Getting Started with Mobile Development for WPF Developers

# Overview

Visual Studio and Xamarin enable you to deliver native Android, iOS, and Windows apps, using existing skills, teams, and code. You can use the same IDE, language, and APIs everywhere to build native apps for multiple platforms on a shared C# codebase.

Xamarin.Forms and WPF development are very similar, however note that some class name and concepts can be a little bit different in Xamarin.Forms than WPF. From a WPF standpoint, there are three main concepts that we will be going over in this lab:

* Templates and Styles
* Databinding
* XAML UI

In this lab you will build a lightweight version of a conference app using the model-view-viewmodel (MVVM) pattern. It includes a list of sessions a user can attend and selecting a session will give the user more details on that session. The application includes native projects for iOS, Android, and UWP.

# Objectives

* Extend the sample app to support basic conference features across devices

# Prerequisites

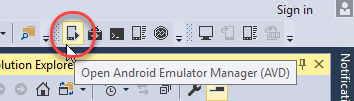
* Windows 10
* Visual Studio 2017

# Intended Audience

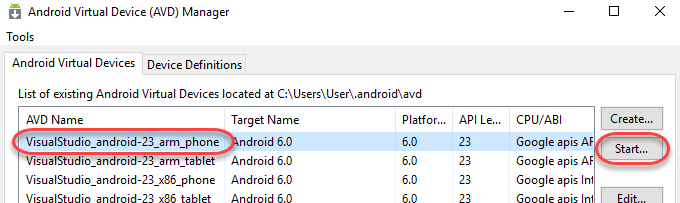
This Quick Start Challenge is intended for developers who are familiar with C# and Visual Studio.

# Task 1: Opening the solution and launching the Android emulator

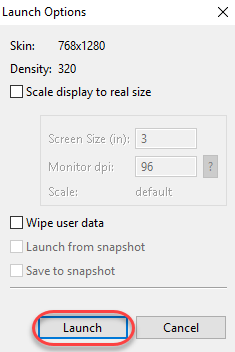
1. Launch a new instance of **Visual Studio**.
2. Select **File | Open | Project/Solution** and select the solution file at **C:\Labs\DEV08 - Getting Started with Mobile Development for WPF Developers\FirstApp\FirstApp.sln**.
3. Close the **Xamarin Mac Agent** dialog if it opens.
4. Click the **Open Android Emulator Manager (AVD)** button from the toolbar. This is an Android SDK utility that can be used to launch the emulator. Since it can take a few minutes for the emulator to be ready, you’ll kick that process off now so that it should be already available to deploy to later on.



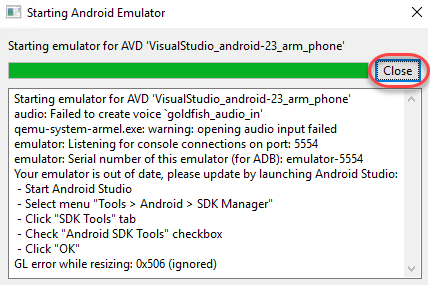
1. Bring **Android Virtual Device (AVD) Manager** to the front (it will open behind Visual Studio).
2. Select the **ARM phone** emulator (it should be the first one) and click **Start**.



1. Click **Launch**.

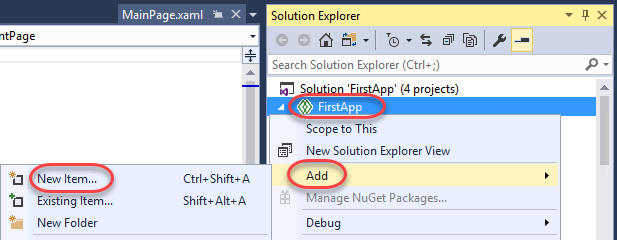


1. It will take a few seconds for the emulator launch to complete. Once the boot process is underway, you’ll see a **Close** button. Click it.

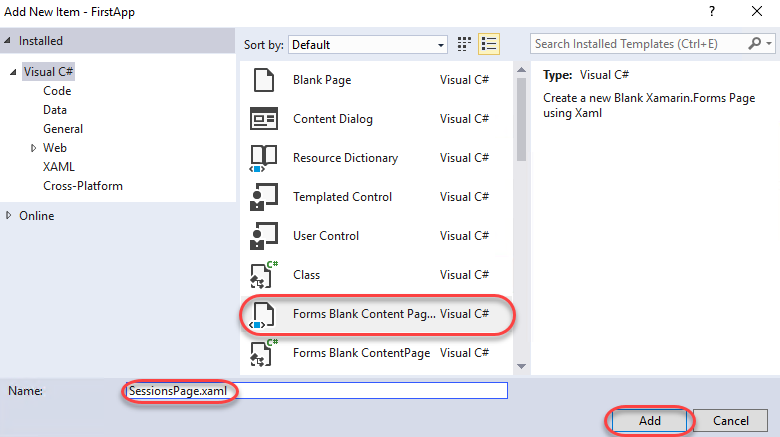


# Task 2: Extending the app

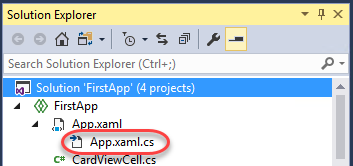
1. Return to Visual Studio. Leave the emulator open to load in the background.
2. The solution structure includes four projects: an Android platform project, an iOS platform project, a UWP platform project, and a Shared Project **(FirstApp)** with all of the shared code. We will be working exclusively with the shared code in this lab. Although the provided solution has already been created, it’s relatively empty at this time. We will now extend it to deliver an application that shows the conference sessions and details.
3. In **Solution Explorer**, right-click the **FirstApp** node and select **Add | New Item**. You will add a few files to this project during this lab, and they’ll all follow this general flow.



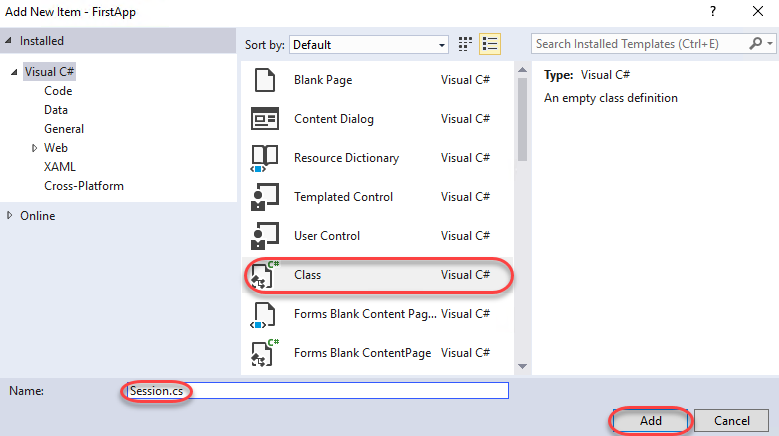
1. Select the **Forms Blank Content Page Xaml** template and set the **Name** to **“SessionsPage.xaml”**. Click **Add**.



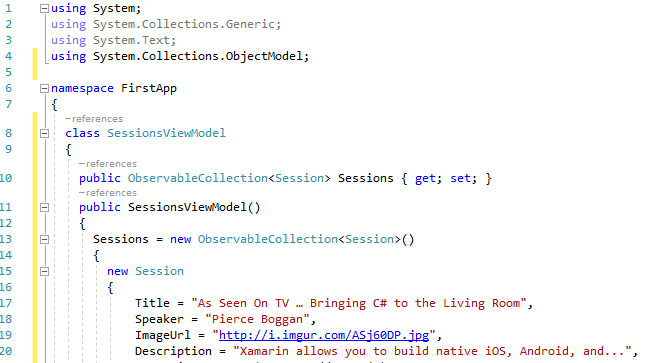
1. **SessionsPage** will ultimately replace the default **MainPage**, so open **App.xaml.cs**.



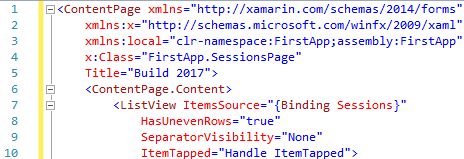
1. Locate the **App()** constructor and replace the line setting **MainPage** with the code below.
   1. MainPage = new NavigationPage(new SessionsPage());
2. Right-click the **FirstApp** node in **Solution Explorer** and select **Add | New Item** like before. This time, add a new **Class** named **Session**. This will serve as the model for each conference session.



1. Replace the default empty class declaration with the following.
   1. public class Session
   2. {
   3. public string Title { get; set; }
   4. public string Speaker { get; set; }
   5. public string ImageUrl { get; set; }
   6. public string Description { get; set; }
   7. public DateTime StartTime { get; set; }
   8. public DateTime EndTime { get; set; }
   9. }
2. Add another class to the shared **FirstApp** project named **SessionsViewModel.cs**. This will serve as the viewmodel for the conference session list and bind each session model to the user interface.
3. Add a **using** directive for **System.Collections.ObjectModel** to the top of the file.
   1. using System.Collections.ObjectModel;
4. Add an **ObservableCollection** property named **Sessions** to the viewmodel using the code below.
   1. public ObservableCollection<Session> Sessions { get; set; }
5. Add the constructor below to initialize the collection with some demo data.
   1. public SessionsViewModel()
   2. {
   3. Sessions = new ObservableCollection<Session>()
   4. {
   5. new Session
   6. {
   7. Title = "As Seen On TV … Bringing C# to the Living Room",
   8. Speaker = "Pierce Boggan",
   9. ImageUrl = "http://i.imgur.com/ASj60DP.jpg",
   10. Description = "Xamarin allows you to build native iOS, Android, and...",
   11. StartTime = DateTime.Now.AddHours(1),
   12. EndTime = DateTime.Now.AddHours(3)
   13. },
   14. new Session
   15. {
   16. Title = "Designing for Touch",
   17. Speaker = "James Montemagno",
   18. ImageUrl = "http://i.imgur.com/UPhZonl.jpg",
   19. Description = "Xamarin allows you to build native iOS, Android, and...",
   20. StartTime = DateTime.Now.AddHours(1),
   21. EndTime = DateTime.Now.AddHours(3)
   22. },
   23. new Session
   24. {
   25. Title = "Google Fit, Android Wear, and Xamarin",
   26. Speaker = "Chris Hardy",
   27. ImageUrl = "http://i.imgur.com/XT8VHMf.jpg",
   28. Description = "Xamarin allows you to build native iOS, Android, and...",
   29. StartTime = DateTime.Now.AddHours(1),
   30. EndTime = DateTime.Now.AddHours(3)
   31. },
   32. new Session
   33. {
   34. Title = "Effective iOS Memory Management",
   35. Speaker = "Joseph Hill",
   36. ImageUrl = "http://i.imgur.com/YDEv0oz.jpg",
   37. Description = "Xamarin allows you to build native iOS, Android, and...",
   38. StartTime = DateTime.Now.AddHours(1),
   39. EndTime = DateTime.Now.AddHours(3)
   40. },
   41. new Session
   42. {
   43. Title = "Creating Custom Layouts",
   44. Speaker = "Nat Friedman",
   45. ImageUrl = "http://i.imgur.com/DbZlxvM.jpg",
   46. Description = "Xamarin allows you to build native iOS, Android, and...",
   47. StartTime = DateTime.Now.AddHours(1),
   48. EndTime = DateTime.Now.AddHours(3)
   49. },
   50. };
   51. }
6. Your final class should start something like the code below.



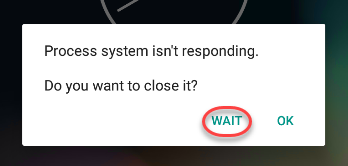
1. Open **SessionsPage.xaml**. This will serve as the view for our model **(Session)** and viewmodel **(SessionsViewModel)**.
2. Replace the contents of the file with the code below so that it is aware of our custom controls (the **xmlns:local**) and has a proper **Title**.
   1. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
   2. xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   3. xmlns:local="clr-namespace:FirstApp;assembly:FirstApp"
   4. x:Class="FirstApp.SessionsPage"
   5. Title="Build 2017">
   6. </ContentPage>
3. Insert the declaration of the **ContentPage.Content** property within the **ContentPage** tag. This will allow you to set that property using nested XAML.
   1. <ContentPage.Content>
   2. </ContentPage.Content>
4. Insert the **ListView** below within the **ContentPage.Content** tag. This will bind to a session list we’ll configure next and display each item based on the provided template via **ItemTemplate**. In this case, the template is simply an instance of a control provided to offer a nicer user experience. Also notice the **ItemTapped** event is wired up to a method we’ll need to implement later as well. There are also some additional properties being set. **HasUnevenRows** is set to **true** so that items can have different heights depending on the size they need. **SeparatorVisibility** is set to **None** so that there is no separator between cells.
   1. <ListView ItemsSource="{Binding Sessions}"
   2. HasUnevenRows="true"
   3. SeparatorVisibility="None"
   4. ItemTapped="Handle\_ItemTapped">
   5. <ListView.ItemTemplate>
   6. <DataTemplate>
   7. <local:CardViewCell Text="{Binding Title}" Detail="{Binding Speaker}"
   8. ImageSource="{Binding ImageUrl}"/>
   9. </DataTemplate>
   10. </ListView.ItemTemplate>
   11. </ListView>
5. The final XAML file should start as shown below. Note that the main advantage of using XAML is that it helps keep a clean separation of markup and code using the MVVM pattern. You can then concentrate on the behavior of your application and let a design team handle the rest.



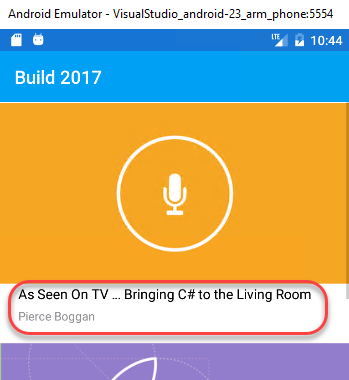
1. Open **SessionsPage.xaml.cs**.
2. Add the following line to the end of the constructor to set up the data binding source. The binding mechanisms in Xamarin.Forms should be very familiar to WPF developers. You typically have a binding target and binding source which supports different binding modes such as **OneWay** and **TwoWay**.
   1. BindingContext = new SessionsViewModel();
3. Add the **Handle\_ItemTapped** method below to provide a handler for when items are tapped. There will be an error in the code because **SessionDetailPage** has not yet been created. We’ll do that next.
   1. async void Handle\_ItemTapped(object sender, Xamarin.Forms.ItemTappedEventArgs e)
   2. {
   3. var session = (Session)e.Item;
   4. await Navigation.PushAsync(new SessionDetailPage(session));
   5. }
4. Add another **Forms Blank Content Page Xaml** file to the shared project and name it **SessionDetailPage.xaml**.
5. Add the markup below inside the **ContentPage** tag. Note that this also uses data binding for virtually every field. We will need to set the source next.
   1. <ContentPage.Content>
   2. <StackLayout VerticalOptions="StartAndExpand">
   3. <Image Source="{Binding ImageUrl}" />
   4. <StackLayout Padding="10" Spacing="5">
   5. <Label Text="{Binding Title}" Font="25,Bold" />
   6. <Label Text="{Binding Speaker}" FontSize="22" />
   7. <Label Text="{Binding StartTime}" FontSize = "20" />
   8. <Label Text="{Binding CreationDate, StringFormat='{0:HH:mm:ss}'}" />
   9. <Label Text="{Binding Description}" LineBreakMode="WordWrap" />
   10. </StackLayout>
   11. </StackLayout>
   12. </ContentPage.Content>
6. Open SessionDetailPage.xaml.cs.
7. Add the following property to the class. It will represent a session and be our data source.
   1. public Session Session { get; set; }
8. Add an overloaded constructor that accepts a **Session** parameter as shown below. It will set up the data binding relationship based on the parameter passed in.
   1. public SessionDetailPage(Session session)
   2. {
   3. InitializeComponent();
   4. Session = session;
   5. BindingContext = Session;
   6. }
9. Open **App.xaml**.
10. Insert the markup below inside the **Application.Resources** tag. This is just some styling to make the final product a little nicer. As WPF developers, we are used to the ability of creating templates and styles for our XAML markup. We are simply applying a small style to our **NavigationPage**.
    1. <ResourceDictionary>
    2. <Style TargetType="NavigationPage">
    3. <Setter Property="BarBackgroundColor" Value="#00a1f1" />
    4. <Setter Property="BarTextColor" Value="White" />
    5. </Style>
    6. </ResourceDictionary>

# Task 3: Building and deploying the app to an emulator (or a device)

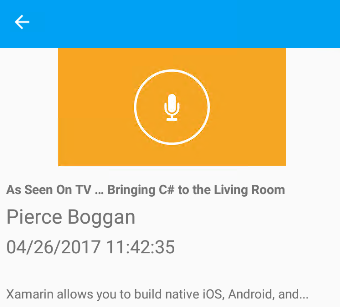
1. Press **F5** to build and run the project. The project has been configured to deploy the Android app to the emulator launched earlier, but you’d follow the same process for deploying to other platforms and/or real devices.
2. Switch over to the Android emulator. If you see this dialog, click **Wait**. Sometimes you need to click it twice.



1. The build and deploy process may take a few minutes. Once it’s done, you’ll be able to take it for a spin. If the app doesn’t load correctly, return to Visual Studio and select **Debug | Stop Debugging**. This will keep the emulator running while you make changes to the project and redeploy. If you close the emulator, it will need to boot again.
2. Once the app loads, tap one of the sessions to navigate to its details page.



1. And that’s it! You just built a cross-platform mobile app with a lot of potential in a few minutes. You can continue to explore the app by adding more functionality or debugging with breakpoints.



# Summary

Congratulations on completing this Quick Start Challenge! In this lab, you’ve learned how to apply your WPF experience toward building mobile applications that span Android, iOS, and Windows.

# Additional Resources

If you are interested in learning more about this topic, you can refer to the following resources:

**Documentation**: <https://developer.xamarin.com>

**GitHub**: <https://github.com/xamarin>

**Team blog**: <https://blog.xamarin.com>