Xamarin

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Xamarin is a software company based in San Francisco. It provides commercial software development tools that allow a user to develop applications for Android, iOS and Windows using C# language and the .NET framework.

Xamarin is built on the .NET Framework. It allows one to create apps that easily run across multiple platforms. In this tutorial, we will explain how you can use Xamarin to deliver native iOS, Android, and Windows Apps.

Audience

This tutorial has been developed for beginners to help them understand the basics of creating native Apps using Xamarin.

Prerequisites

All the programs in this tutorial have been developed using Visual C#. Therefore, you should have a good understanding of code written in C# programming language.

Copyright & Disclaimer

Table of Contents

1. Xamarin – Installation

Xamarin is built on the .NET Framework. It allows one to create apps that easily run across multiple platforms. In this tutorial, we will explain how you can use Xamarin to deliver native iOS, Android, and Windows Apps.

Let's start the tutorial with a discussion on how to install Xamarin in Windows and Mac systems.

System Requirements

Windows

- A computer with at least 2GB of RAM and running Windows 7 or higher *(Windows 8-10 is highly recommended)*
- Visual Studio 2012 Professional or higher
- Xamarin for Visual Studio

Mac

- A Mac computer running OS X Yosemite (10.10) or higher
- Xamarin iOS SDK
- Apple's Xcode (7+) IDE and iOS SDK
- Xamarin Studio

Installation on Windows

Download the Xamarin Installer from http://www.xamarin.com/download. Before running the Xamarin installer, make sure you have installed Android SDK and Java SDK on your computer.

Run the downloaded installer to begin the installation process:

- The Xamarin license agreement screen appears. Click the **Next** button to accept the agreement.
- The installer will search for any missing components and prompt you to download and install them.
- After the Xamarin installation is complete, click the **Close** button to exit and get ready to start using Xamarin.

Installation on Mac

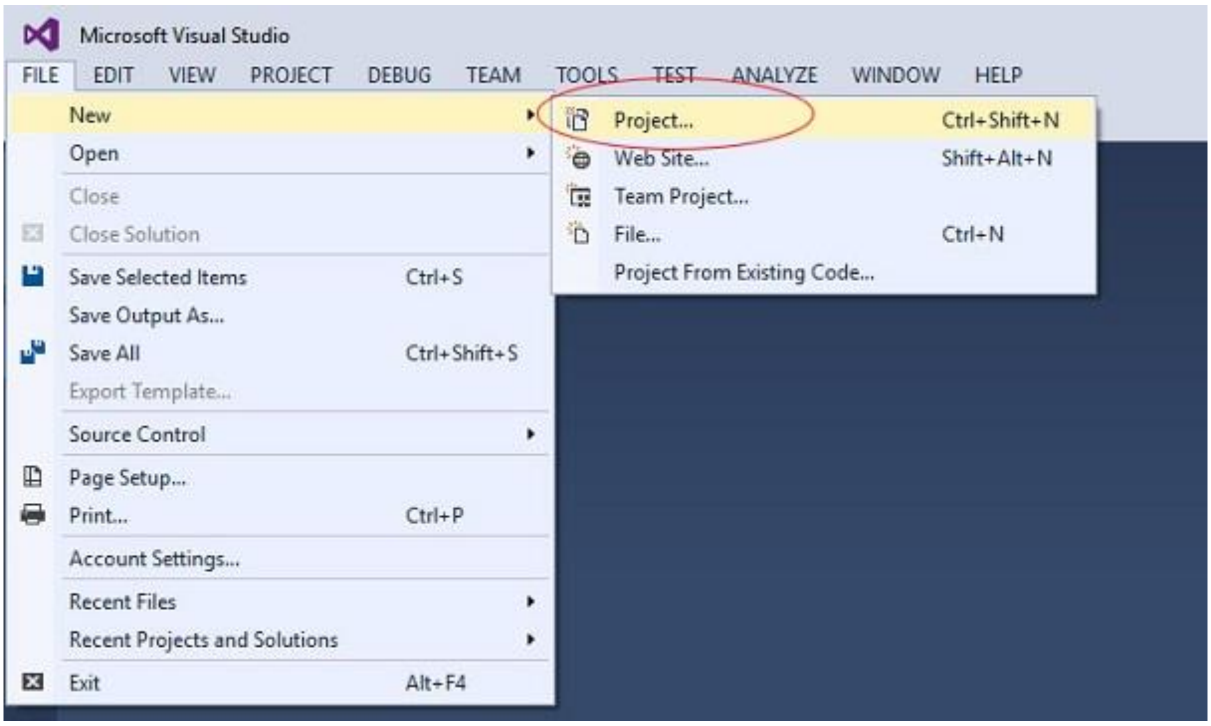
- Download the Xamarin Studio Installer on your Mac system.
- Run the Xamarin installer you downloaded and follow the steps given in the Installation Wizard.
- After the installation is complete, you can start using Xamarin on your system.

2. Xamarin – First Application

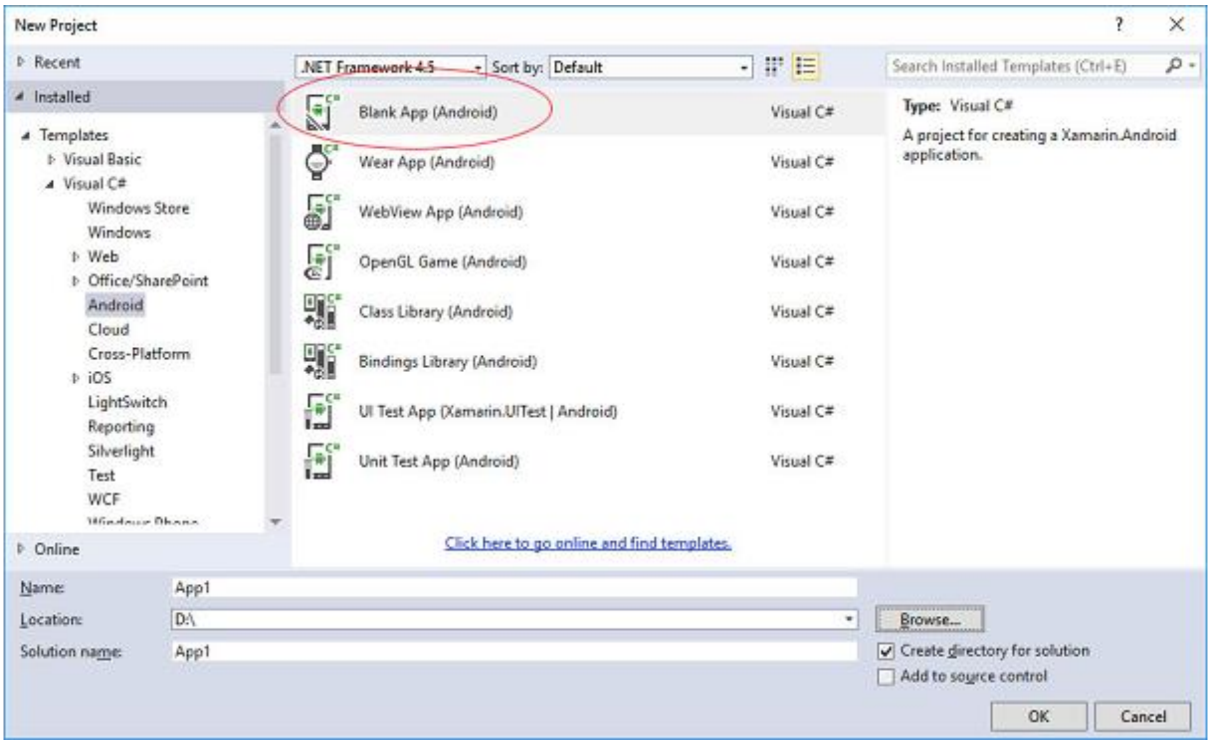
In this chapter, we will see how to create a small Android application using Xamarin.

Hello Xamarin! Application

First of all, start a new instance of Visual Studio and go to **File -> New -> Project**.



On the Menu dialog box that appears, go to **Templates -> Visual C# -> Android -> Blank App (Android)**.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Store
Windows
Web
Office/SharePoint
Android
Cloud
Cross-Platform
iOS
LightSwitch
Reporting
Silverlight
Test
WCF
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Android) Visual C#
Wear App (Android) Visual C#
WebView App (Android) Visual C#
OpenGL Game (Android) Visual C#
Class Library (Android) Visual C#
Bindings Library (Android) Visual C#
UI Test App (Xamarin.UITest | Android) Visual C#
Unit Test App (Android) Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#

A project for creating a Xamarin.Android application.

Name: App1

Location: D:\

Solution name: App1

[✓] Create directory for solution

[ ] Add to source control

Give an appropriate name for your application. In our case, we name it "**helloWorld**" and save it in the default location provided. Next, click the OK button for the new "**helloXamarin**" project to load.

On the **solution**, open **Resources -> layout -> Main.axml** file. Switch from **Design View** and go to the **Source** file and type the following lines of code to build your app.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#d3d3d3"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:text="@string/HelloXamarin"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView2"
        android:textColor="@android:color/black" />
</LinearLayout>
```

In the above code, we have created a new Android **textview**. Next, open the folder values and double-click **Strings.xml** to open it. Here, we are going to store information and values about the **button** created above.

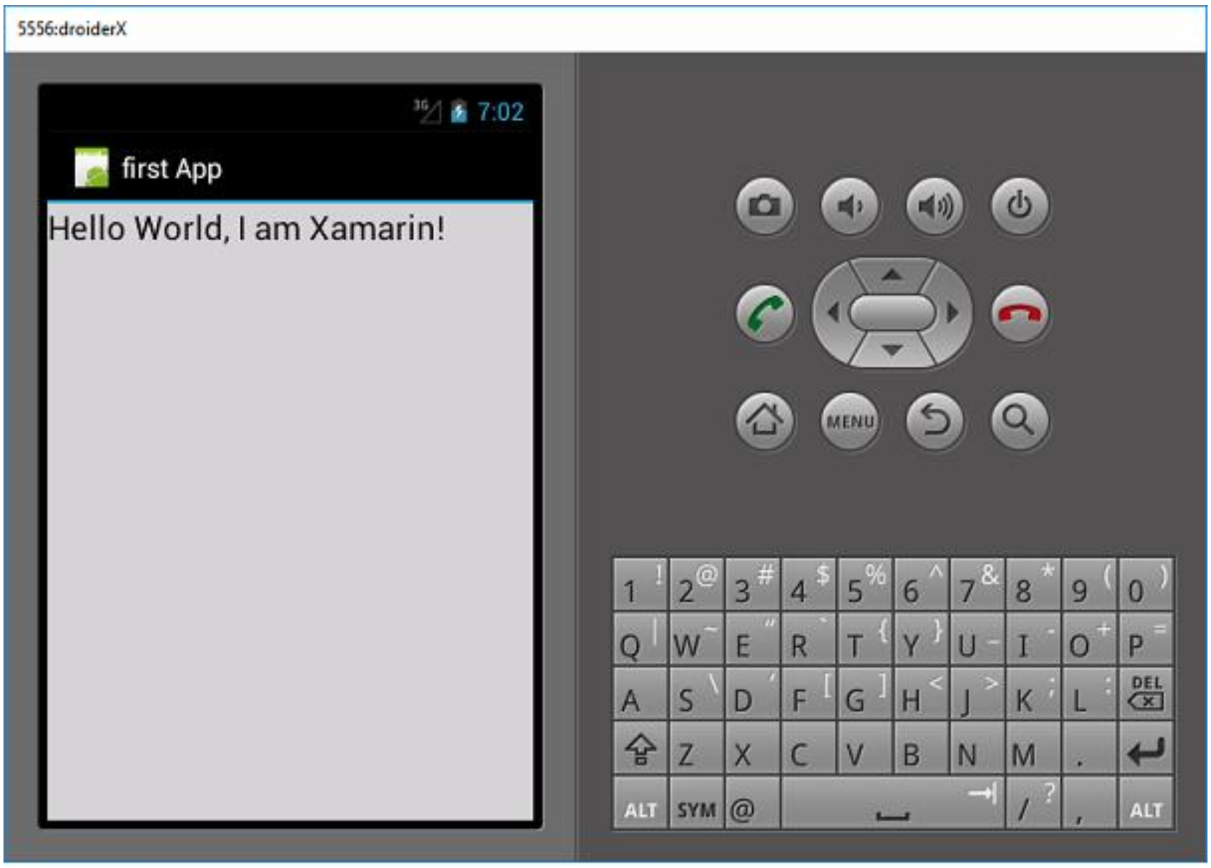
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="HelloXamarin">Hello World, I am Xamarin!</string>
    <string name="ApplicationName">helloWorld</string>
</resources>
```

Open **MainActivity.cs** file and replace the existing code with the following lines of code.

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace HelloXamarin
{
    public class MainActivity : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView(Resource.Layout.Main);
        }
    }
}
```

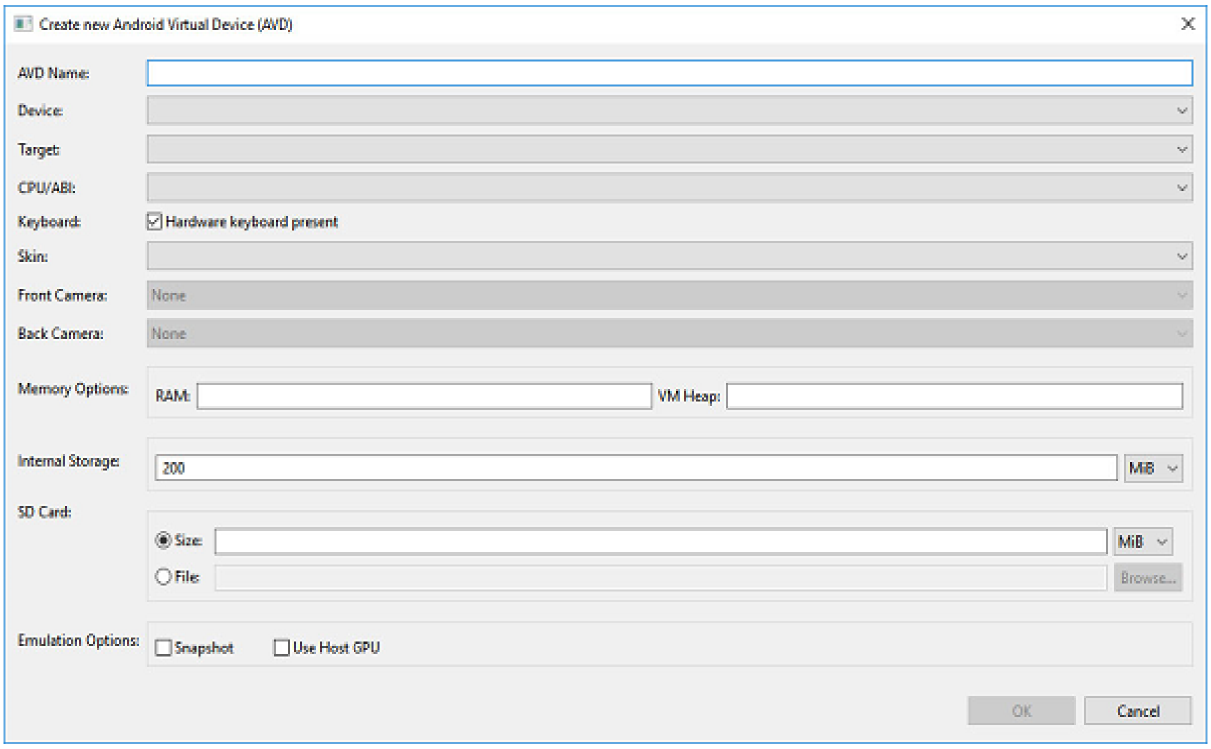
Save the application. Build and then run it to display the created app in an Android Emulator.



If you do not have an Android Emulator, then follow the steps given in the next section to create one.

Setting Up an Android Emulator

On your Visual Studio menu, go to **Tools -> Android -> Android Emulator Manager**. On the pop-up window that appears, click the **Create** button. It will display the following screen.

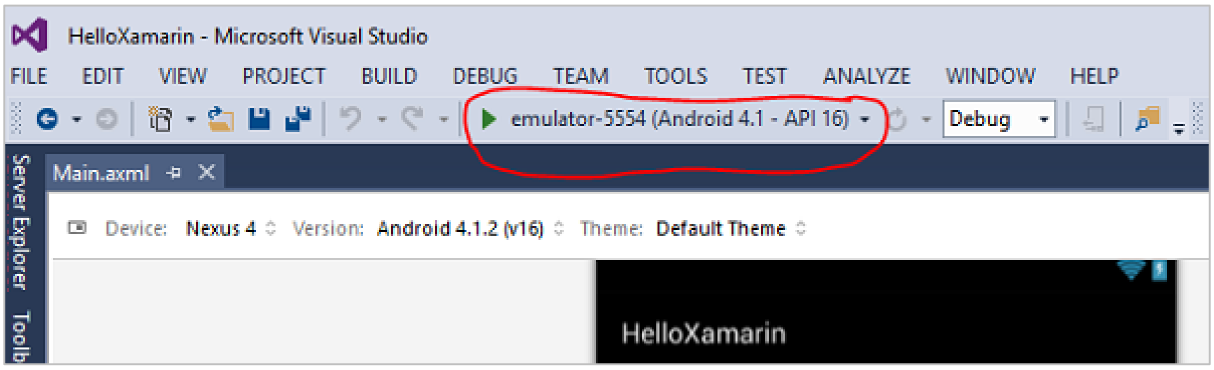


Create new Android Virtual Device (AVD)

AVD Name:
Device:
Target:
CPU/ABI:
Keyboard: ☑ Hardware keyboard present
Skin:
Front Camera: None
Back Camera: None
Memory Options: RAM: VM Heap:
Internal Storage: 200 MiB
SD Card: ● Size: MiB ○ File: Browse...
Emulation Options: ☐ Snapshot ☐ Use Host GPU
OK Cancel

On the above screen, supply the **AVD name** you want. Select a **device** that is appropriate for your display, e.g., Nexus 4" display. Select your **target platform**. It is always advisable to test on a minimum target platform, e.g., API 10 Android 2.3 (Gingerbread) so as to ensure your App works across all Android platforms.

Fill in the rest of the fields and click the OK button. Your emulator is now ready. You can select it from the list of existing Android Virtual Devices and then click **Start** to launch it.



HelloXamarin - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP

emulator-5554 (Android 4.1 - API 16) Debug

Main.axml

Device: Nexus 4 Version: Android 4.1.2 (v16) Theme: Default Theme

Modifying HelloXamarin App

In this section, we will modify our project and create a button which will display text upon click. Open **main.axml** and switch to **source view**. After our **textview** that we created, we will add a button as shown below.

```
<Button
        android:id="@+id/MyButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/ButtonClick" />
```

After adding a button, our full code will look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:text="@string/HelloXamarin"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView2" />
    <Button
        android:id="@+id/MyButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/ButtonClick" />
</LinearLayout>
```

Next, we register our button values in the **strings.xml** file.

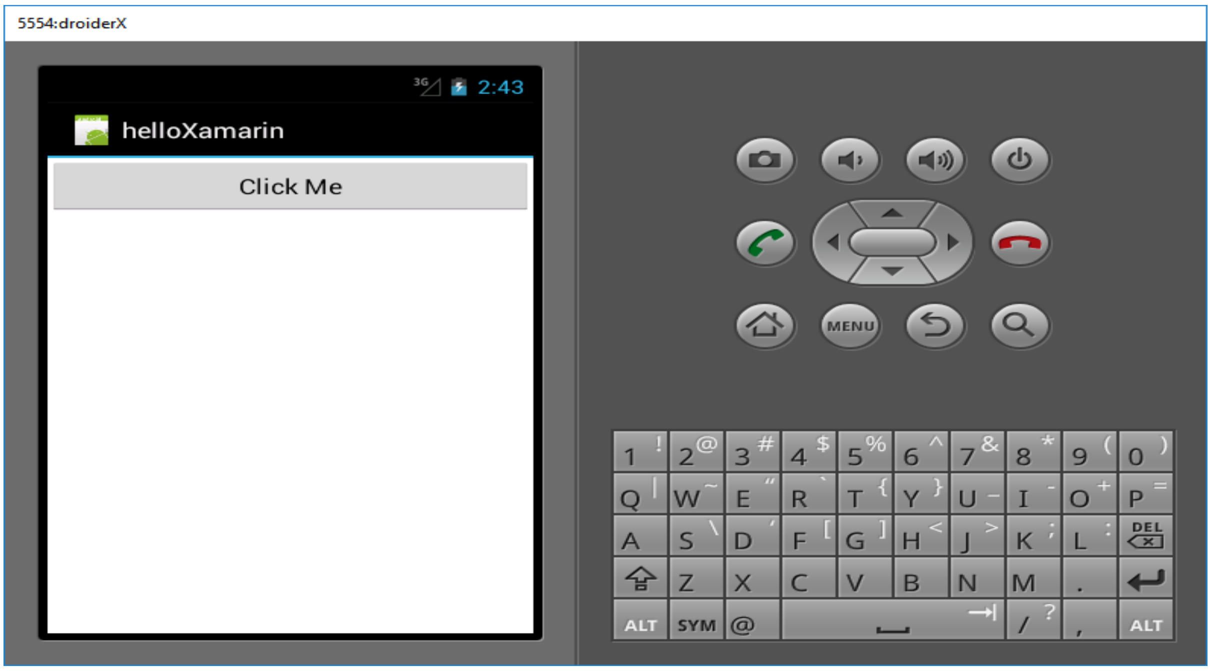
```
<string name="ButtonClick">Click Me!</string>
```

After adding our button in the **strings.xml** file, we will open **MainActivity.cs** file to add an action for our button when it is clicked, as shown in the following code.

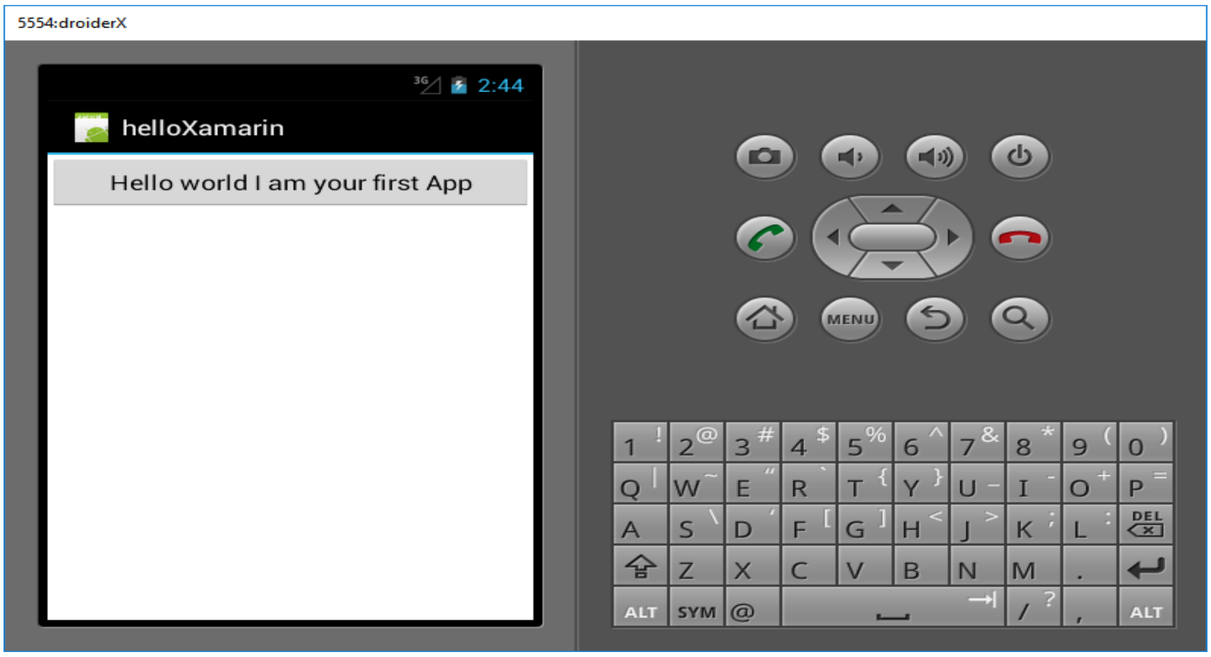
```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace HelloXamarin
{
    [Activity(Label = "HelloXamarin", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            SetContentView(Resource.Layout.Main);
            Button button = FindViewById<Button>(Resource.Id.MyButton);
            button.Click += delegate { button.Text = "Hello world I am your
first App"; };
        }
    }
}
```

Next, build and run your application.



After clicking on the button, you will get the following output:



Hello world I am your first App

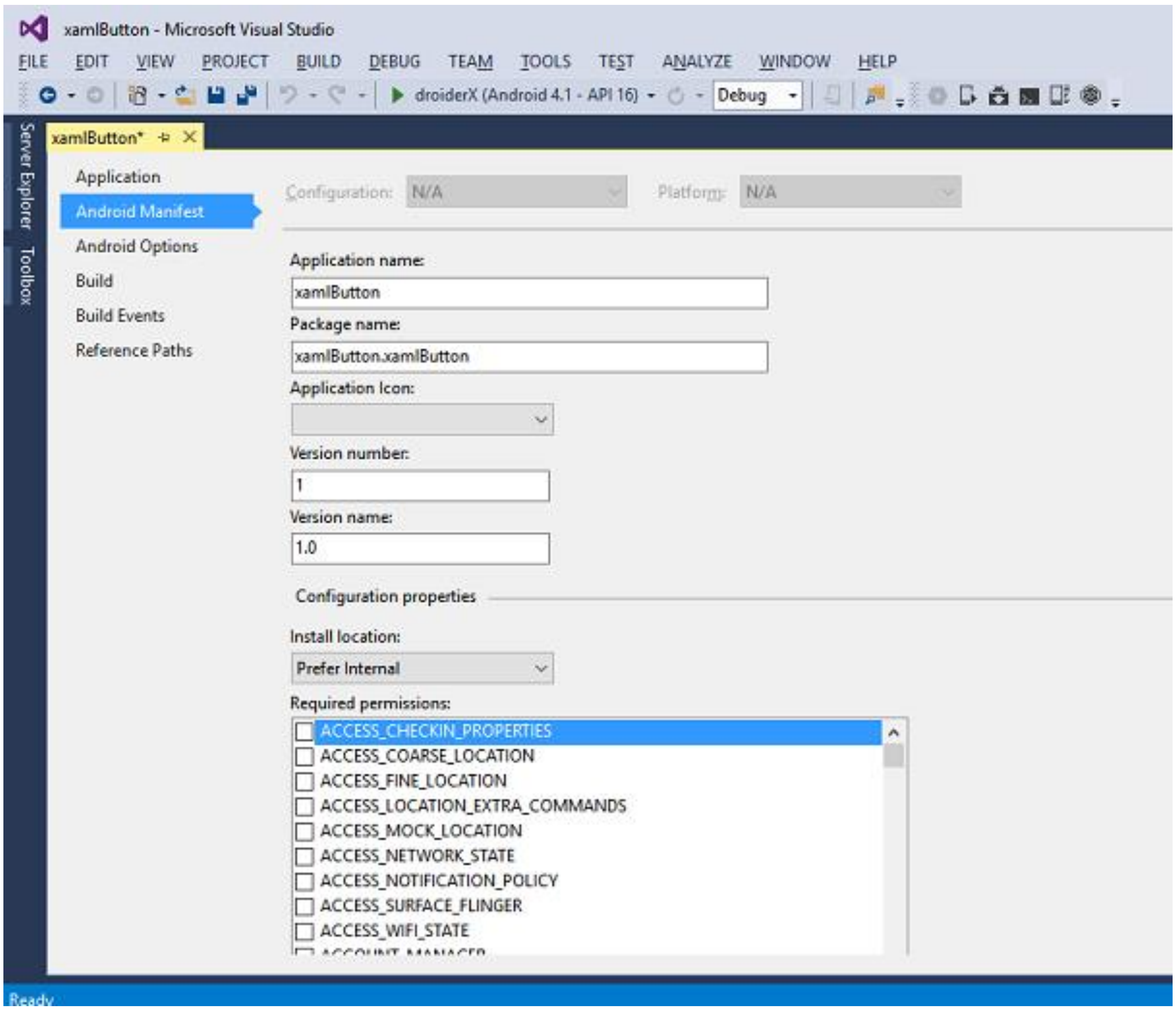
3. Xamarin – Application Manifest

All Android Apps have a **manifest file** commonly referred to as **AndroidManifest.xml**. The manifest file contains everything about the Android platform that an App needs in order to run successfully.

Here, we have listed down some of the important functions of a manifest file:

- It declares the **minimum API level** required by the application.
- It declares the permissions required by the application, e.g., camera, location, etc.
- It gives permissions to hardware and software features used or required by the application.
- It lists the libraries that the application must be linked.

The following screenshot shows a Manifest file.


xamlButton - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
droiderX (Android 4.1 - API 16)
Debug
Server Explorer
Toolbox
xamlButton*
Application
Android Manifest
Android Options
Build
Build Events
Reference Paths
Ready

Configuration: N/A
Platform: N/A

Application name: xamlButton
Package name: xamlButton.xamlButton
Application Icon:
Version number: 1
Version name: 1.0

Configuration properties

Install location: Prefer Internal

Required permissions:
- [ ] ACCESS_CHECKIN_PROPERTIES
- [ ] ACCESS_COARSE_LOCATION
- [ ] ACCESS_FINE_LOCATION
- [ ] ACCESS_LOCATION_EXTRA_COMMANDS
- [ ] ACCESS_MOCK_LOCATION
- [ ] ACCESS_NETWORK_STATE
- [ ] ACCESS_NOTIFICATION_POLICY
- [ ] ACCESS_SURFACE_FLINGER
- [ ] ACCESS_WIFI_STATE

Application name: It refers to the title of your App

Package name: It is a unique name used to identify your App.

Application Icon: It is the icon displayed on the Android home screen for your App.

Version Number: It is a single number that is used to show one version of your App is more recent than another.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" >
```

Version Name: It is a user-friendly version string for your App that users will see on your App settings and on the Google PlayStore. The following code shows an example of a version name.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionName="1.0.0">
```

Minimum Android Version: It is the lowest Android version platform which your application supports.

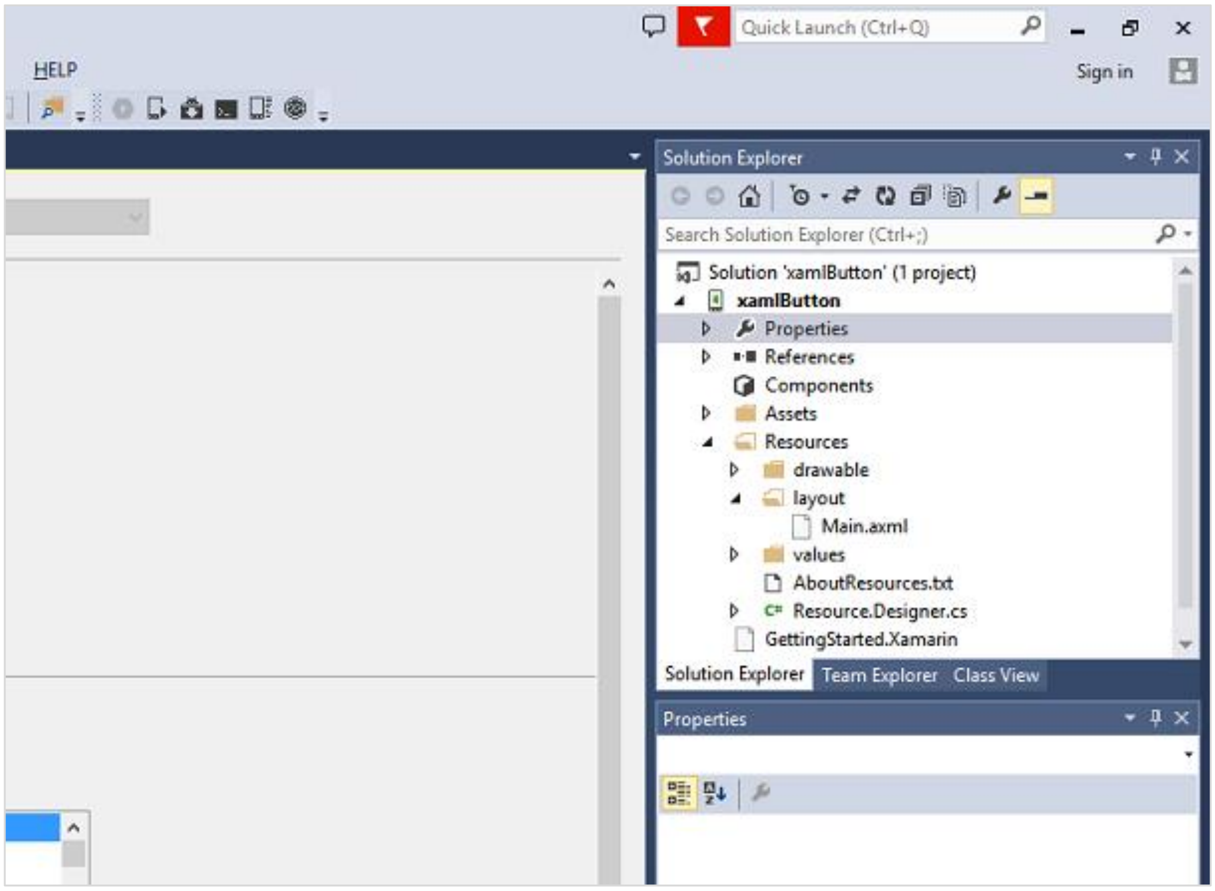
```
<uses-sdk android:minSdkVersion="16" />
```

In the above example, our minimum Android version is API Level 16, commonly referred to as **JELLY BEAN**.

Target Android Version: It is the Android version on which your App is compiled against.

4. Xamarin – Android Resources

When a new Android project is created, there are some files that are added to the project, by default. We call these default project files and folders as **Android Resources**. Take a look at the following screenshot.



The default Android resources include the following:

- **AndroidManifest.xml file:** It contains information about your Android applications, e.g., the application name, permissions, etc.
- Resources folder: Resources can be images, layouts, strings, etc. that can be loaded via Android's resource system.
- Resources/drawable folder: It stores all the images that you are going to use in your application.
- Resources/layout folder: It contains all the Android XML files (.axml) that Android uses to build user interfaces.

- **The Resources/values folder:** It contains XML files to declare key-value pairs for strings (and other types) throughout an application. This is how localization for multiple languages is normally set up on Android.

- **Resources.designer.cs** This file is created automatically when the Android projected is created and it contains unique identifiers that reference the Android resources.

- MainActivity.cs file: This is the first activity of your Android application and from where the main application actions are launched from.

Resource files can be accessed programmatically through a **unique ID** which is stored in the **resources.designer.cs** file. The ID is contained under a class called **Resource**. Any resource added to the project is automatically generated inside the **resource class**.

The following code shows how to create a gridview project containing seven images.

```
namespace HelloGridView
{
     [System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Android.Build.Tas
ks", "1.0.0.0")]
     public partial class Resource
     {
     static Resource()
            {
            global::Android.Runtime.ResourceIdManager.UpdateIdValues();
            }

            public static void UpdateIdValues() { }
            public partial class Attribute
            {
                   static Attribute()
                   {
                   global::Android.Runtime.ResourceIdManager.UpdateIdValues();
                   }
                   private Attribute() { }
            }
            public partial class Drawable
            {
                   // aapt resource value: 0x7f020000
                   public const int Icon = 2130837504;
                   // aapt resource value: 0x7f020001
                   public const int img1 = 2130837505;


                   // aapt resource value: 0x7f020002
                   public const int img2 = 2130837506;
                   // aapt resource value: 0x7f020003
                   public const int img3 = 2130837507;

                   // aapt resource value: 0x7f020004
                   public const int img4 = 2130837508;

                   // aapt resource value: 0x7f020005
                   public const int img5 = 2130837509;

                   // aapt resource value: 0x7f020006
                   public const int img6 = 2130837510;

                   // aapt resource value: 0x7f020007
                   public const int img7 = 2130837511;

                   static Drawable()
                   {
                   global::Android.Runtime.ResourceIdManager.UpdateIdValues();
                   }
                   private Drawable() { }
            }

            public partial class Id
            {
                   // aapt resource value: 0x7f050000
                   public const int gridview = 2131034112;

                   static Id()
                   {
                   global::Android.Runtime.ResourceIdManager.UpdateIdValues();
                   }
                   private Id() { }
            }

            public partial class Layout
            {
                  // aapt resource value: 0x7f030000
                  public const int Main = 2130903040;
                  static Layout()
                  {
                  global::Android.Runtime.ResourceIdManager.UpdateIdValues();
                  }
                  private Layout() {  }
            }

            public partial class String
            {
                  // aapt resource value: 0x7f040001
                  public const int ApplicationName = 2130968577;

                  // aapt resource value: 0x7f040000
                  public const int Hello = 2130968576;

                  static String()
                  {
                  global::Android.Runtime.ResourceIdManager.UpdateIdValues();
                  }
                  private String() {  }
            }
      }
}
```

From the above code, the seven images are referenced in a class called **drawable**. These images are added programmatically. If a user adds another image to the project, it will also be added to the **drawable** class. The **gridview** contained in the project is also added and stored in a class on its own. Each item contained in the **resources folder** is automatically generated and stored in a class.