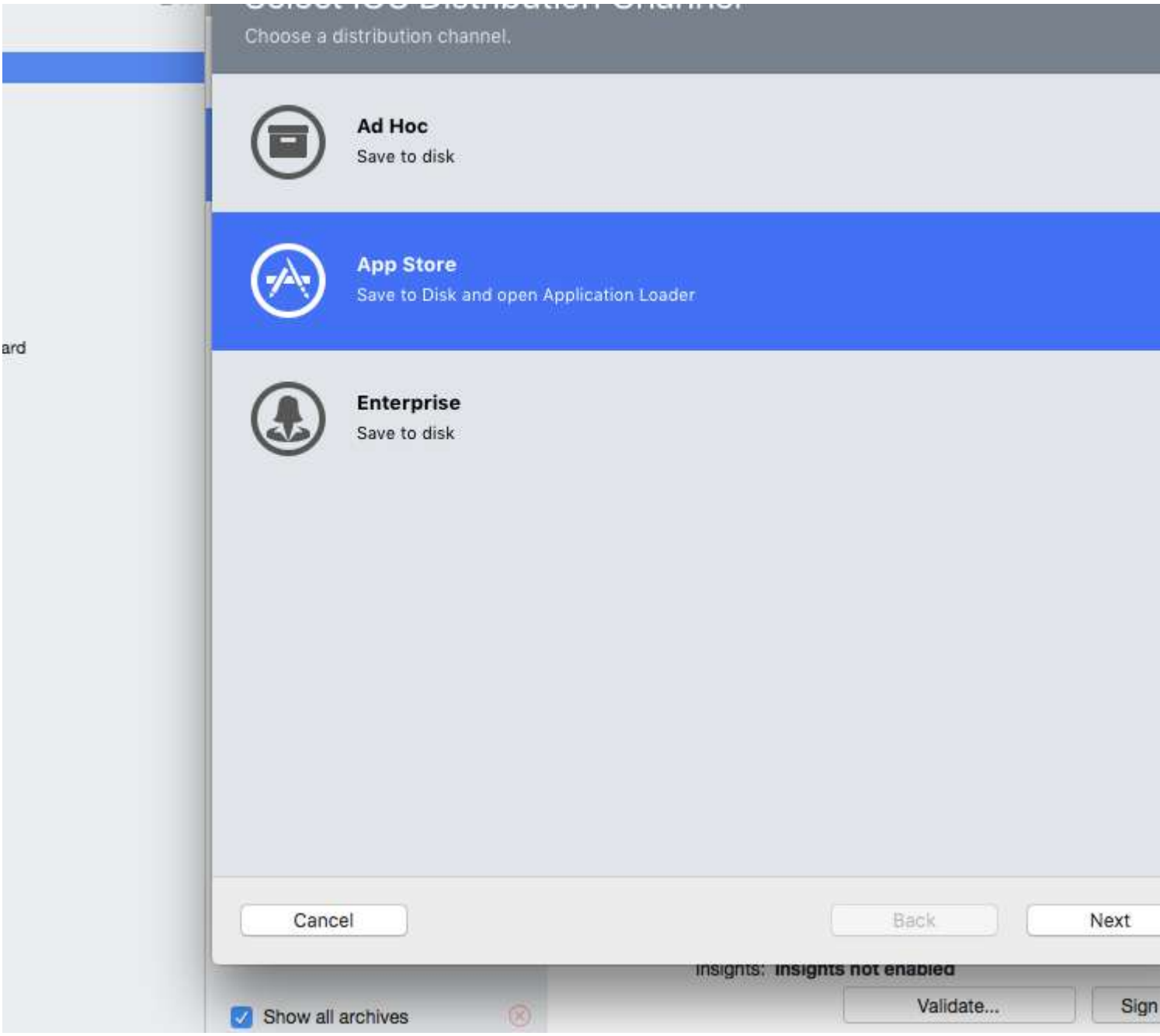
Apple requires your App to be sent with a SwiftSupport folder alongside your Payload folder. Both are inside your IPA package.

You can use this script https://github.com/bq/ipa-packager to do this work for you.

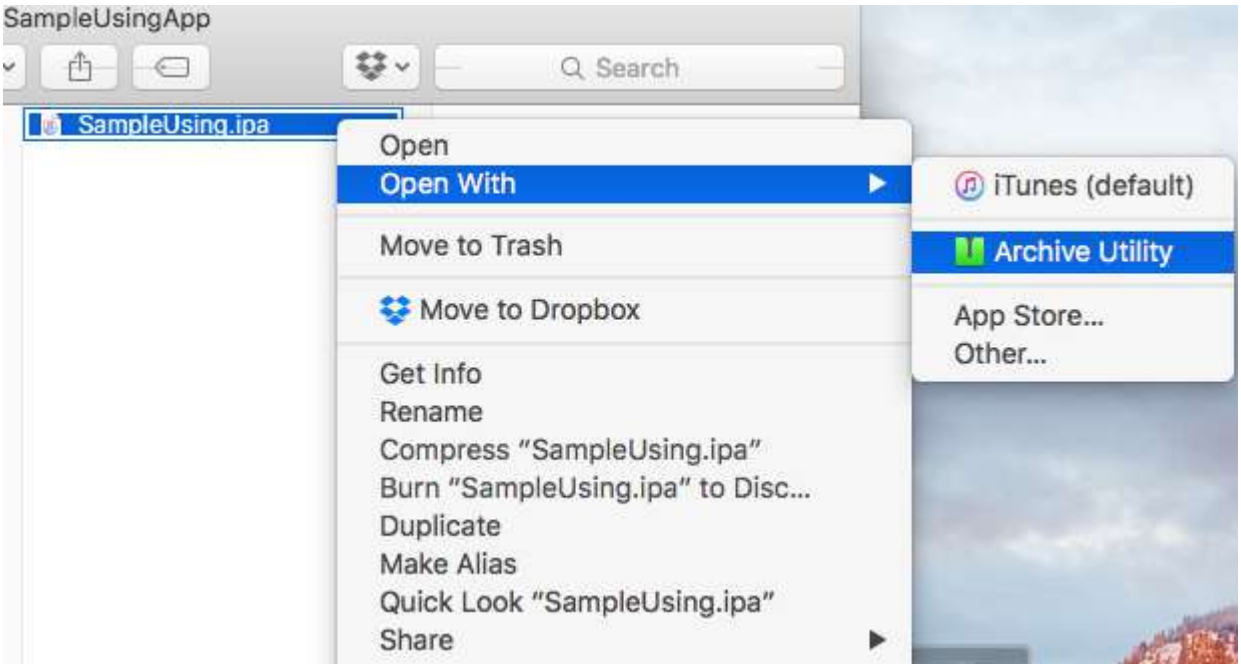
This process is the only one the library consumer will have to do manually. Every time he/she tries to push the App to AppStore.



Click 'Sign and Distribute' and Save to Disk

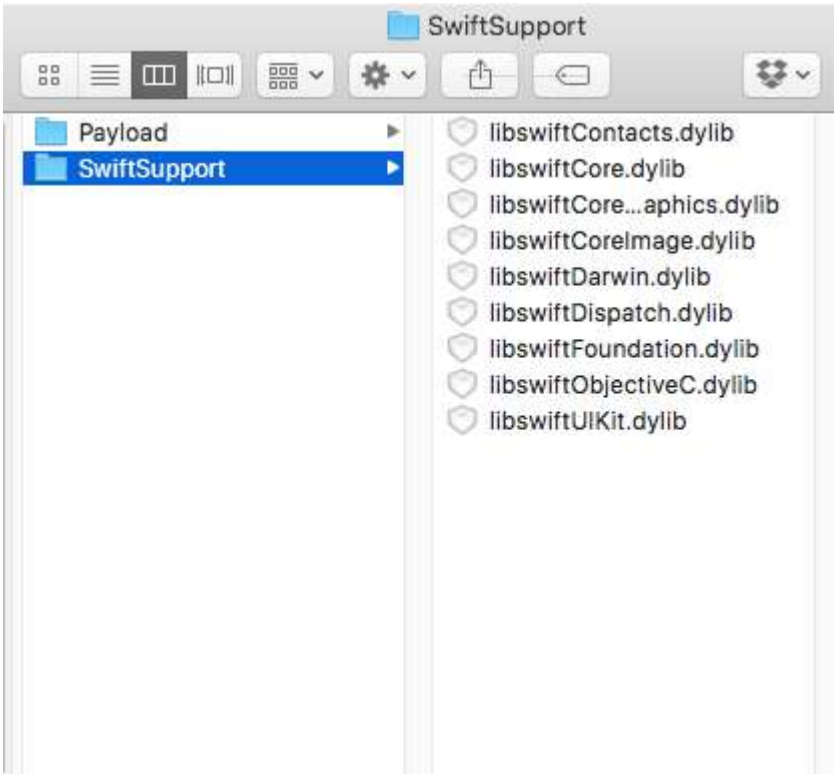


Unzip your .IPA



Create the new IPA using the script before mentioned

If you Unzip the file know, it will contain the SwiftSupport folder.



Remarks

When building a library in Xcode it has an option to include the swift libraries. Don't! They will be included in your final app as NAME.app/Frameworks/LIBRARY.framework/Frameworks/libswift*.dylib but they must be included as NAME.app/Frameworks/libswift*.dylib

You can find this information elsewhere, but it's worth mention: Don't include Bitcode in the library. As of right now Xamarin don't include Bitcode for iOS and Apple requires all libraries to support the same architectures.

Disclaimer

This guide is originally created by Lucas Teixeira. All credits belong to him. Thank you, Lucas.

Read Binding Swift Libraries online: https://riptutorial.com/xamarin-ios/topic/6091/binding-swift-libraries

Chapter 10: Calculating variable row height in GetHeightForRow

Remarks

Calculating row heights might be expensive and scrolling performance can suffer if you have larger amounts of data. In that scenario, override `UITableViewSource.EstimatedHeight(UITableView, NSIndexPath)` to quickly provide a number sufficient for rapid scrolling, e.g.,:

```
public override nfloat EstimatedHeight(UITableView tableView, NSIndexPath indexPath)
{
    return 44.0f;
}
```

Examples

Using GetHeightForRow

To set a custom row height, override `UITableViewSource.GetHeightForRow(UITableView,NSIndexPath)`:

```
public class ColorTableDataSource : UITableViewSource
{
    List<DomainClass> Model { get; set; }

    public override nfloat GetHeightForRow(UITableView tableView, NSIndexPath indexPath)
    {
        var height = Model[indexPath.Row % Model.Count].Height;
        return height;
    }

    //...etc ...
}
```

The domain class for the table (in this case, it has 1 of 3 random colors and 1 of 3 random heights):

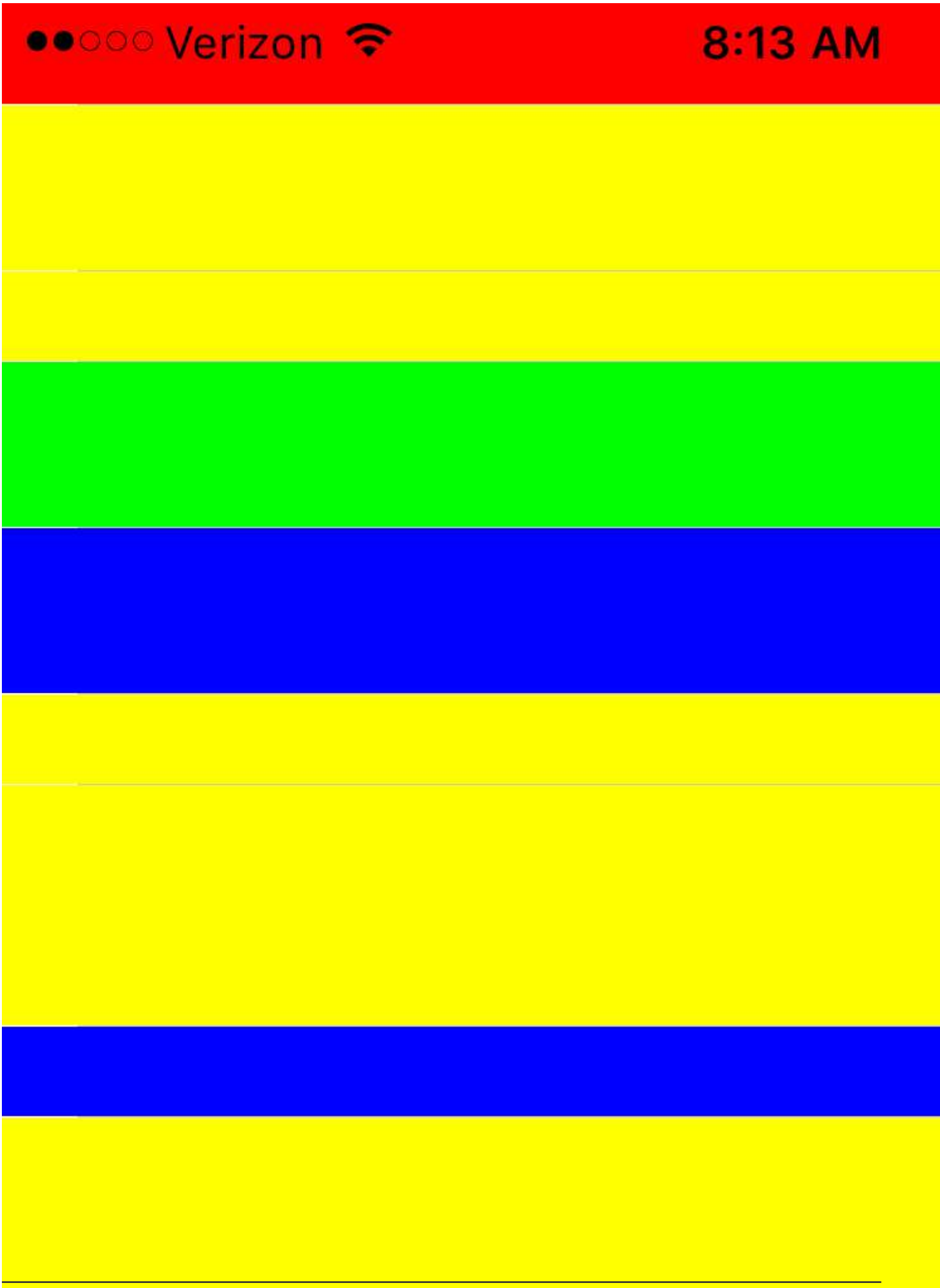
```
public class DomainClass
{
    static Random rand = new Random(0);
    public UIColor Color { get; protected set; }
    public float Height { get; protected set; }

    static UIColor[] Colors = new UIColor[]
    {
        UIColor.Red,
        UIColor.Green,
        UIColor.Blue,
        UIColor.Yellow
    };
```

```
    public DomainClass()
    {
        Color = Colors[rand.Next(Colors.Length)];
        switch (rand.Next(3))
        {
            case 0:
                Height = 24.0f;
                break;
            case 1:
                Height = 44.0f;
                break;
            case 2:
                Height = 64.0f;
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
    }

    public override string ToString()
    {
        return string.Format("[DomainClass: Color={0}, Height={1}]", Color, Height);
    }
}
```

Which looks like:


Verizon
8:13 AM

https://riptutorial.com/xamarin-ios/topic/6515/calculating-variable-row-height-in-getheightforrow

Chapter 11: Concurrent Programming in Xamarin.iOS

Examples

Manipulating UI from background threads

Background threads cannot modify the UI; almost all UIKit methods must be called on the main thread.

From a subclass of `NSObject` (including any `UIViewController` or `UIView`):

```
InvokeOnMainThread(() =>
{
    // Call UI methods here
});
```

From a standard C# class:

```
UIApplication.SharedApplication.InvokeOnMainThread(() =>
{
    // Call UI methods here
});
```

`InvokeOnMainThread` waits for your code running on the main thread to execute before continuing. If you don't need to wait, use `BeginInvokeOnMainThread`.

Using Async and await

You can use async methods to handle asynchronous executions. For example POST and GET requests. Let say below is your get data method.

```
Task<List> GetDataFromServer(int type);
```

You can call that method as shown below

```
var result = await GetDataFromServer(1);
```

However, in real day practice this method will be in a service layer interface. There for best way to do that is create a separate method to call this and update UI shown below.

```
//Calling from viewDidLoad
void async ViewDidLoad()
{
    await GetDataListFromServer(1);
    //Do Something else
```

```
}

//New method call to handle the async task
private async Task GetArchivedListFromServer(int type)
{
    var result = await GetDataFromServer(type);
    DataList.AddRange(result.toList());
    tableView.ReloadData();
}
```

In the above code snippet, GetDataListFromServer method will get called and it will send the web request. Nevertheless, it will not block the UI thread till it gets the response from the server. It will move down the line after `await GetDataListFromServer(1)`. However, inside the `private async Task GetArchivedListFromServer(int type)` method, it will wait till it gets the response from the server in order to execute the lines after `var result = await GetDataFromServer(type);`.

Read Concurrent Programming in Xamarin.iOS online: https://riptutorial.com/xamarin-ios/topic/1364/concurrent-programming-in-xamarin-ios

Chapter 12: Connecting with Microsoft Cognitive Services

Remarks

In this example we used Microsoft.ProjectOxford.Vision NuGet package: https://www.nuget.org/packages/Microsoft.ProjectOxford.Vision/

To read more about Microsoft Cognitive Services please refer to the official documentation: https://www.microsoft.com/cognitive-services/en-us/computer-vision-api

Please find uploaded sample on my GitHub: https://github.com/Daniel-Krzyczkowski/XamarinIOS/tree/master/XamariniOS_CognitiveServices

I also attach link to my blog where I presented how to use Cognitive Services with Xamarin Forms application: http://mobileprogrammer.pl

Examples

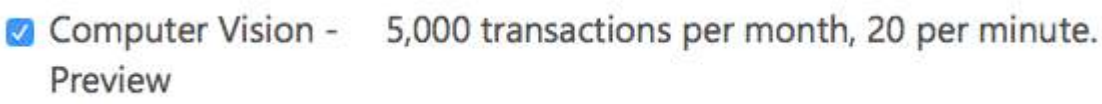
Connecting with Microsoft Cognitive Services

In this example you will learn how to use Microsoft Cognitive Services with Xamarin iOS mobile application. We will use Computer Vision API to detect what is in the picture.

Once you create Xamarin.iOS project please add below NuGet package to the project:

https://www.nuget.org/packages/Microsoft.ProjectOxford.Vision/

With this library we will be able to utilize Cognitive Services in our iOS app. I assume that you have Microsoft account already registered to use it and you have enabled Computer Vision Api like on the screen below:



Once you click "Subscribe" at the bottom Api Key will be generated:

Computer Vision - Preview	5,000 transactions per month, 20 per minute.	Key 1: XXXXXXXXXXXXXXXXXXXXXXXXXXXX Regenerate | Show | Copy Key 2: XXXXXXXXXXXXXXXXXXXXXXXXXXXX Regenerate | Show | Copy

Now we can start configuring access to Cognitive Services from the iOS app. Firstly we need to get some picture for the analysis. To do it we can use Xamarin Media Component available below: https://components.xamarin.com/view/mediaplugin

Once it is successfully installed let's create simple UI with the image and button to select picture from the gallery. Size of the controls is up to you.

Open Main.storyboard and add UIImageView and UIButton controls do default ViewController. Add them names: "SelectedPictureImageView" and "SelectButton":

Now we should add "Touch Up Inside" event handler to handle image selection:

```
partial void SelectButtonClick(UIButton sender)
{
    selectImage();
}

async void selectImage()
{
    var selectedImage = await CrossMedia.Current.PickPhotoAsync();
    SelectedPictureImageView.Image =  new
UIImage(NSData.FromStream(selectedImage.GetStream()));
}
```

Now we would like to display analysis information once Cognitive Services returns the information. Add label below the button called "AnalysisLabel":

SELECT PICTURE

Analysis result....

It is time to connect Computer Vision API!

To get information about selected picture add below method. Remember to paste you API Key!

```
async Task analyseImage(Stream imageStream)
{
    try
    {
        VisionServiceClient visionClient = new VisionServiceClient("<<YOUR API KEY HERE>>");
        VisualFeature[] features = { VisualFeature.Tags, VisualFeature.Categories,
VisualFeature.Description };
        var analysisResult = await visionClient.AnalyzeImageAsync(imageStream,
features.ToList(), null);
        AnalysisLabel.Text = string.Empty;
        analysisResult.Description.Tags.ToList().ForEach(tag => AnalysisLabel.Text =
AnalysisLabel.Text + tag + "\n");
    }
    catch (Microsoft.ProjectOxford.Vision.ClientException ex)
    {
        AnalysisLabel.Text = ex.Error.Message;
    }
}
```

Now you can invoke it in "selectImage" method:

```
async void selectImage()
```

```
{
    var selectedImage = await CrossMedia.Current.PickPhotoAsync();
    SelectedPictureImageView.Image =  new
UIImage(NSData.FromStream(selectedImage.GetStream()));
    await analyseImage(selectedImage.GetStream());
}
```

Once you select image, Microsoft Cognitive Services will analyze it and return the result:

Remember that you image cannot be too large - in this case you will receive information like below:

SELECT PICTURE

Input image is too large.

There are many other services which you can try to use. Please refer to the official documentation (link attached) to discover more.

Read Connecting with Microsoft Cognitive Services online: https://riptutorial.com/xamarin-ios/topic/6122/connecting-with-microsoft-cognitive-services

Chapter 13: Controlling the Screenshot in the iOS Multitasking Switcher

Introduction

In the App Programming Guide for iOS:

Remove sensitive information from views before moving to the background.

When an app transitions to the background, the system takes a snapshot of the app's main window, which it then presents briefly when transitioning your app back to the foreground.

Remarks

Adapted from actual StackOverflow Question Controlling the Screenshot in the iOS7 Multitasking Switcher and answer Obj-c Answer

Examples

Show an Image for Snapshot

```
public override void DidEnterBackground(UIApplication application)
 {
    //to add the background image in place of 'active' image
    var backgroundImage = new UIImageView();
    backgroundImage.Tag = 1234;
    backgroundImage.Image = UIImage.FromBundle("Background");
    backgroundImage.Frame = this.window.Frame;
    this.window.AddSubview(backgroundImage);
    this.window.BringSubviewToFront(backgroundImage);
}

public override void WillEnterForeground(UIApplication application)
{
    //remove 'background' image
    var backgroundView = this.window.ViewWithTag(1234);
    if(null != backgroundView)
        backgroundView.RemoveFromSuperview();
}
```

Read Controlling the Screenshot in the iOS Multitasking Switcher online: https://riptutorial.com/xamarin-ios/topic/8681/controlling-the-screenshot-in-the-ios-multitasking-switcher

Chapter 14: Create and use custom prototype table cells in xamarin.iOS using storyboard

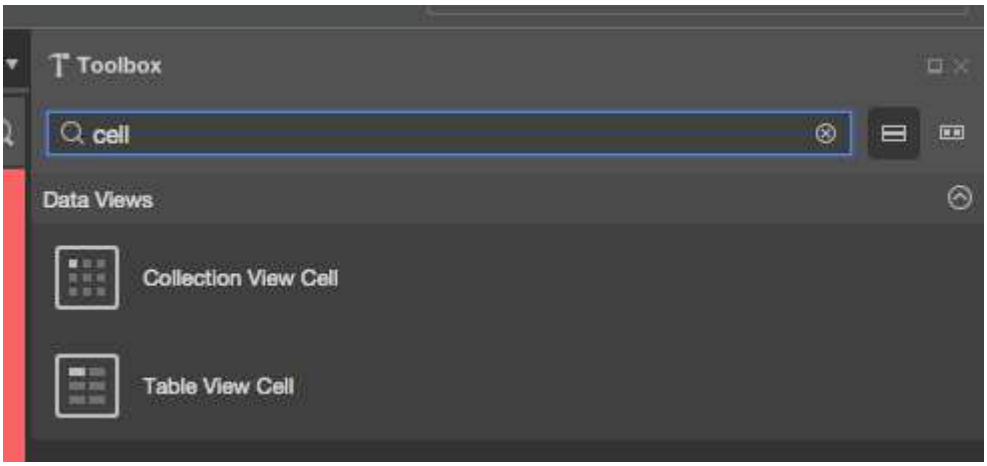
Examples

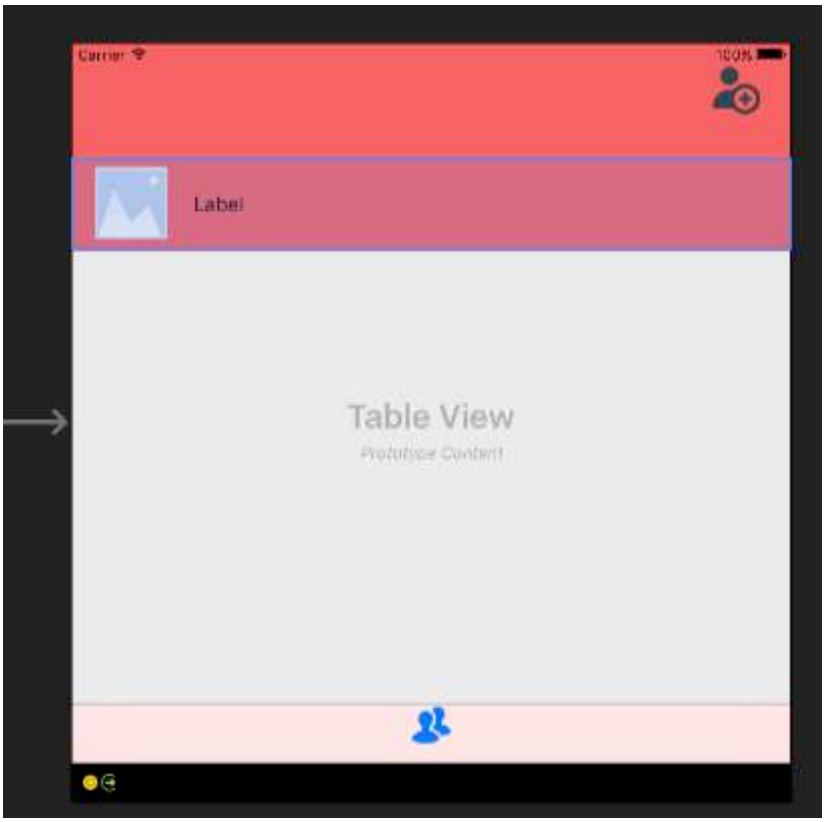
Create custom cell using Storyboard

Open Storyboard where you have your ViewController with TableView:

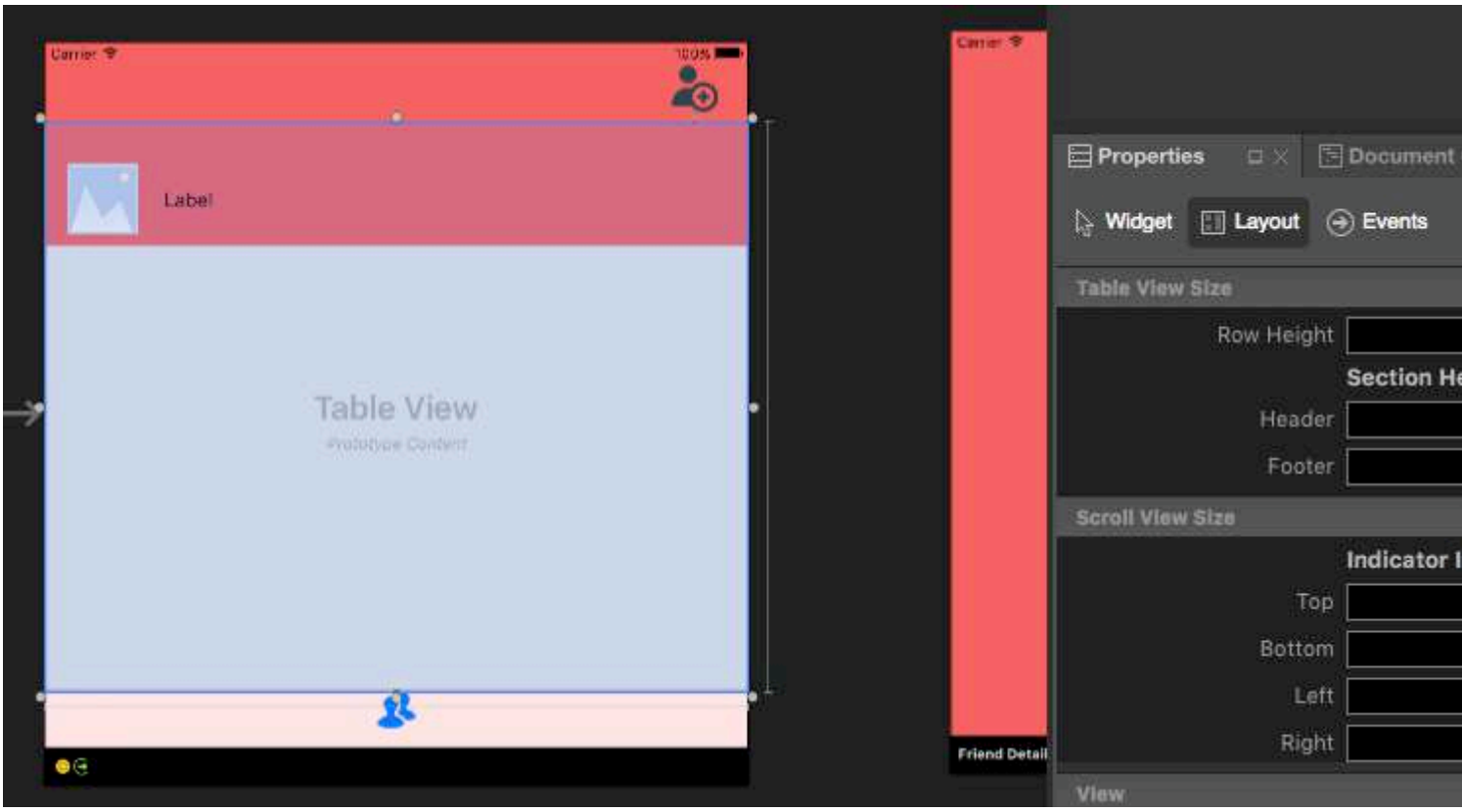
Add prototype cell (if there is no cell added before):

Customize cell as you want (in my case there is custom UIImage and Label):

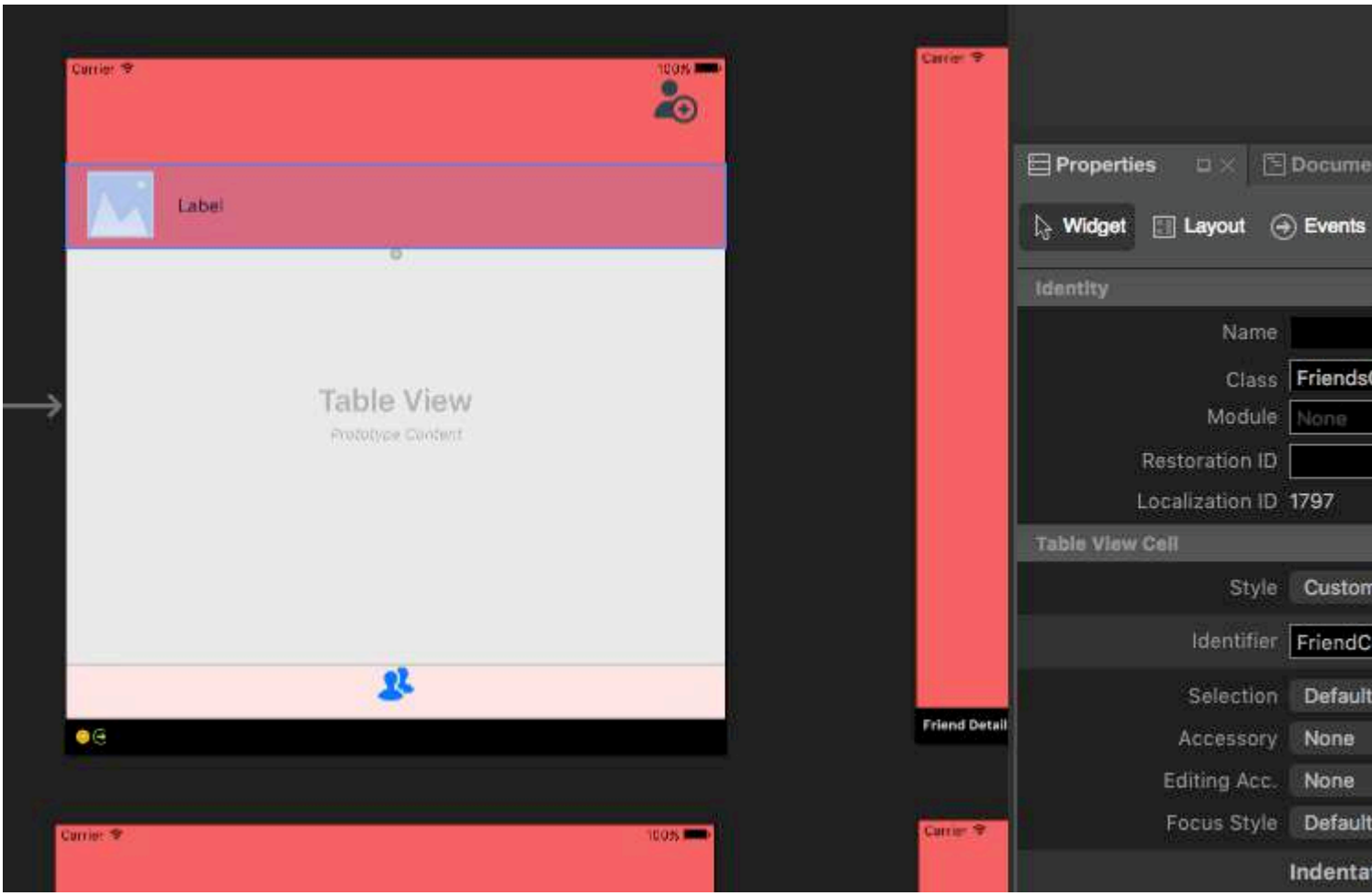




Remember to set height of the cell. To do it select your whole TableView and from the Properties window select "Layout" tab. On the top of the properties window you should see "row height" - put the appropriate value:



Now select prototype cell once again. In the Properties window type the name of the class (it will create code-behind class for it). In my case this is "FriendsCustomTableViewCell". After that provide "Identifier" for your cell. As you can see my is "FriendCell". Last thing to set is "Style" property set to custom. "Name" field should be empty. Once you click "enter" after typing "Class" code-behind file will be automatically created:



Now code behind for the cell should look like below:

```
public partial class FriendsCustomTableViewCell : UITableViewCell
{
    public FriendsCustomTableViewCell (IntPtr handle) : base (handle)
    {
    }

    public FriendsCustomTableViewCell(NSString cellId, string friendName, UIImage friendPhoto)
: base (UITableViewCellStyle.Default, cellId)
    {
        FriendNameLabel.Text = friendName;
        FriendPhotoImageView.Image = friendPhoto;
    }

    //This methods is to update cell data when reuse:
    public void UpdateCellData(string friendName, UIImage friendPhoto)
    {
        FriendNameLabel.Text = friendName;
        FriendPhotoImageView.Image = friendPhoto;
    }
}
```

In UITableViewSource you have to declare cellIdentifier at the top of the class (in my case it is "FriendCell") and in "GetCell" method you have to cast cells and set data for them:

```
string cellIdentifier = "FriendCell";

public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
{
    FriendsCustomTableViewCell cell = (FriendsCustomTableViewCell)
tableView.DequeueReusableCell(cellIdentifier);
    Friend friend = _friends[indexPath.Row];

    //---- if there are no cells to reuse, create a new one
    if (cell == null)
    { cell = new FriendsCustomTableViewCell(new NSString(cellIdentifier), friend.FriendName,
new UIImage(NSData.FromArray(friend.FriendPhoto))); }

    cell.UpdateCellData(friend.UserName, new UIImage(NSData.FromArray(friend.FriendPhoto)));

    return cell;
}
```

Read Create and use custom prototype table cells in xamarin.iOS using storyboard online: https://riptutorial.com/xamarin-ios/topic/5907/create-and-use-custom-prototype-table-cells-in-xamarin-ios-using-storyboard

Chapter 15: How to use asset Asset Catalogs

Examples

Using Asset Catalogs

To use an Asset Catalog, you need to do the following:

1. Double-click the Info.plist file in the Solution Explorer to open it for editing.
2. Scroll down to the App Icons section.
3. From the Source dropdown list, ensure AppIcons is selected.
4. From the Solution Explorer, double-click the Assets.xcassets file to open it for editing.
5. Select AppIcons from the list of assets to display the Icon Editor.
6. Either click on given icon type and select an image file for the required type/size or drag in an image from a folder and drop it on the desired size.
7. Click the Open button to include the image in the project and set it in the xcasset.

Read How to use asset Asset Catalogs online: https://riptutorial.com/xamarin-ios/topic/6539/how-to-use-asset-asset-catalogs

Chapter 16: Resizing Methods for UIImage

Examples

Resize Image - with Aspect Ratio

```
// resize the image to be contained within a maximum width and height, keeping aspect ratio
public static UIImage MaxResizeImage(this UIImage sourceImage, float maxWidth, float
maxHeight)
{
    var sourceSize = sourceImage.Size;
    var maxResizeFactor = Math.Min(maxWidth / sourceSize.Width, maxHeight /
sourceSize.Height);
    if (maxResizeFactor > 1) return sourceImage;
    var width = maxResizeFactor * sourceSize.Width;
    var height = maxResizeFactor * sourceSize.Height;
    UIGraphics.BeginImageContext(new CGSize(width, height));
    sourceImage.Draw(new CGRect(0, 0, width, height));
    var resultImage = UIGraphics.GetImageFromCurrentImageContext();
    UIGraphics.EndImageContext();
    return resultImage;
}
```

Resize Image - without Aspect Ratio

```
// resize the image (without trying to maintain aspect ratio)
public static UIImage ResizeImage(this UIImage sourceImage, float width, float height)
{
    UIGraphics.BeginImageContext(new SizeF(width, height));
    sourceImage.Draw(new RectangleF(0, 0, width, height));
    var resultImage = UIGraphics.GetImageFromCurrentImageContext();
    UIGraphics.EndImageContext();
    return resultImage;
}
```

Crop Image without Resize

```
// crop the image, without resizing
public static UIImage CropImage(this UIImage sourceImage, int crop_x, int crop_y, int width,
int height)
{
    var imgSize = sourceImage.Size;
    UIGraphics.BeginImageContext(new SizeF(width, height));
    var context = UIGraphics.GetCurrentContext();
    var clippedRect = new RectangleF(0, 0, width, height);
    context.ClipToRect(clippedRect);
    var drawRect = new CGRect(-crop_x, -crop_y, imgSize.Width, imgSize.Height);
    sourceImage.Draw(drawRect);
    var modifiedImage = UIGraphics.GetImageFromCurrentImageContext();
    UIGraphics.EndImageContext();
    return modifiedImage;
}
```

Read Resizing Methods for UIImage online: https://riptutorial.com/xamarin-ios/topic/6542/resizing-methods-for-uiimage

Chapter 17: Touch ID

Parameters

Column	Column
Cell	Cell

Remarks

First, establish if the device is capable of accepting Touch ID input.

```
if (context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out
AuthError))
```

If it does then we can display the Touch ID UI by using:

```
context.EvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, myReason,
replyHandler);
```

There are three pieces of information we have to pass into `EvaluatePolicy` – the policy itself, a string explaining why authentication is necessary, and a reply handler. The reply handler tells the application what it should do in the case of a successful, or unsuccessful, authentication.

One of the caveats of Local Authentication is that it must be run on the foreground, so make sure to use `InvokeOnMainThread` for the reply handler:

```
var replyHandler = new LAContextReplyHandler((success, error) =>
    {

        this.InvokeOnMainThread(() =>
        {
            if (success)
            {
                Console.WriteLine("You logged in!");
                PerformSegue("AuthenticationSegue", this);
            }
            else {
                //Show fallback mechanism here
            }
        });

    });
```

To determine whether the database of authorized fingerprints has been modified you can check the opaque structure (NSData) returned by `context.EvaluatedPolicyDomainState`. Your app will need to store and compare the policy state to detect changes. One thing to note which Apple states:

However, the nature of the change cannot be determined from this data.

```
if (context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out
AuthError))
{
    var policyState = context.EvaluatedPolicyDomainState;

    var replyHandler = new LAContextReplyHandler((success, error) =>
    {

        this.InvokeOnMainThread(() =>
        {
            if (success)
            {
                Console.WriteLine("You logged in!");
                PerformSegue("AuthenticationSegue", this);
            }
            else {
                //Show fallback mechanism here
            }
        });

    });
    context.EvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, myReason,
replyHandler);
};
```

Examples

Add Touch ID to your App

First, establish if the device is capable of accepting Touch ID input.

```
if (context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out
AuthError))
```

If it does then we can display the Touch ID UI by using:

```
context.EvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, myReason,
replyHandler);
```

There are three pieces of information we have to pass into `EvaluatePolicy` – the policy itself, a string explaining why authentication is necessary, and a reply handler. The reply handler tells the application what it should do in the case of a successful, or unsuccessful, authentication.

One of the caveats of Local Authentication is that it must be run on the foreground, so make sure to use `InvokeOnMainThread` for the reply handler:

```
var replyHandler = new LAContextReplyHandler((success, error) =>
{
    this.InvokeOnMainThread(() =>
    {
        if (success)
```

```
        {
            Console.WriteLine("You logged in!");
            PerformSegue("AuthenticationSegue", this);
        }
        else {
            //Show fallback mechanism here
        }
    });
});
```

To determine whether the database of authorized fingerprints has been modified you can check the opaque structure (NSData) returned by `context.EvaluatedPolicyDomainState`. Your app will need to store and compare the policy state to detect changes. One thing to note which Apple states:

> However, the nature of the change cannot be determined from this data.

```
if (context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out
AuthError))
{
    var policyState = context.EvaluatedPolicyDomainState;

    var replyHandler = new LAContextReplyHandler((success, error) =>
    {

        this.InvokeOnMainThread(() =>
        {
            if (success)
            {
                Console.WriteLine("You logged in!");
                PerformSegue("AuthenticationSegue", this);
            }
            else {
                //Show fallback mechanism here
            }
        });

    });
    context.EvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, myReason,
replyHandler);
};
```

Button Example

```
partial void AuthenticateMe(UIButton sender)
{
    var context = new LAContext();
    //Describes an authentication context
    //that allows apps to request user authentication using Touch ID.
    NSError AuthError;
    //create the reference for error should it occur during the authentication.
    var myReason = new NSString("To add a new chore");
    //this is the string displayed at the window for touch id

    if (context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out
AuthError))
    // check if the device have touchId capabilities.
    {
        var replyHandler = new LAContextReplyHandler((success, error) =>
```

```
        {

            this.InvokeOnMainThread(() =>
            {
                if (success)
                {
                    Console.WriteLine("You logged in!");
                    PerformSegue("AuthenticationSegue", this);
                }
                else {
                    //Show fallback mechanism here
                }
            });

        });
        context.EvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, myReason,
replyHandler);//send touch id request
    };
}
```

Using Keychain

Working Source - https://github.com/benhysell/V.TouchIdExample

Long form description - http://benjaminhysell.com/archive/2014/11/authentication-in-xamarin-ios-with-touch-id-or-passcode/

```
//Simple View with a switch to enable / disable Touch ID and
//a button to invoke authentication

/// <summary>
/// Enable/Disable Touch ID
/// </summary>
/// <param name="sender">Sender.</param>
partial void TouchIdEnableDisable(UISwitch sender)
{
    if (sender.On)
    {
        //enable Touch ID
        //set our record
        //note what you fill in here doesn't matter, just needs to be
        //consistent across all uses of the record
        var secRecord = new SecRecord(SecKind.GenericPassword)
        {
            Label = "Keychain Item",
            Description = "fake item for keychain access",
            Account = "Account",
            Service = "com.yourcompany.touchIdExample",
            Comment = "Your comment here",
            ValueData = NSData.FromString("my-secret-password"),
            Generic = NSData.FromString("foo")
        };

        secRecord.AccessControl = new
SecAccessControl(SecAccessible.WhenPasscodeSetThisDeviceOnly,
SecAccessControlCreateFlags.UserPresence);
        SecKeyChain.Add(secRecord);
```

```
            authenticateButton.Enabled = true;
        }
        else
        {
            //disable Touch ID
            var record = new SecRecord(SecKind.GenericPassword)
            {
                Service = "com.yourcompany.touchIdExample",
                UseOperationPrompt = "Authenticate to Remove Touch ID / Passcode from Test App"
            };

            SecStatusCode result;

            //query one last time to ensure they can remove it
            SecKeyChain.QueryAsRecord(record, out result);
            if (SecStatusCode.Success == result || SecStatusCode.ItemNotFound == result)
            {
                //remove the record
                SecKeyChain.Remove(record);
                authenticateButton.Enabled = false;
            }
            else
            {
                //could not authenticate, leave switch on
                sender.On = true;
             }
        }
    }

    /// <summary>
    /// Show Touch ID to user and evaluate authentication
    /// </summary>
    /// <param name="sender">Sender.</param>
    partial void AuthenticateUser(UIButton sender)
    {
        var rec = new SecRecord(SecKind.GenericPassword)
        {
            Service = "com.yourcompany.touchIdExample",
            UseOperationPrompt = "Authenticate to access Test App"
        };
        SecStatusCode res;
        SecKeyChain.QueryAsRecord(rec, out res);
        if (SecStatusCode.Success == res || SecStatusCode.ItemNotFound == res)
        {
            //Success!!
            //add your code here to continue into your application
            AuthenticatedLabel.Hidden = false;
        }
        else
        {
            //Failure
            AuthenticatedLabel.Hidden = true;
        }
    }
```

Read Touch ID online: https://riptutorial.com/xamarin-ios/topic/577/touch-id

Chapter 18: UIImageView zoom in combination with UIScrollView

Remarks

The UIImageView has to be within a scrollview in order for this to work.

The DoubleTap method will toggle between the minScale and the doubleTapScale.

Examples

Double Tap

```
private float minScale = 1f;
private float doubleTapScale = 2f;
private float maxScale = 4f;

private void SetUpDoubleTapZoom()
{
    imageViewToZoom.ContentMode = UIViewContentMode.ScaleAspectFit;
    scrollView.MaximumZoomScale = maxScale;
    scrollView.MinimumZoomScale = minScale;

    var doubleTap = new UITapGestureRecognizer(OnDoubleTap)
    {
        NumberOfTapsRequired = 2
    };

    scrollView.AddGestureRecognizer(doubleTap);
}

private void OnDoubleTap(UIGestureRecognizer gesture)
{
    scrollView.ZoomScale = (scrollView.ZoomScale.Equals(minScale)) ? doubleTapScale :
minScale;
}
```

Pinch gesture zoom

```
private float minScale = 1f;
private float maxScale = 4f;

private void SetUpPinchGestureZoom()
{
    imageViewToZoom.ContentMode = UIViewContentMode.ScaleAspectFit;

    scrollView.MaximumZoomScale = maxScale;
    scrollView.MinimumZoomScale = minScale;

    scrollView.ViewForZoomingInScrollView += (UIScrollView sv) => { return imageViewToZoom; };
}
```

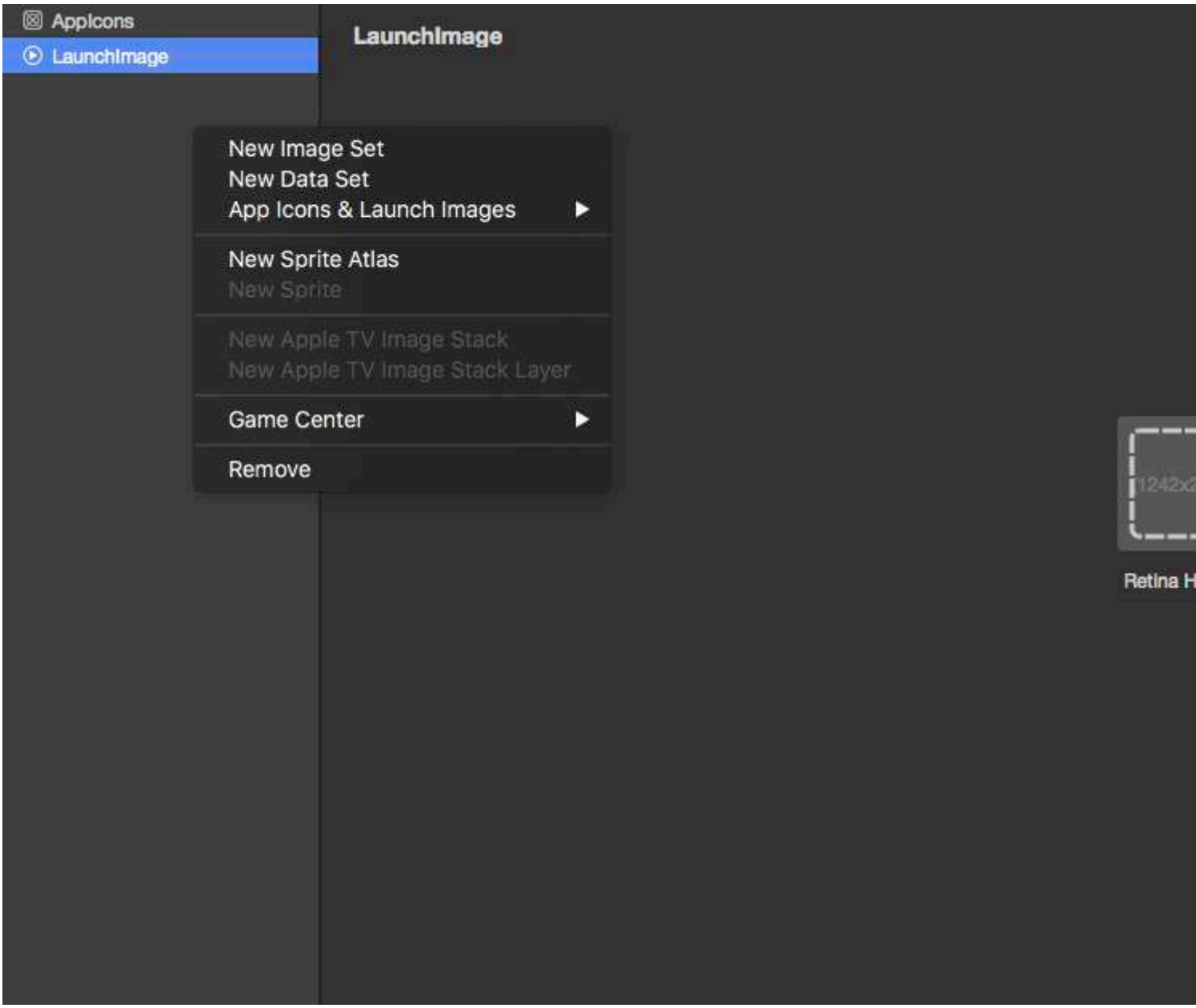
Read UIImageView zoom in combination with UIScrollView online: https://riptutorial.com/xamarin-ios/topic/6346/uiimageview-zoom-in-combination-with-uiscrollview

Chapter 19: Using Asset Catalogs

Examples

Adding image assets to asset catalog

This is how the Asset Catalog in Xamarin Studio looks like,



As shown in above picture there are 5 types of assets you can create within the catalog.

I will cover only image set, because its the simplest one.

When you create a new image set. You will get options like this

To add images to the catalog you can simply click on the dashed squares and select the image

you want to set for particular option.

In XCode you have options of 1x, 2x and 3x to cover the most recent iOS device screen sizes. But Xamarin has one extra option Vector using which you can upload PDF formatted Vector image which would be automatically scaled depending on the device your application is running on.

For iPhone images Xamarin retains the iOS7 special image size R4 which is used for 4-Inch screen sized iPhone (5, 5S and SE).

Please refer Xamarin Documentation on how to add images to iOS application for more information.

Read Using Asset Catalogs online: https://riptutorial.com/xamarin-ios/topic/6630/using-asset-catalogs

Chapter 20: Using iOS Asset Catalogs to Manage Images

Remarks

Asset catalogs are way to manage multiple resolutions of iOS image assets. In order to display optimal images, iOS uses 1x, 2x, and 3x versions of each image according to the device's screen density. The 1x version is only for very old, non-retina devices so it isn't necessary for apps only supporting iOS 9.

Asset catalogs will help support app thinning and slicing, optimizing the resources users have to download to install an app from the App Store.

Examples

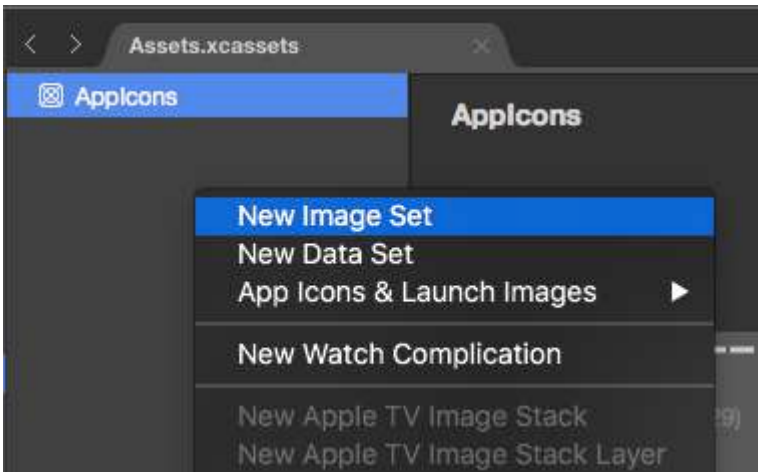
Loading an asset catalog image

Load an image from an asset catalog using `UIImage.FromBundle(string imageName)`

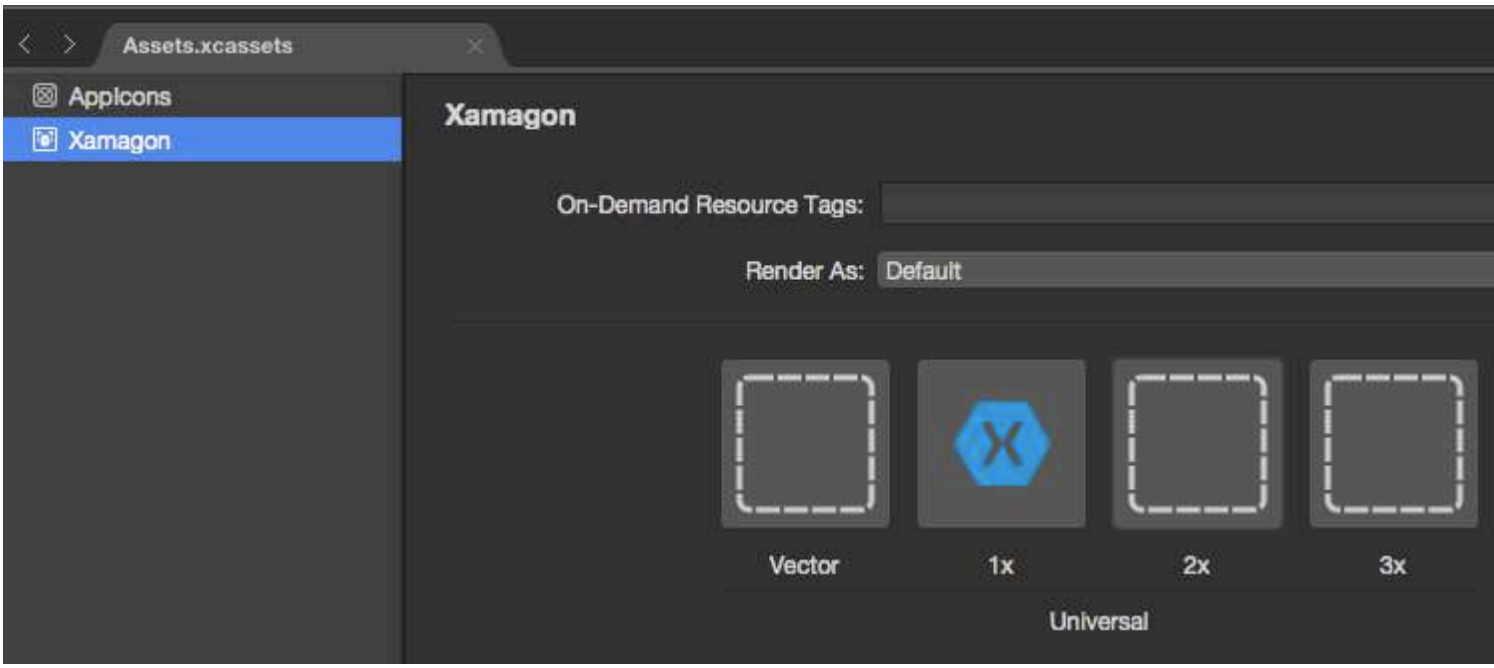
```
UIImage image = UIImage.FromBundle("ImageName");
// use the name of the image set from the asset catalog
```

You can use the image for a `UIImageView` or anything else you need to do.

Managing Images in an asset catalog



Asset catalogs allow managing images, app icons, and launch images. Image Set is used for images which are displayed in the app. Universal images are usually the best option. You can either use a vector based image (such as PDF) which will scale for all screens, or include a 1x, 2x, and 3x variant and iOS will select the appropriate version of the image for the user's current device.



You can change the name of any set in the asset catalog by double-clicking on name. Images can be added by either drag and drop or click on the image you want to fill-in for a file picker.

Adding Asset Catalog images in storyboard

Asset catalog images can be used from storyboards like any other kind of image added to the project. They will be automatically populated as an option in `UIImageView` and other views which support adding an image.

Read Using iOS Asset Catalogs to Manage Images online: https://riptutorial.com/xamarin-ios/topic/6241/using-ios-asset-catalogs-to-manage-images

Chapter 21: Working with Xib and Storyboards in Xamarin.iOS

Examples

Opening Xib/Sotryboard in Xcode Interface Builder instead

Xamarin sturio opens Xib file and Storyboards by default in the Xamarin Designer. User can right click on the file and `Open With` -> `Xcode Interface Builder'

Read Working with Xib and Storyboards in Xamarin.iOS online: https://riptutorial.com/xamarin-ios/topic/6182/working-with-xib-and-storyboards-in-xamarin-ios

Chapter 22: Xamarin iOS Google Places Autocomplete

Introduction

Since starting to work with Xamarin there have been many things I wish someone else had already documented. Here I explain how to use 1 aspect of the google places autocompletion, the UI control making use of a results controller. While this code is based on googles own examples converted to C# from Swift I have tried my best to make sure it is a fully working example. I hope very much this documentation will help others.

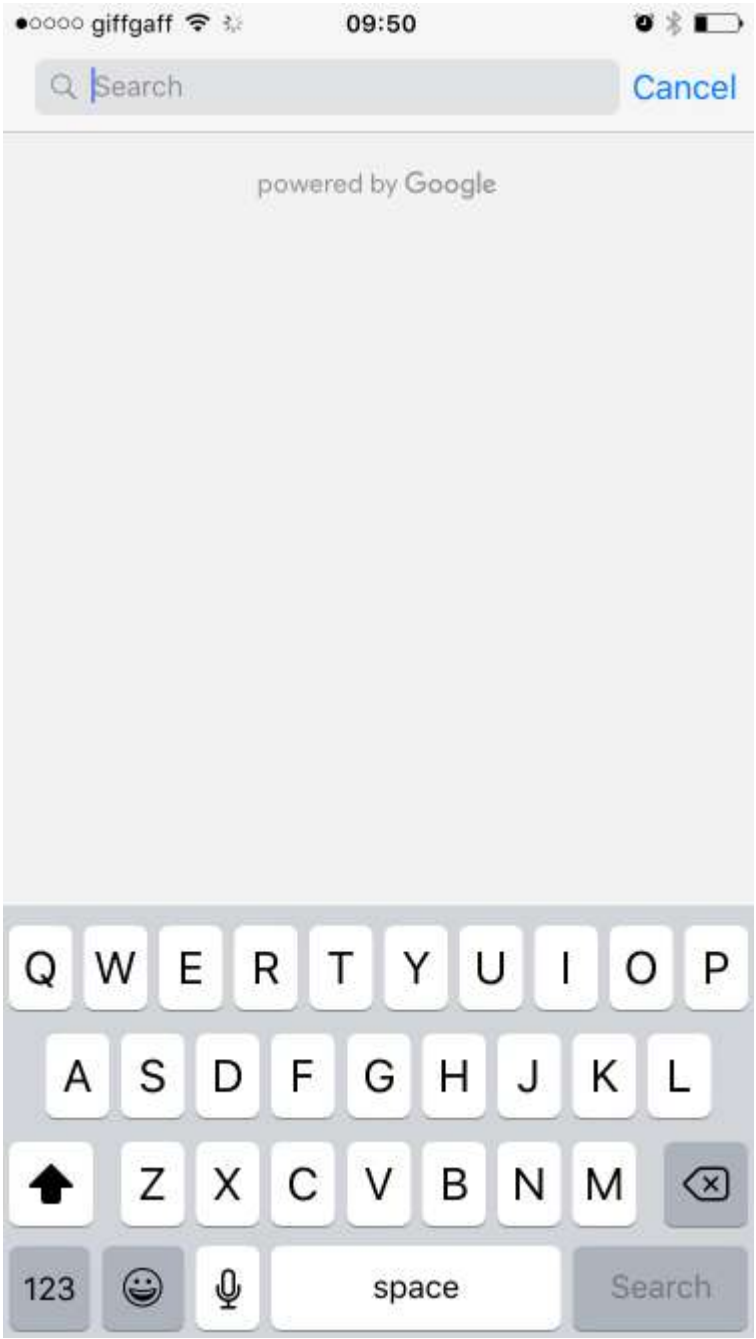
Examples

Add an autocomplete UI control with results controller.

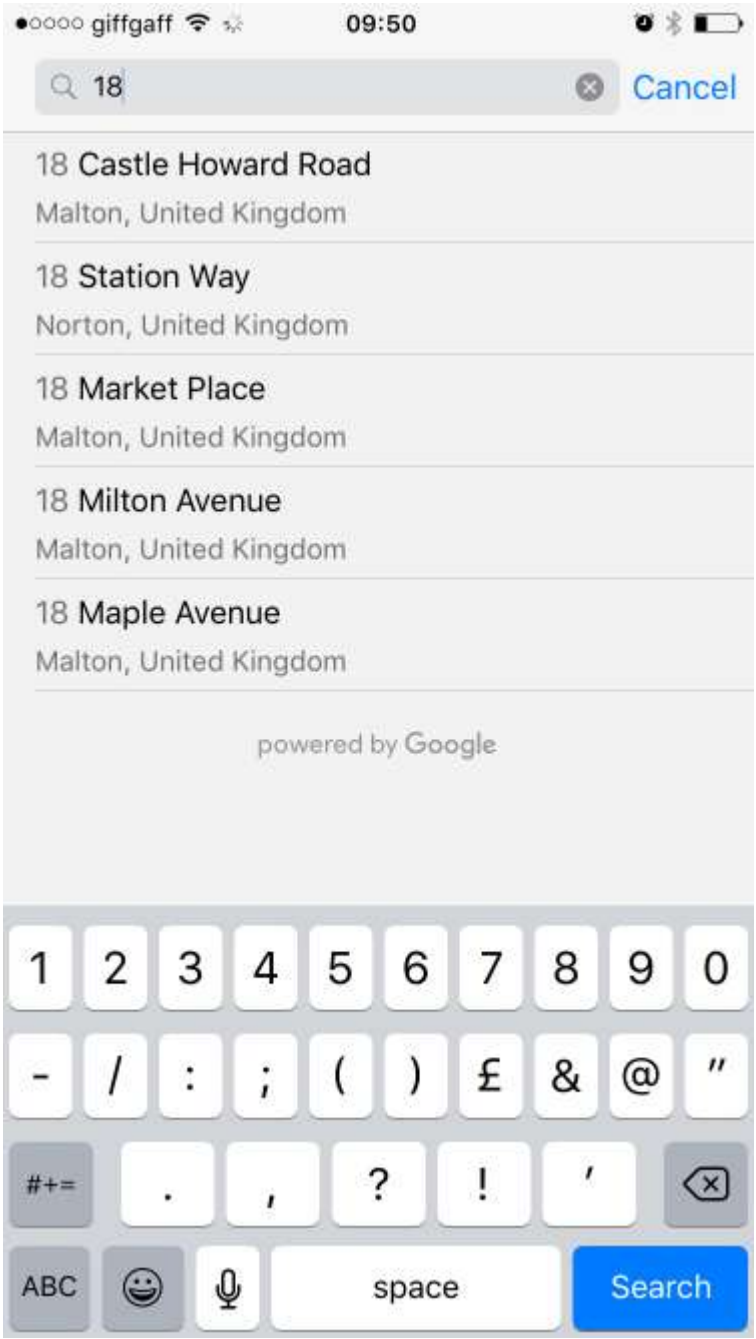
The autocomplete UI control is a search dialog with built-in autocomplete functionality. As a user enters search terms, the control presents a list of predicted places to choose from. When the user makes a selection, a GMSPlace (Place in Xamarin) instance is returned, which your app can then use to get details about the selected place.

As mentioned above this example uses a results controller which allows for more control over the text input UI. The results controller will dynamically toggle the visibility of the results list based on input UI focus.

The aim of this code is to display a screen just like the below:



This will autocomplete your address when you start typing, like the image below:



Instructions:

1. First we need to add the google maps API to our Visual Studio, It's available through Nuget, just search for Xamarin.Google.iOS.Maps, add it to your iOS project, alternatively you can download it from Xamarin Xamarin Google Maps iOS SDK

2. We need something like a button to trigger the google autocomplete view controller. In this example I do this with a story board and have added a button named GoogleButton, you could trigger it with code it doesn't really matter.

3. Under ViewDidLoad in your view controllers class add the following code. In my example below I am not using the mobile devices actual location, my final solution would of course do this, but this was a test and I didn't want to implement extra code until I had proved this worked or dilute what I am trying to show you:

// Code to bring up the google places auto complete view controller.

```
GoogleButton.TouchUpInside += (sender, ea) =>
{

    var FakeCoordinates = new CLLocationCoordinate2D()
    {
        Latitude = 54.135364,
        Longitude = -0.797888
    };

    var north = LocationWithBearing(45, 3000, FakeCoordinates);
    var east = LocationWithBearing(225, 3000, FakeCoordinates);

    var autocompleteController = new AutocompleteViewController();
    autocompleteController.Delegate = new AutoCompleteDelegate();
    autocompleteController.AutocompleteBounds = new CoordinateBounds(north, east);
    PresentViewController(autocompleteController, true, null);

};
```

4. This is optional, but I have added a function to calculate a local bounds, I am passing in 3000 this number is in metres, so if you want a bigger initial bounds feel free to adjust, please note that google search will still find any address in the world, it just weights the initial results to this local areas bounds first. This function was borrowed from a stack overflow post, I have converted it from Swift to C# for our purposes:

```
public CLLocationCoordinate2D LocationWithBearing(Double bearing, Double distanceMeters,
CLLocationCoordinate2D origin)
{
    var distRadians = distanceMeters/(6372797.6);

    var rbearing = bearing*Math.PI/180.0;

    var lat1 = origin.Latitude*Math.PI/180;
    var lon1 = origin.Longitude*Math.PI/180;

    var lat2 = Math.Asin(Math.Sin(lat1)*Math.Cos(distRadians) +
Math.Cos(lat1)*Math.Sin(distRadians)*Math.Cos(rbearing));
    var lon2 = lon1 + Math.Atan2(Math.Sin(rbearing)*Math.Sin(distRadians)*Math.Cos(lat1),
                    Math.Cos(distRadians) - Math.Sin(lat1)*Math.Sin(lat2));

    return new CLLocationCoordinate2D(latitude: lat2*180/ Math.PI, longitude:
lon2*180/Math.PI);
}
```

5. This final code snippet is the delegate for the autocomplete, we need this delegate to handle all that google will return to us:

```
public  class AutoCompleteDelegate : AutocompleteViewControllerDelegate
{

    public override void DidFailAutocomplete(AutocompleteViewController viewController,
NSError error)
    {
        // TODO: handle the error.
```

```
        Debug.Print("Error: " + error.Description);
    }


    public override void DidAutocomplete(AutocompleteViewController viewController, Place
place)
    {
        Debug.Print(place.Name);
        Debug.Print(place.FormattedAddress);

        viewController.DismissViewController(true, null);
    }

    public override void DidRequestAutocompletePredictions(AutocompleteViewController
viewController)
    {
        UIApplication.SharedApplication.NetworkActivityIndicatorVisible = true;
    }

    public override void DidUpdateAutocompletePredictions(AutocompleteViewController
viewController)
    {
        UIApplication.SharedApplication.NetworkActivityIndicatorVisible = true;
    }

    public override void WasCancelled(AutocompleteViewController viewController)
    {
        viewController.DismissViewController(true, null);
    }

}
```

Run your project and it should work perfectly, this example is quite focussed, but hopefully it till give you a basic example of how any of the google autocomplete UI controls would need to work. Thanks!

Read Xamarin iOS Google Places Autocomplete online: https://riptutorial.com/xamarin-ios/topic/9041/xamarin-ios-google-places-autocomplete

Chapter 23: Xamarin.iOS Navigation Drawer

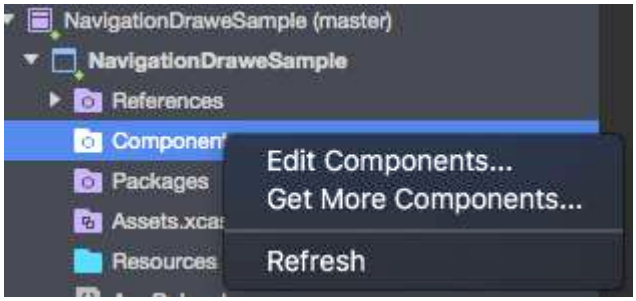
Syntax

1. Flayout Navigation Component: https://components.xamarin.com/view/flyoutnavigation

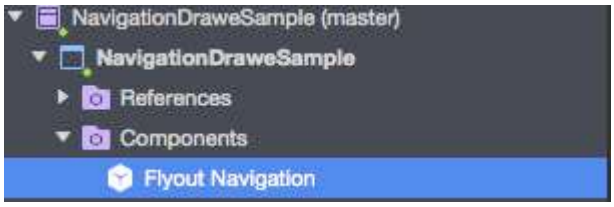
Examples

Xamarin.iOS Navigation Drawer

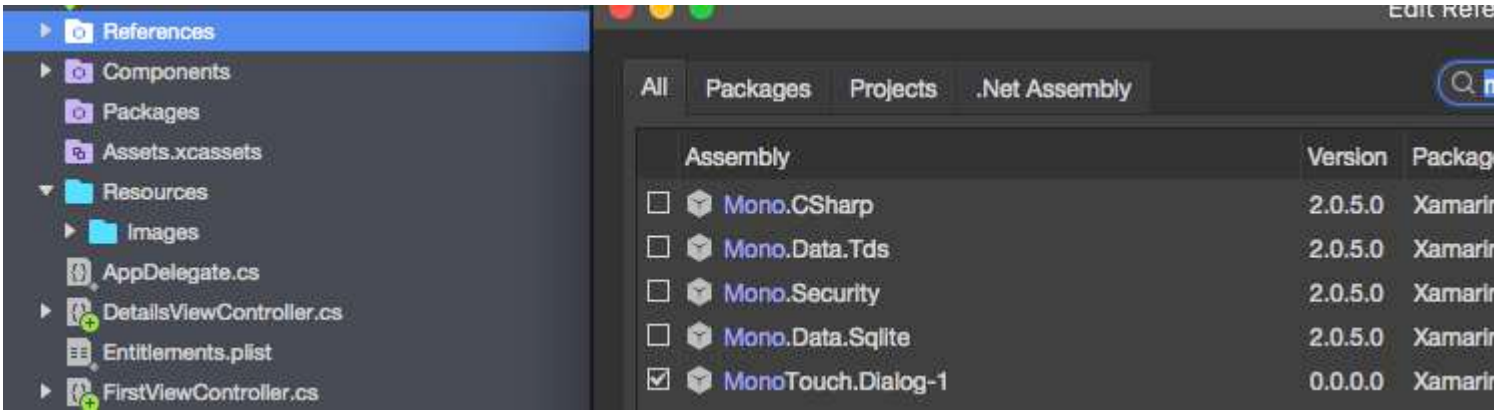
1. Create new Xamarin.iOS blank project (Single View App).
2. Right click on the "Components" folder and select "Get More Components":



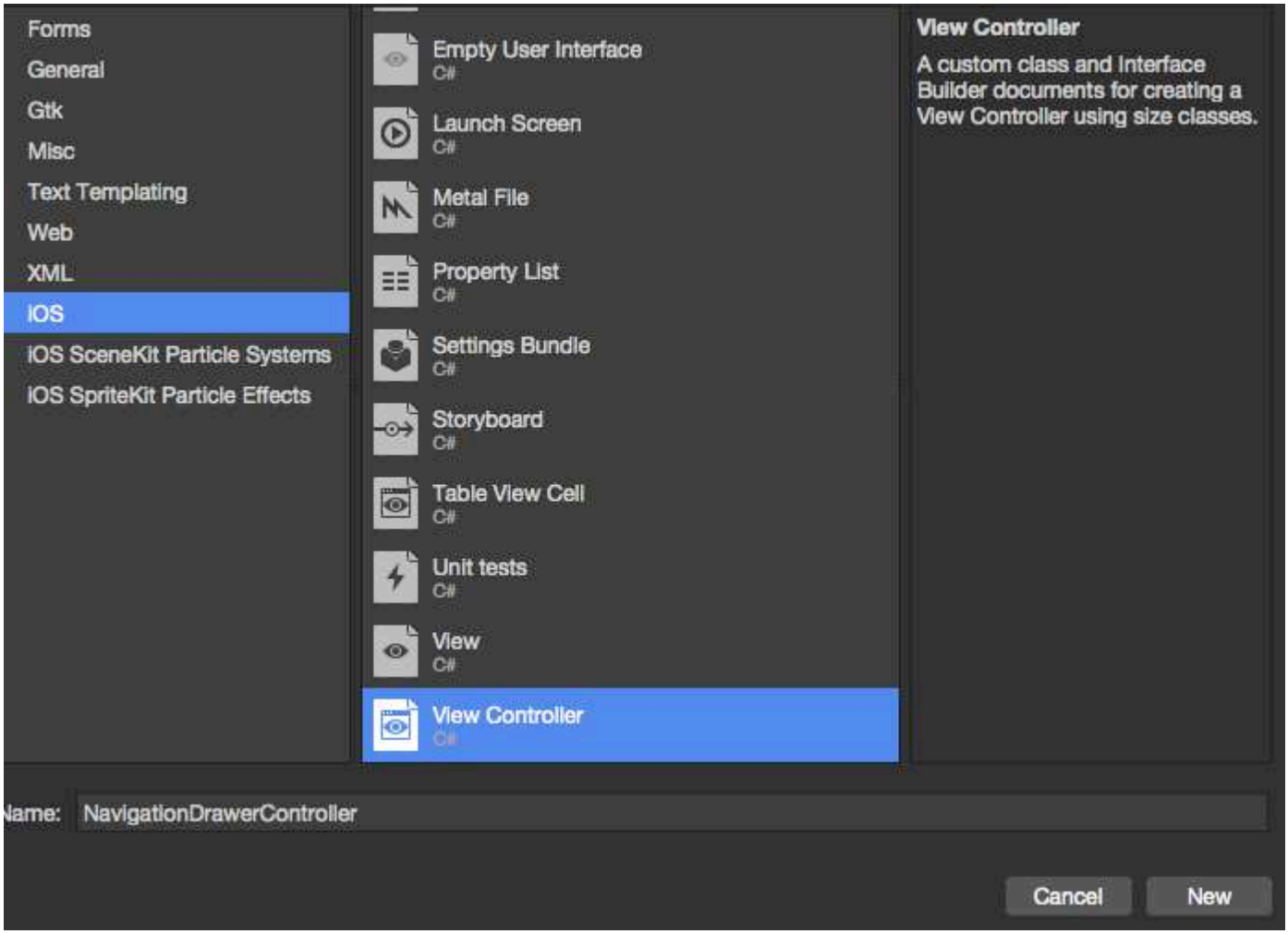
3. In search box type: "Flout Navigation" and add below component to your app:



Remember also to add "Mono.Touch.Dialog-1" reference:



4. Now tight click on the project and add new UIViewController called "NavigationDrawerController":



5. Now code for "NavigationDrawerController" class should look like below:

```
public partial class NavigationDrawerController : UIViewController
{
    public NavigationDrawerController(IntPtr handle) : base(handle)
    {
    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
```

```
        NavigationItem.LeftBarButtonItem = getMenuItem();
        NavigationItem.RightBarButtonItem = new UIBarButtonItem { Width = 40 };
    }

    UIBarButtonItem getMenuItem()
    {
        var item = new UIBarButtonItem();
        item.Width = 40;
        //Please provide your own icon or take mine from the GitHub sample:
        item.Image = UIImage.FromFile("Images/menu_button@2x.png");
        item.Clicked += (sender, e) =>
        {
            if (ParentViewController is MainNavigationController)
                (ParentViewController as MainNavigationController).ToggleMenu();
        };

        return item;
    }
}
```

No worries that "MainNavigationController" is highlighted red - we will add it in the next step.

6. Now open "Main.storyboard" file:

a) Add one UIViewController:

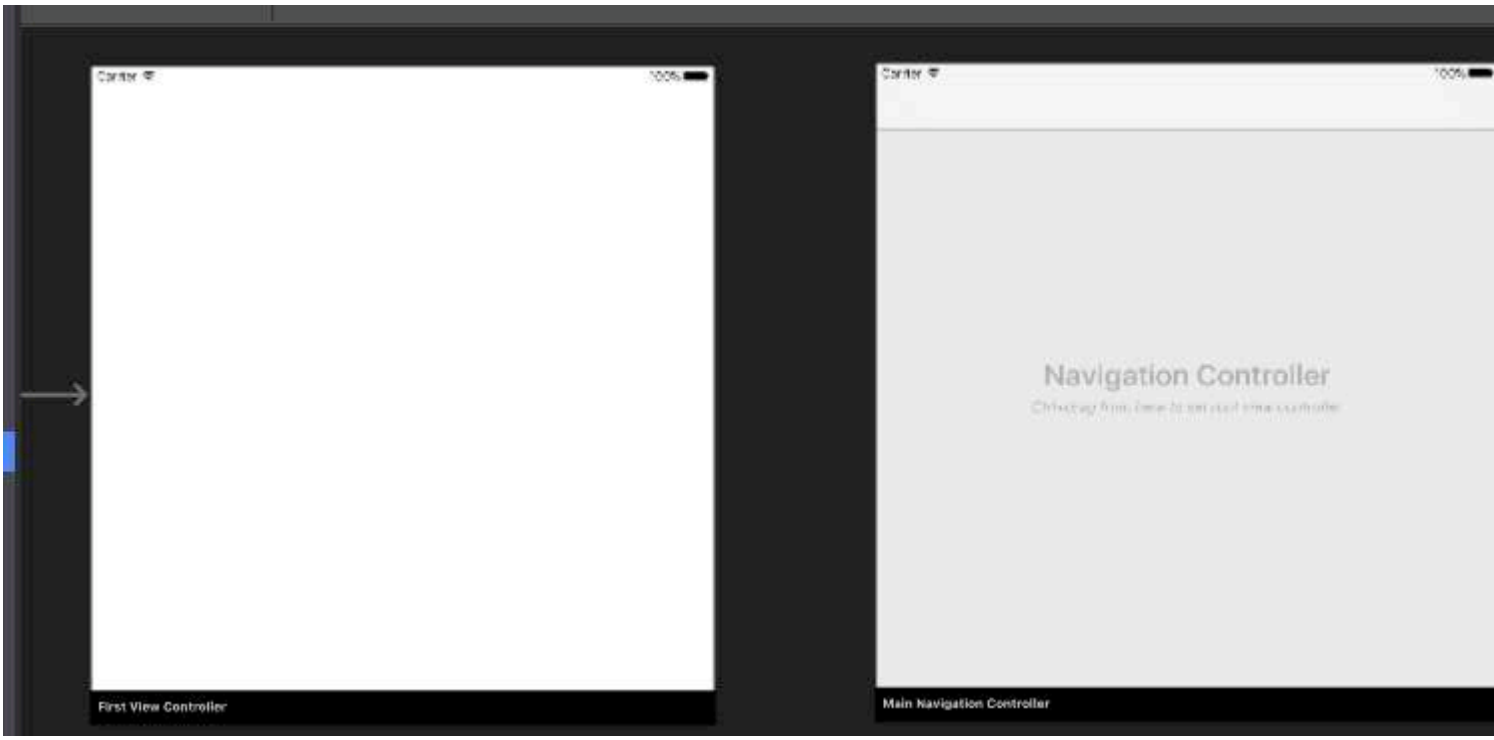
Fill "Class" and "StoryboardID" fields with this name: "FirstViewController"

b) After that add Navigation Controller with root UIViewController:

Fill "Class" and "StoryboardID" fields with this name: "MainNavigationController" for the Navigation Controller

Fill "Class" and "StoryboardID" fields with this name: "DetailsViewController" for the Root Controller

Xamarin (or Visual) Studio will create code-behind classes for above controllers.



7. Now open "FirstViewController" class and paste below code:

```
public partial class FirstViewController : UIViewController
{
    public FirstViewController (IntPtr handle) : base (handle)
    {
    }

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();
        createNavigationFlyout();
    }

    void createNavigationFlyout()
    {
        var navigation = new FlyoutNavigationController
        {
            //Here are sections definied for the drawer:
            NavigationRoot = new RootElement("Navigation")
            {
                new Section ("Pages")
                {
                    new StringElement ("MainPage")
                }
            },

            //Here are controllers definied for the drawer (in this case navigation controller
with one root):
             ViewControllers = new[]
             {

(MainNavigationController)Storyboard.InstantiateViewController("MainNavigationController")
             }
        };

        View.AddSubview(navigation.View);
```

```
    }
}
```

8. Open "MainNavigationController" class and paste below code:

```
public partial class MainNavigationController : UINavigationController
{
     public MainNavigationController (IntPtr handle) : base (handle)
     {
     }
     //Responsible for opening/closing drawer:
     public void ToggleMenu()
     {
         if (ParentViewController is FlyoutNavigationController)
         (ParentViewController as FlyoutNavigationController).ToggleMenu();
     }
 }
```

9. Last class called "DetailsViewController" should look like this:

```
public partial class DetailsViewController : NavigationDrawerController
{
    public DetailsViewController (IntPtr handle) : base(handle)
    {
    }
}
```

Please note that "DetailsViewController" derives from "NavigationDrawerController" which we created on the beginning.

That's it. Now you can customize the drawer however you want. Please also find ready sample on my GitHub:

https://github.com/Daniel-Krzyczkowski/XamarinIOS/tree/master/Xamarin.iOS.NavigationDrawer

Read Xamarin.iOS Navigation Drawer online: https://riptutorial.com/xamarin-ios/topic/6574/xamarin-ios-navigation-drawer

Credits

S. No	Chapters	Contributors
1	Getting started with Xamarin.iOS	Amy Burns, chrisntr, Community, Dominic, hankide, Sergey, valdetero
2	Add PullToRefresh to UITableView	Aditya Kumar, valdetero
3	Add Search Bar to UITableView	Aditya Kumar, valdetero
4	Adding UIRefreshControl to a table view	manishKungwani, valdetero
5	Alerts	chrisntr, Gil Sand, patridge, Pilatus, Prashant C, valdetero
6	Auto Layout in Xamarin.iOS	ben, patridge, Tom Hawkin, valdetero
7	Best practices for migrating from UILocalNotification to User Notifications framework	Aditya Kumar
8	Binding Swift Libraries	Alex Sorokoletov, Elad Nava, Esam Sherif, J. Rahmati, James Mundy, Lucas Teixeira
9	Calculating variable row height in GetHeightForRow	Larry OBrien, valdetero
10	Concurrent Programming in Xamarin.iOS	Ashan, Pilatus, Tom Gilder, valdetero
11	Connecting with Microsoft Cognitive Services	Daniel Krzyczkowski, valdetero
12	Controlling the Screenshot in the iOS Multitasking	ben

	Switcher	
13	Create and use custom prototype table cells in xamarin.iOS using storyboard	Daniel Krzyczkowski, valdetero
14	How to use asset Asset Catalogs	Aditya Kumar
15	Resizing Methods for UIImage	Frauke Nonnenmacher, raymondis, valdetero
16	Touch ID	Amy Burns, ben, DannyC, Matthew, Peter Zhong, valdetero
17	UIImageView zoom in combination with UIScrollView	Citroenfris, valdetero
18	Using Asset Catalogs	aniket.ghode, mnoronha
19	Using iOS Asset Catalogs to Manage Images	dylansturg, valdetero
20	Working with Xib and Storyboards in Xamarin.iOS	lukya
21	Xamarin iOS Google Places Autocomplete	Conrad
22	Xamarin.iOS Navigation Drawer	Daniel Krzyczkowski, valdetero