4

Cross Platform Mobile App

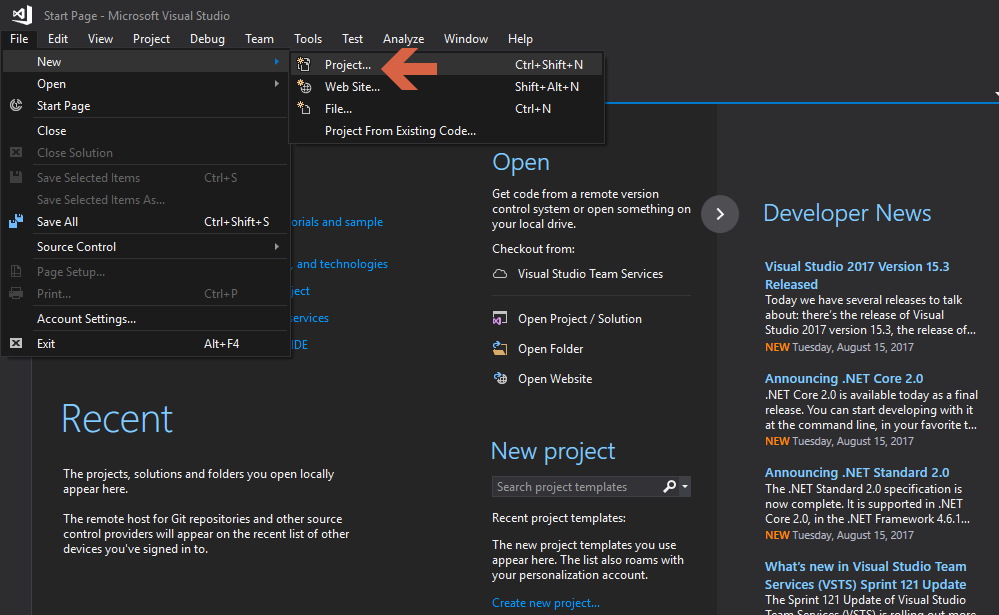
Development with Xamarin

Now that Visual Studio is installed on the Windows machine to start development and Android Virtual Device (AVD) is ready, we can start with our first Android application.

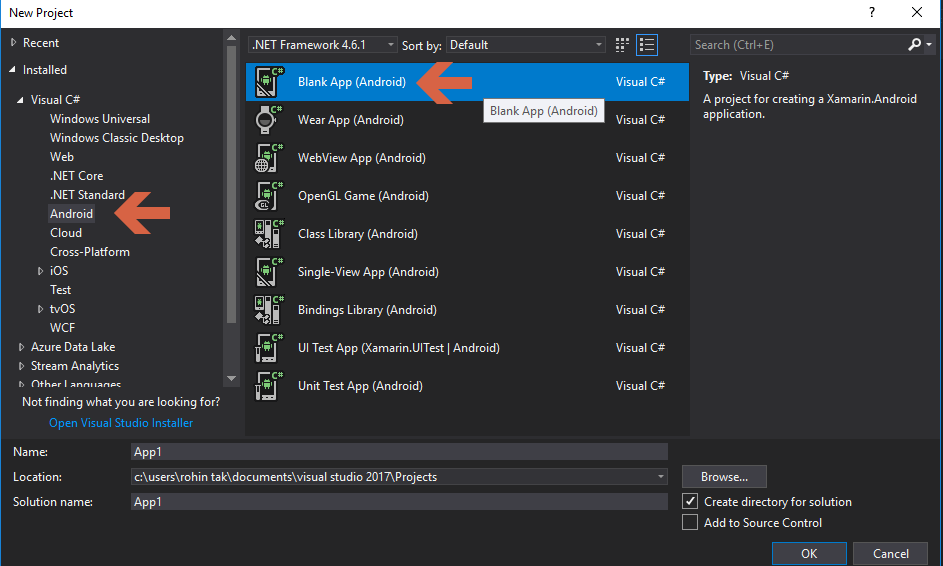
In this chapter, we are going to build our first Android application using Xamarin on Visual Studio while learning some fundamentals of the Android application development.  
  
**Create your first Android project**

To Create a new Android project in Visual Studio, follow the below steps:

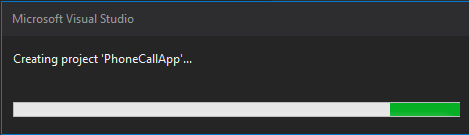
1. Click on File -> New -> Project



1. From the left pane, click on “Android” -> and then select “Blank App”



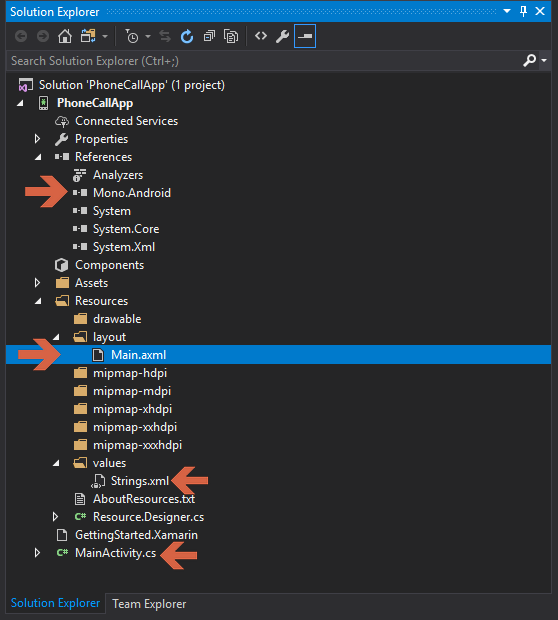
1. In the “Name” section give a name for the project and select a preferred location for your project and Click “OK”. You’ll get a screen as shown below.



Congratulations, you’ve created your first Android project on Visual Studio.

**Xamarin Solution Structure**

Once the project is created, you’ll see the solution structure like below:



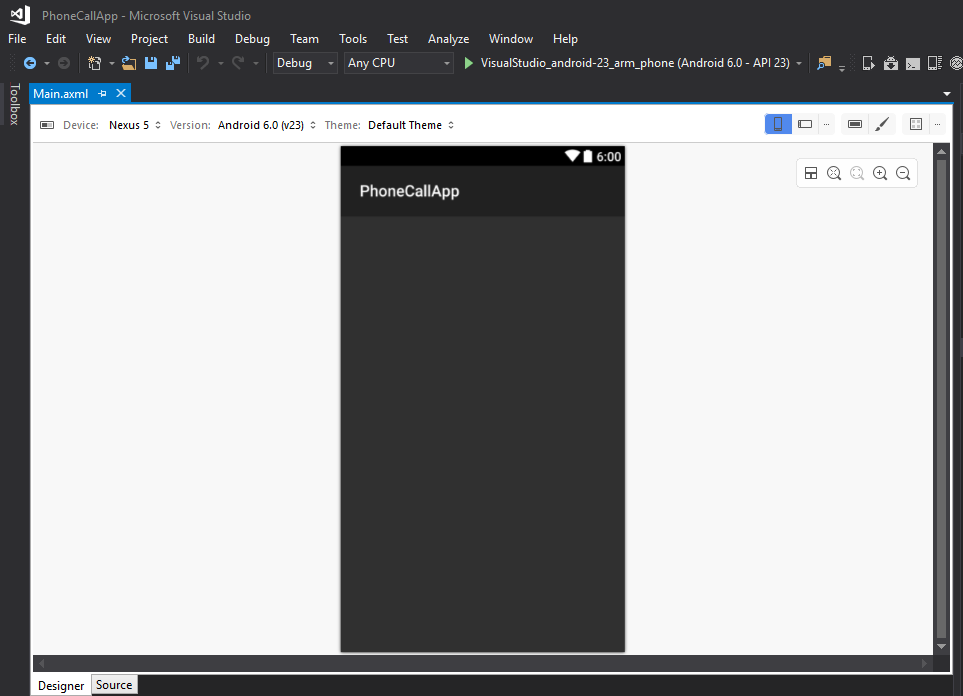
Main parts of the solution that we need to understand for now are:

* References:
  + This section will have all the required libraries for the project listed. As we can see in the image above it has Mono.Android referenced, i.e. library for Xamarin.Android.
* Resources -> It contains all the resources e.g. images, layouts and
* MainActivity.cs file that has our C# code for handling the events and other things in our main screen.

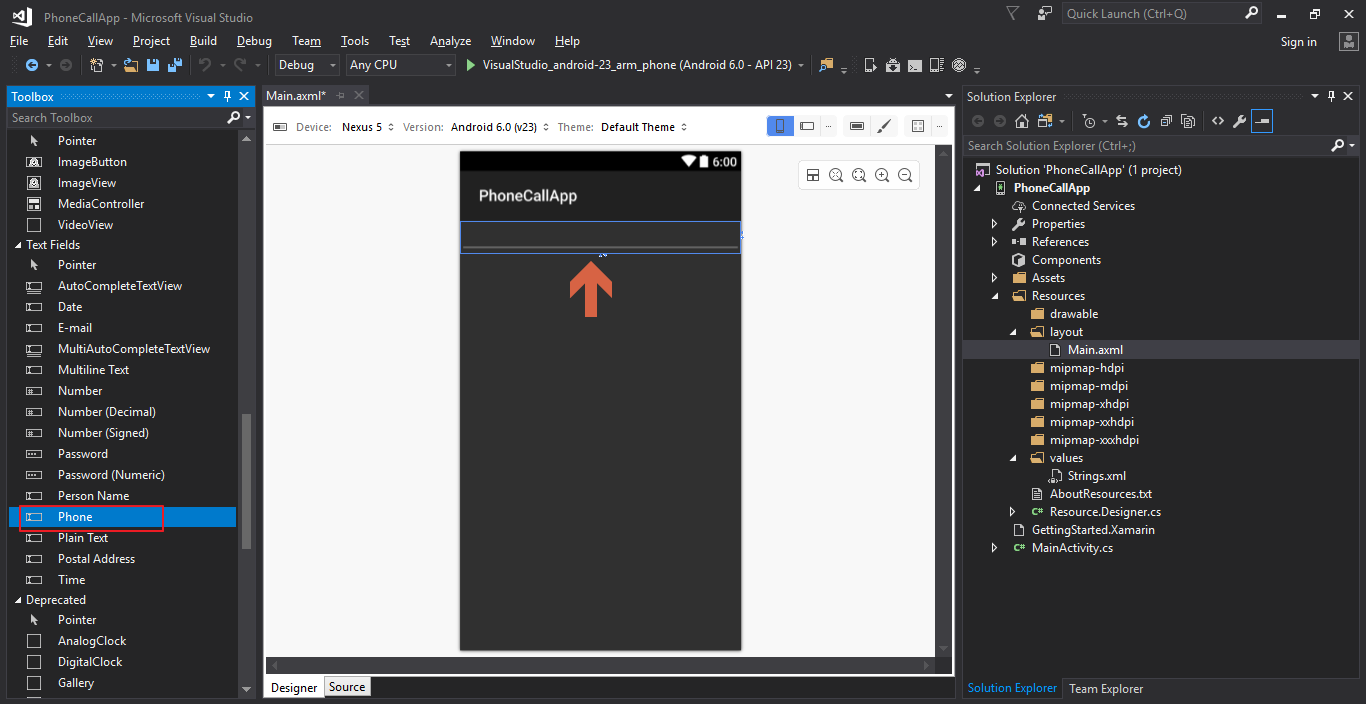
**Creating UI for the Application**

1. Let’s expand “Resources” folder we saw in the previous image and then the layout folder in the Solution Explorer. Double-click Main.axml to open it. This is the layout file for the app's screen. By default, it gets opened in the Android Designer, you can also click on the “Source” tab at the bottom to see the xml code for the same.

This Layout file is the main UI file that we’ll add our UI controls to and what we’ll see when we run our app once it is finished.



1. Let’s Add an input field to enter phone number. Drag Phone field from the toolbox (left pane) into the design view of Main.axml.

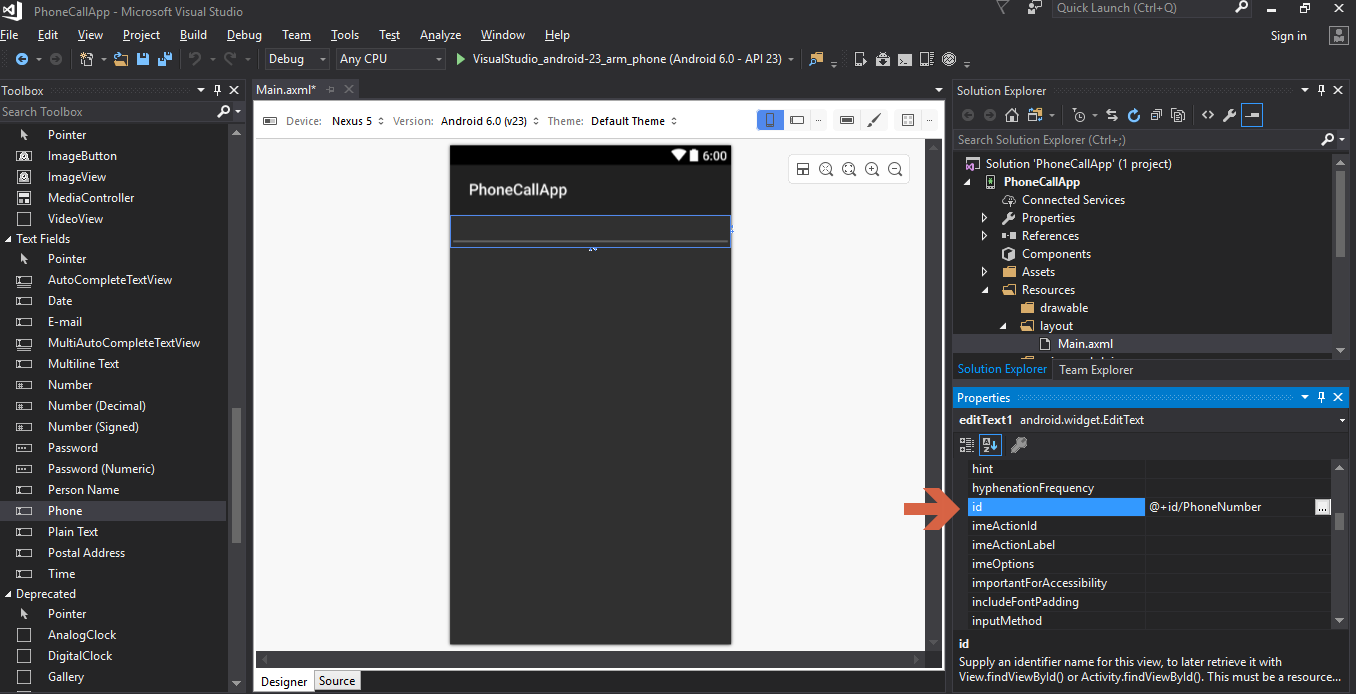


Having the phone textField gives us an advantage of restricting user to only enter a phone number. Also when user taps on the input box they’ll only get a number pad instead of a full text keyboard.

Now we have added an input for the user to put a phone number in order to make a call.

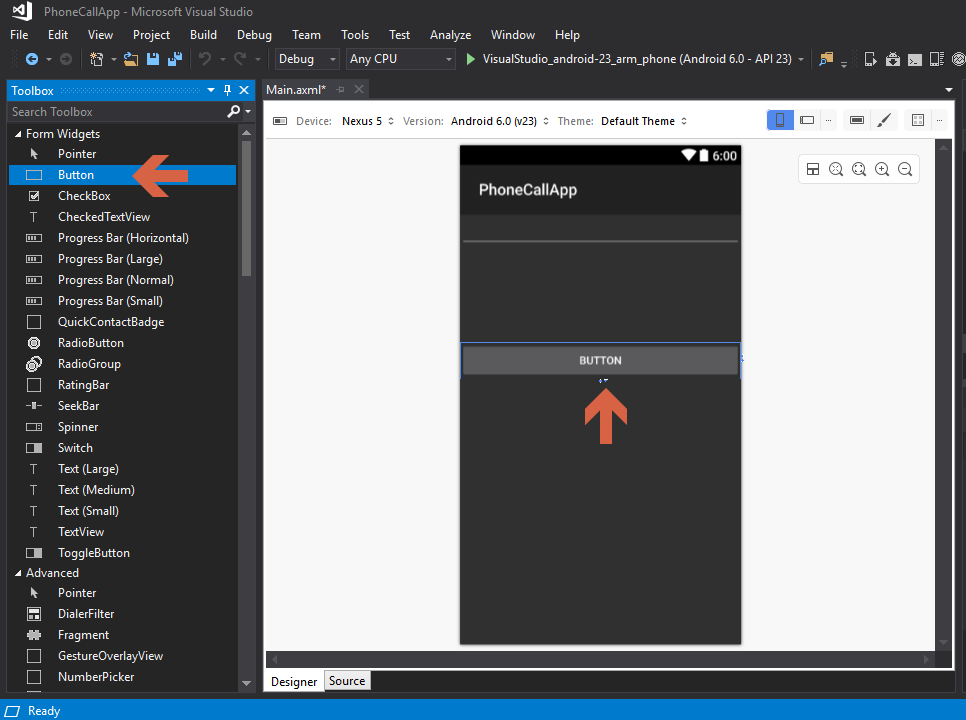
In order to recognize this field from the C# code and get the value inserted we need to give it a unique ID.

1. With the Phone TextField selected on the design surface, use the Properties pane on the right side to change the id property of the Phone input field to @+id/PhoneNumber.



Now that we have added the input field for user to enter phone number, we need a button to take the action to make a call.

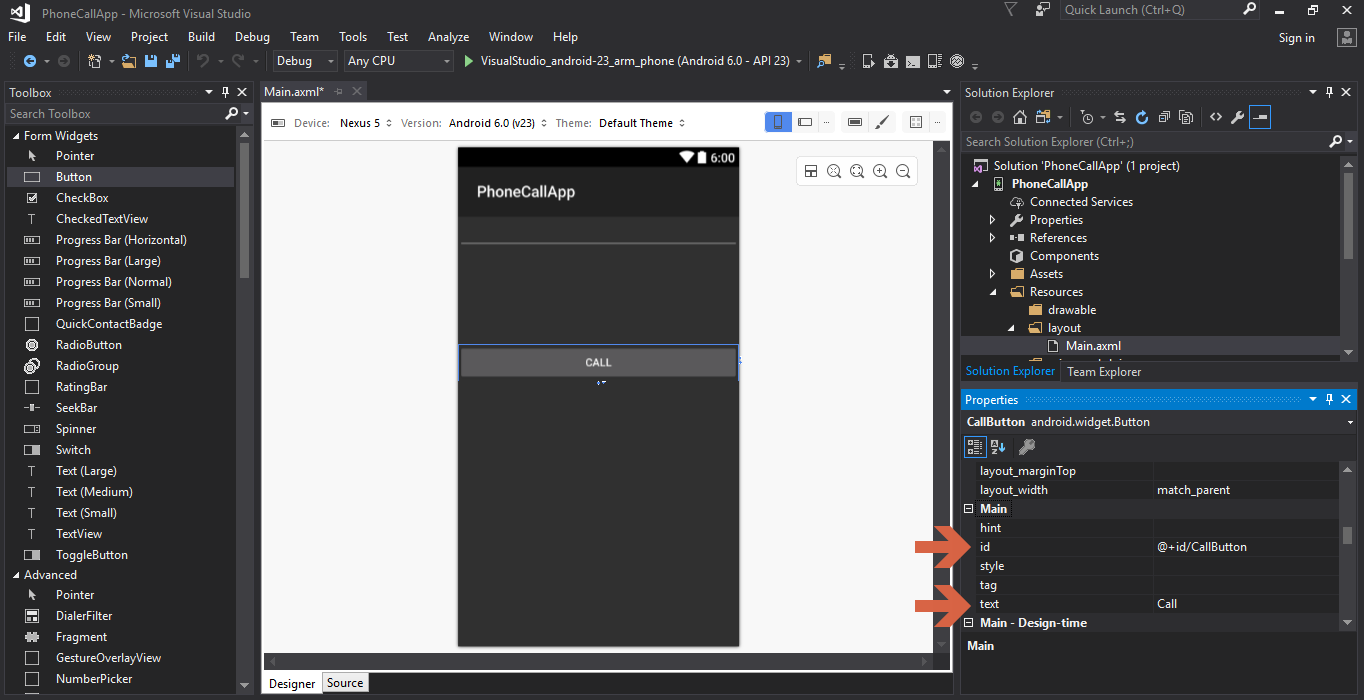
1. Drag a Button from Form widget in the left pane of the Tool Box to the designer view of Main.axml



Similar to the input field, we need to give a unique ID to the button so that our C# code can recognize when the button is clicked and we can take appropriate action, i.e. make a call to the number inserted by the user.

Also, the text on the button should say “Call” right? So let’s make that change as well in the next step.

1. Select Button in the designer and go to properties window on the right, scroll down and change the ID to “@+id/CallButton” and change the text to “Call”.



Now a basic UI is ready for our app with proper ID assigned to respective fields and button.

It is now time to move to our C# code and connect our UI with some backend code to perform some actions.

When user opens the application, MainActivity is what gets opened and Main.axml file is associated with the same.

We will learn more about Activities later, for now let’s write code to handle interactions on the MainActivity.

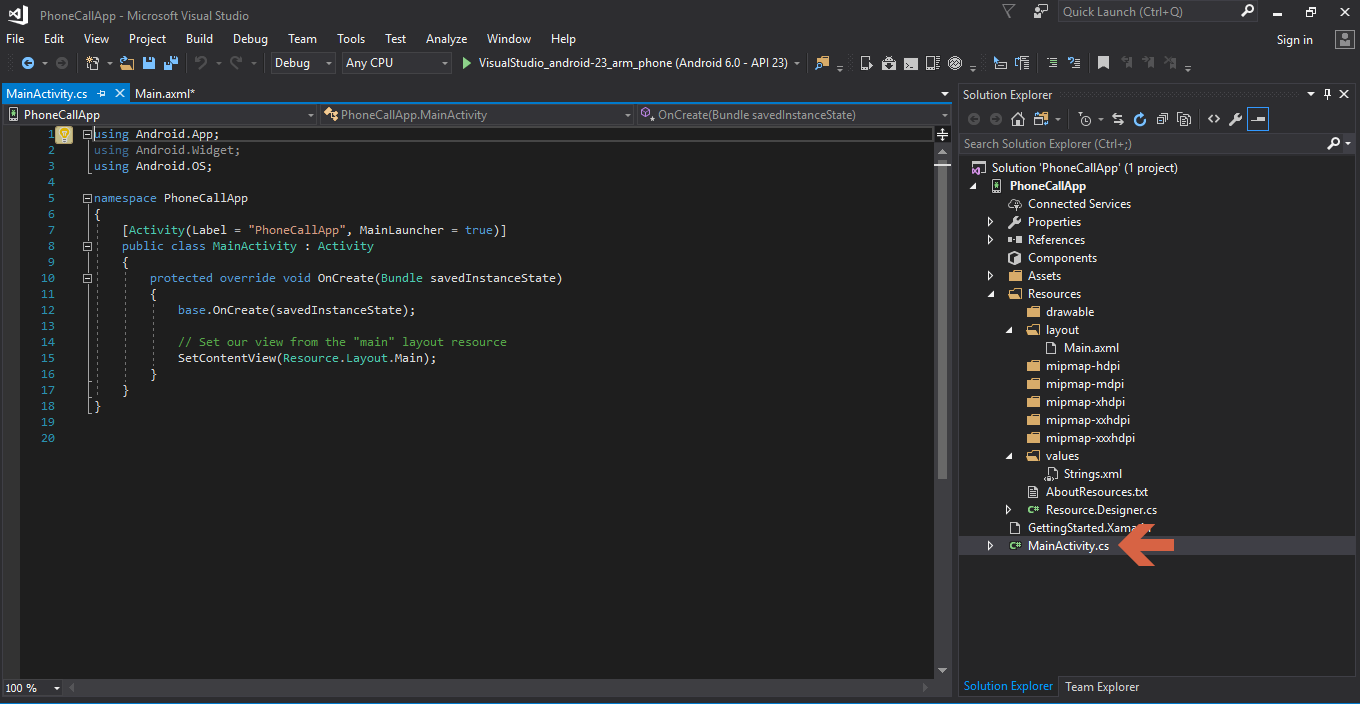
**Handelling User Interactions**

User interaction is the most important aspect of developing a mobile application. A mobile app

has to feel interactive and easy to use for the user.

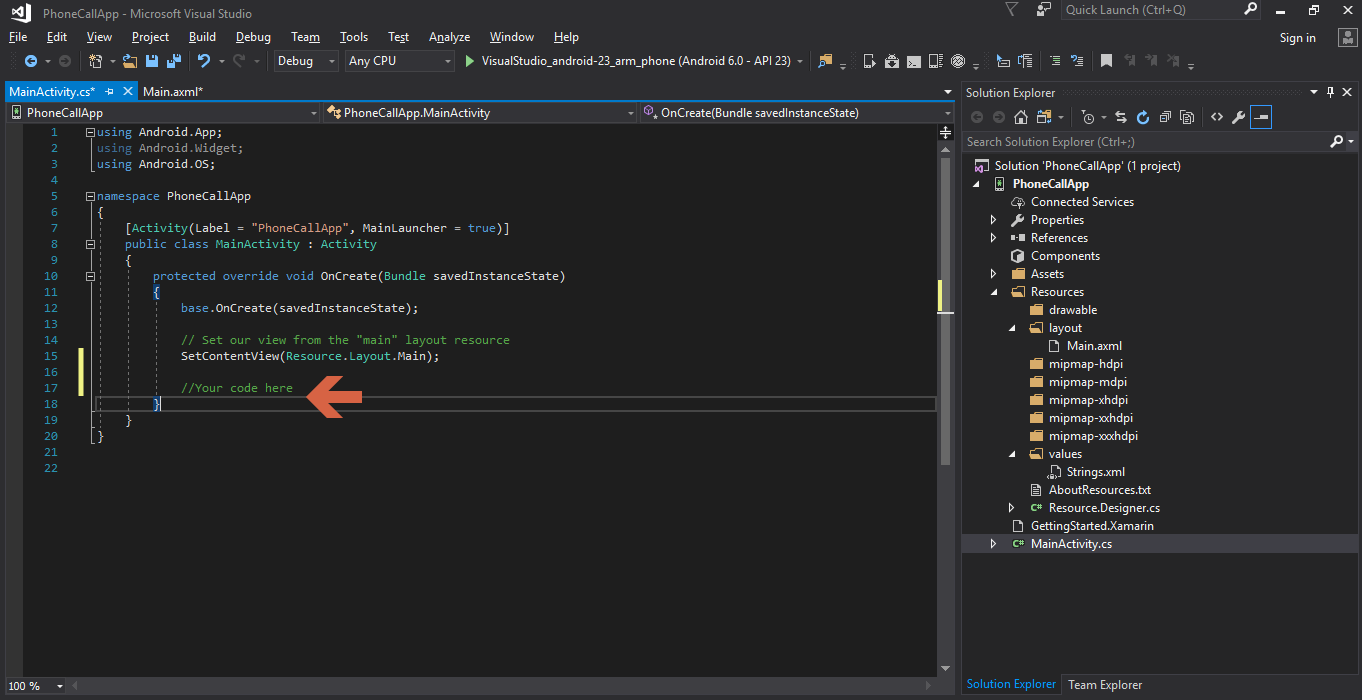
In this basic application we will be writing our user interaction code in C# and it will be part of the “MainActivity.cs” file.

1. Lets click on the MainActivity.cs file from the solution explorer in the left and open it.



It has some autogenerated code that we are going to modify in order to make our application work.

1. We need to write our code inside the OnCreate() method of MainActivity.cs file



Before we Start writing user interaction code, let’s understand the autogenerated code first.

***base.OnCreate(savedInstanceState);***

This piece of code calls the OnCreate method of the parent/base class of *MainActivity.cs* that is *Activity.cs.*

**SetContentView(Resource.Layout.Main);**

As the comments already say, it sets the view from our layout resourse file, Main.axml.

We need to write our code below *SetContentView(Resource.Layout.Mai)* as shown in the image above.

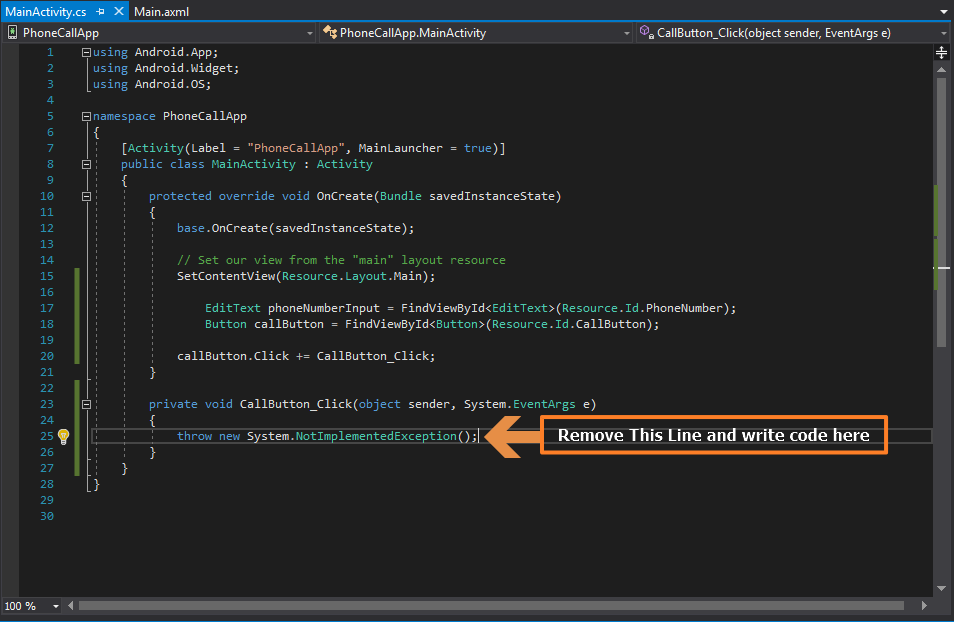
1. First, get a reference to the controls that were created in the layout file via the Android Designer i.e input box for phone number and the button to make a call.

Add the following code inside the OnCreate method:

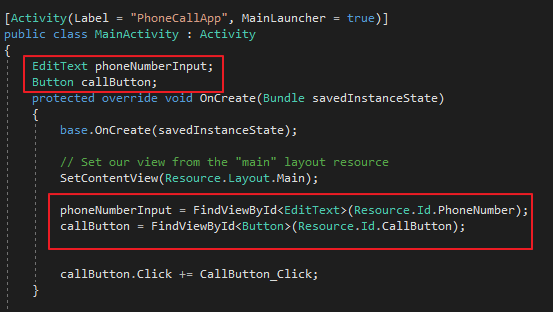
EditText phoneNumberInput = FindViewById<EditText>(Resource.Id.PhoneNumber); Button callButton = FindViewById<Button>(Resource.Id.CallButton);

1. Now that we have reference to the controls, we can write events to perform an action on the “Call” button click. Let’s write event for Call Button Click:

* Type callButton.Click += (Intellisense will give you a suggestion to hit tab)
* Hit tab for autocomplete
* This will create a method named CallButton\_Click
* We’ll be using this new created method to write our code for button click



* Because we are writing our click event in a saperate method, let’s declare the button and input field variables that we used earlier in a global scope, where all the methods of the class can have access to their reference.  
  Declare below variables on class level
  + EditText phoneNumberInput;
  + Button callButton;
* Class and the OnCreate method should now look as shown in below image.



1. In the CallButton\_Click get the value inserted in the input field by the user.

var phoneNumber = phoneNumberInput.Text;

1. Next, we will create an alert dialog box to ask user’s confirmation before making the actual call.  
   To make that dialog box write below code:
   * *var callDialog = new AlertDialog.Builder(this);*
   * We need to set 2 things in this dialog box:
     1. Message to show user:

*callDialog.SetMessage("Do you want to call " + phoneNumber + "?");*

* + 1. Events for OK and Cancel Button of the dialog box:

*callDialog.SetMessage("Do you want to call " + phoneNumber + "?"); callDialog.SetNeutralButton("Call", delegate {*

*var callIntent = new Intent(Intent.ActionCall);*

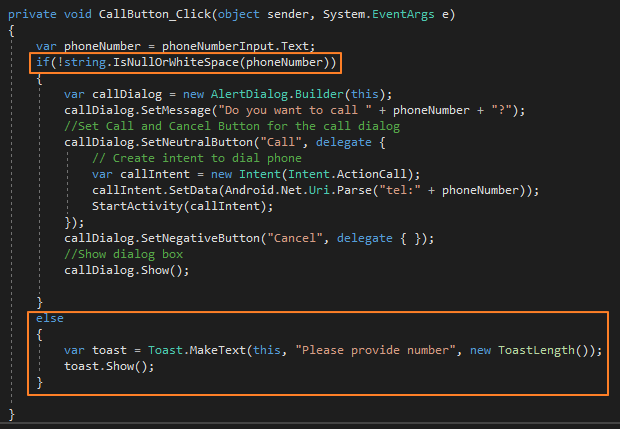
*callIntent.SetData(Android.Net.Uri.Parse("tel:" + phoneNumber));*

*StartActivity(callIntent);*

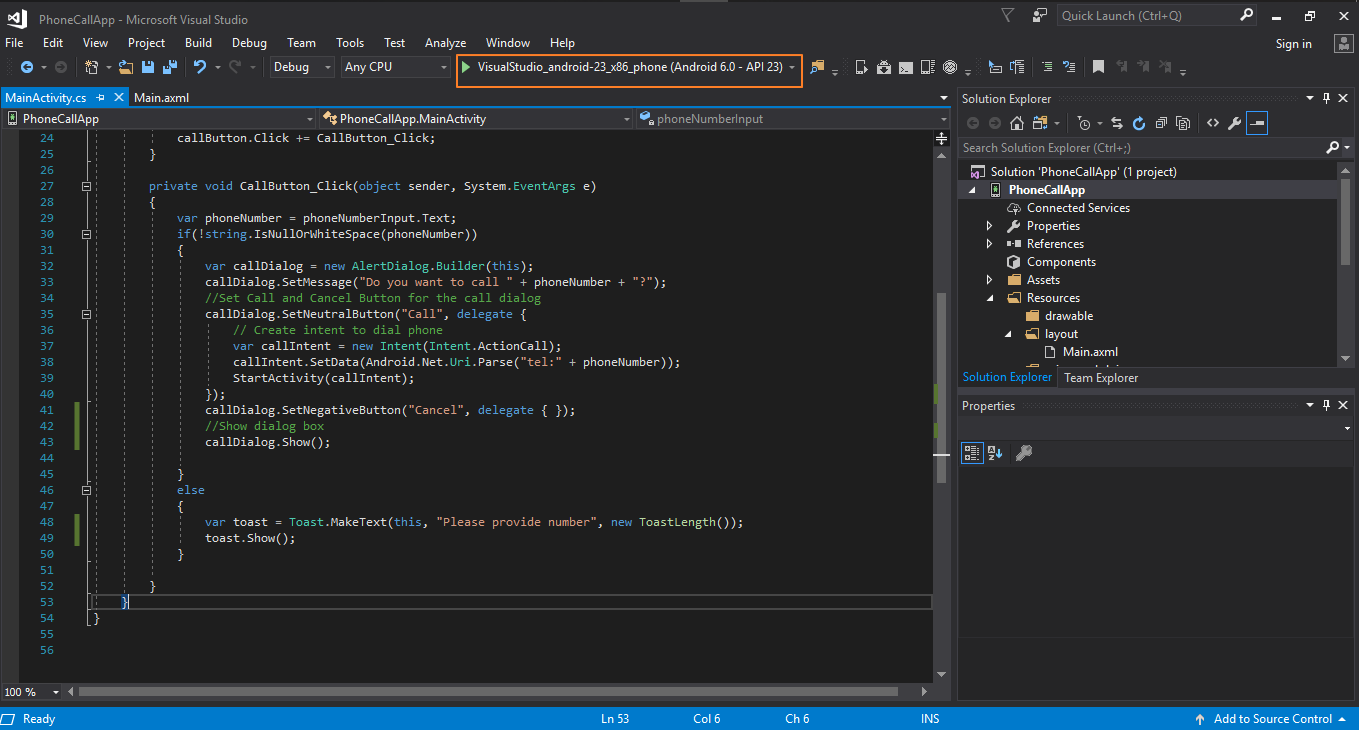
*});*

*callDialog.SetNegativeButton("Cancel", delegate { });*

1. Make some more changes to make the code look like in the image below:

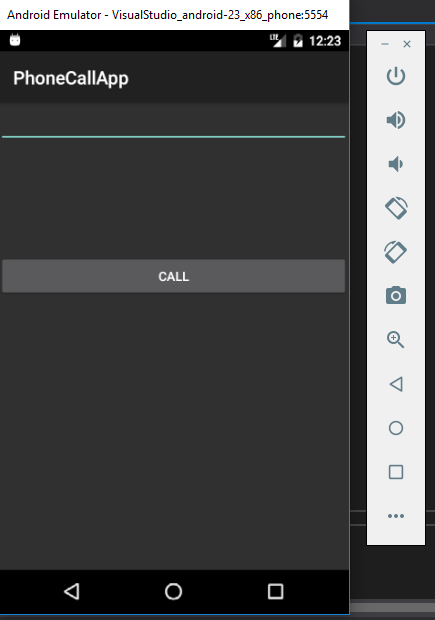


1. The code to handle user interaction is now complete, lets select the emulator from the top and run the application.



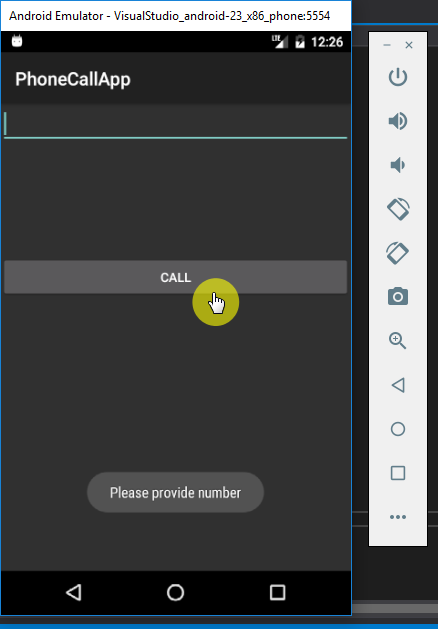
Running the project and deploying it on the emulator for the first time might take some time, be patient and let it complete the deployment.

1. Once the application is deployed, you should be able to see the application running on the emulator:



As we can see in the image above, the UI is what we created on Main.axml layout file.  
Let’s test the code we wrote to handle user interactions.

1. Click on “Call” Button without giving any number as input.

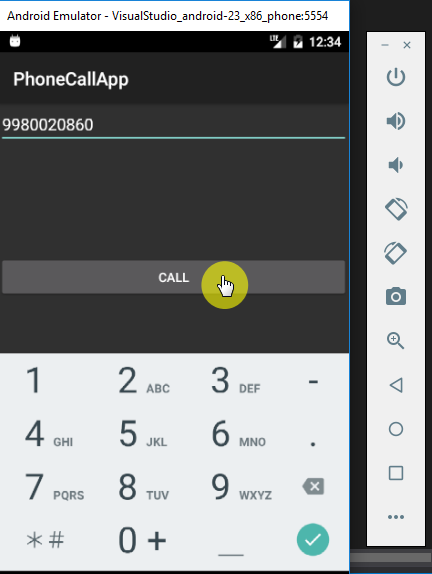


We’ll get a “toast” as shown in the image above. Because we had written a condition to check for Empty or whitespace input in the input number field.  
And if the there was no input provided, we had written below code to show a toast:

*var toast = Toast.MakeText(this, "Please provide number", new ToastLength());*

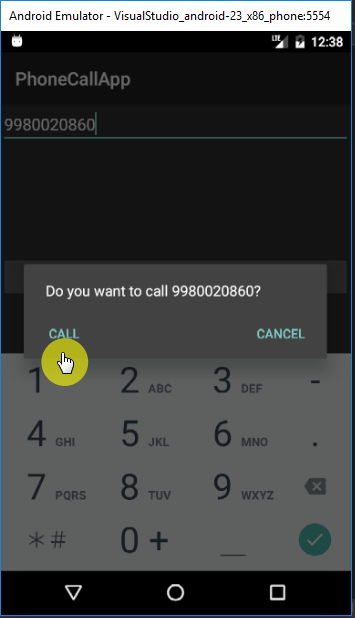
*toast.Show();*

1. Let’s enter some phone number in the text input field and then press call



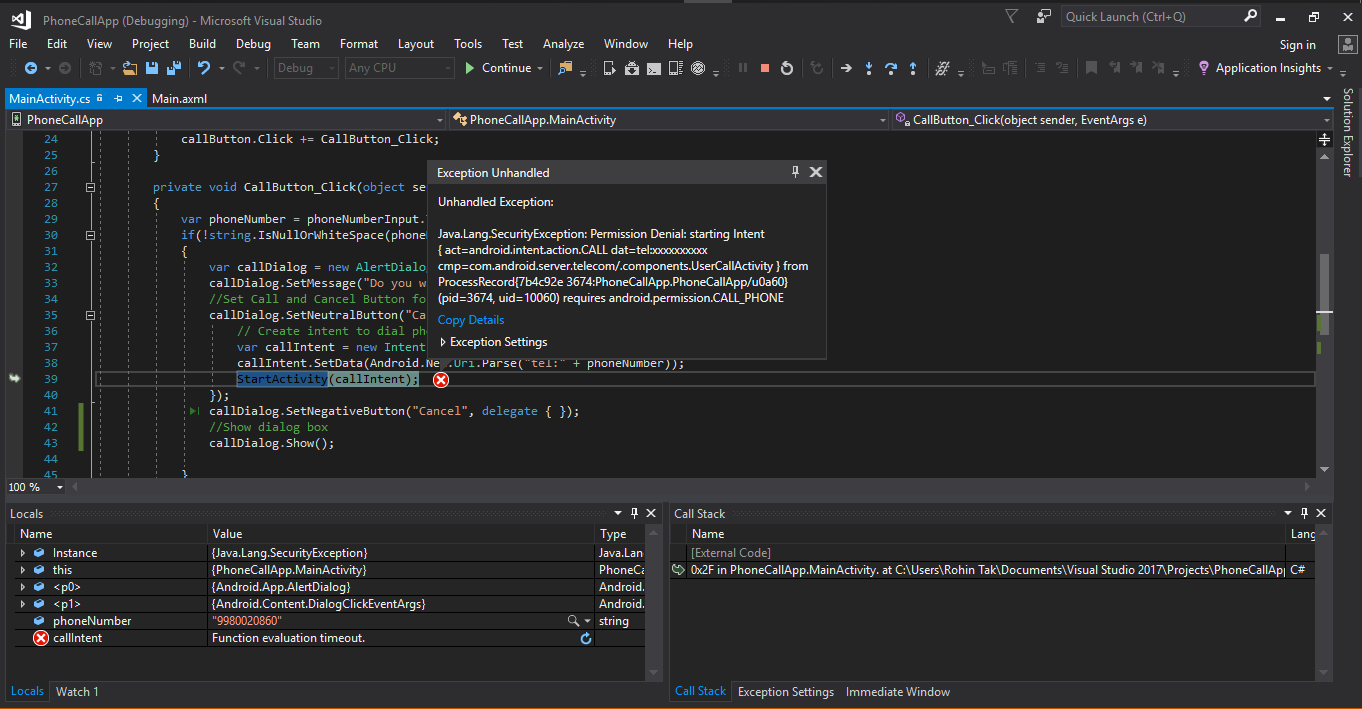
As per our code, we should get a dialog-box with a message saying “Do you want to call 9980020860?”

1. Clicking on Cancel should just close the dialog box. Let’s click on “Call”.



If everything goes fine, a call should be made to the above number. But that’s not what’ll happen once we click on Call button.

1. A java.Lang.SecurityException will be thrown.



The reason why we got this exception is that Android Application requires permissions to do certain operations and tasks.

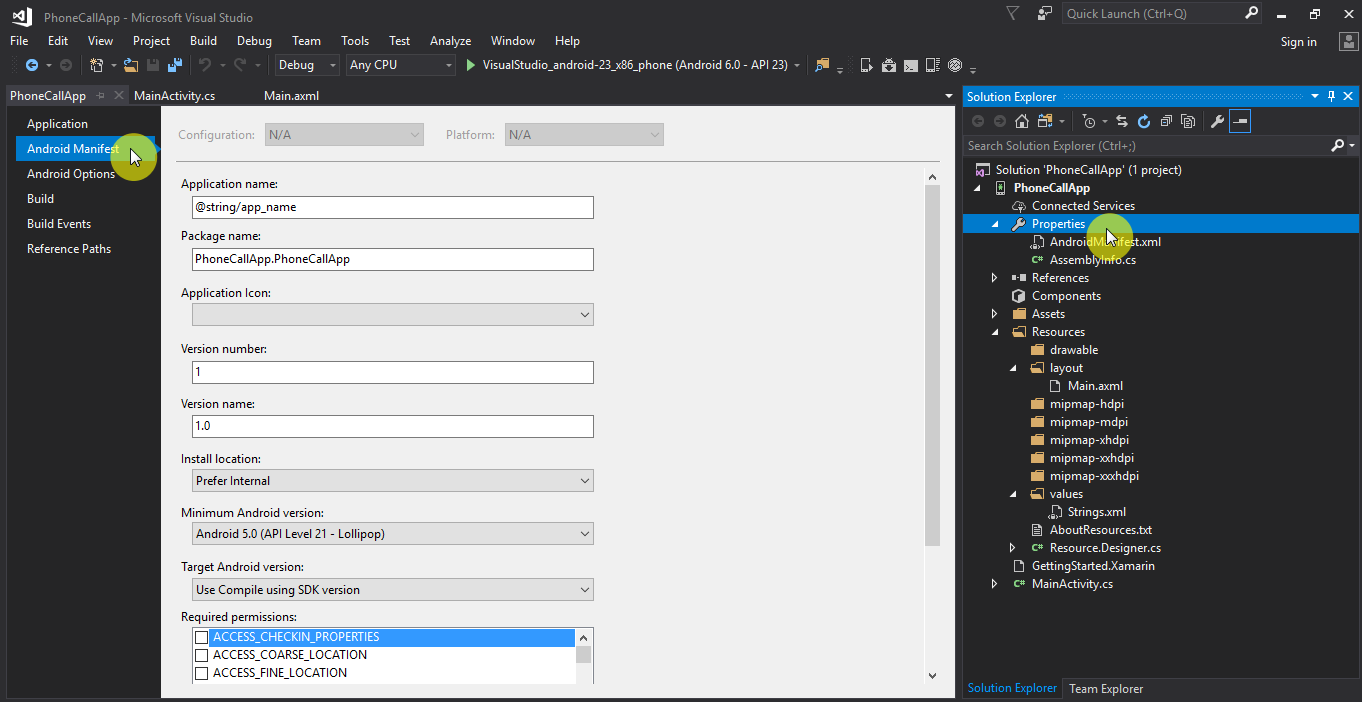
These permissions should be listed in the Android application code so that system knows what all permissions application requires before installing it.

These permissions are listed to user while installing and user allows such permissions to the applications, then only an app can perform such operations.  
So, the next thing we need to do is add permissions to our Android application.

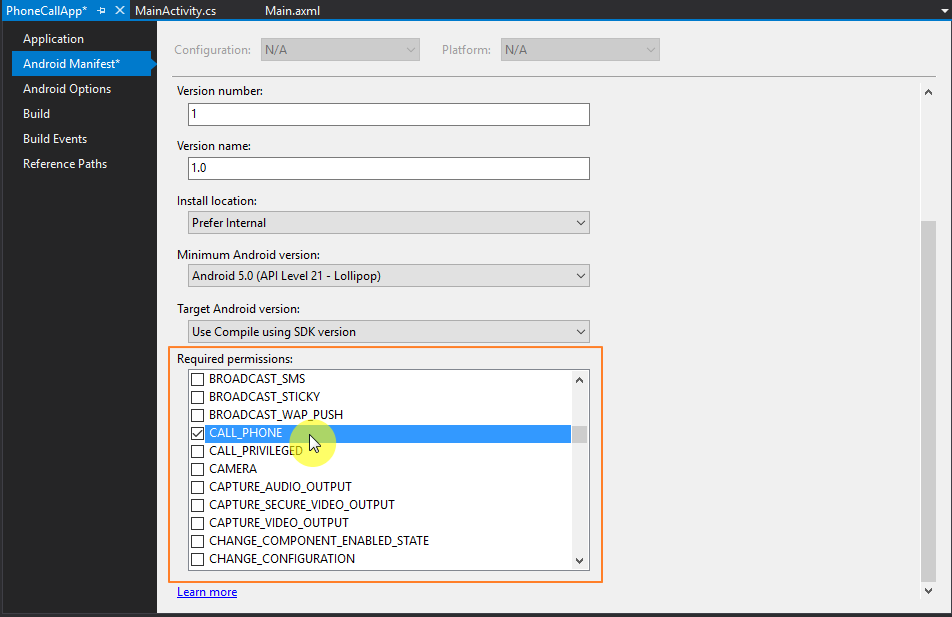
**Add Permissions to Android Manifest**

Our application needs only 1 permission as of now, that is to place a call. To modify or add permissions for the application we need to edit Android Manifest.  
To edit Android Manifest and give permission:

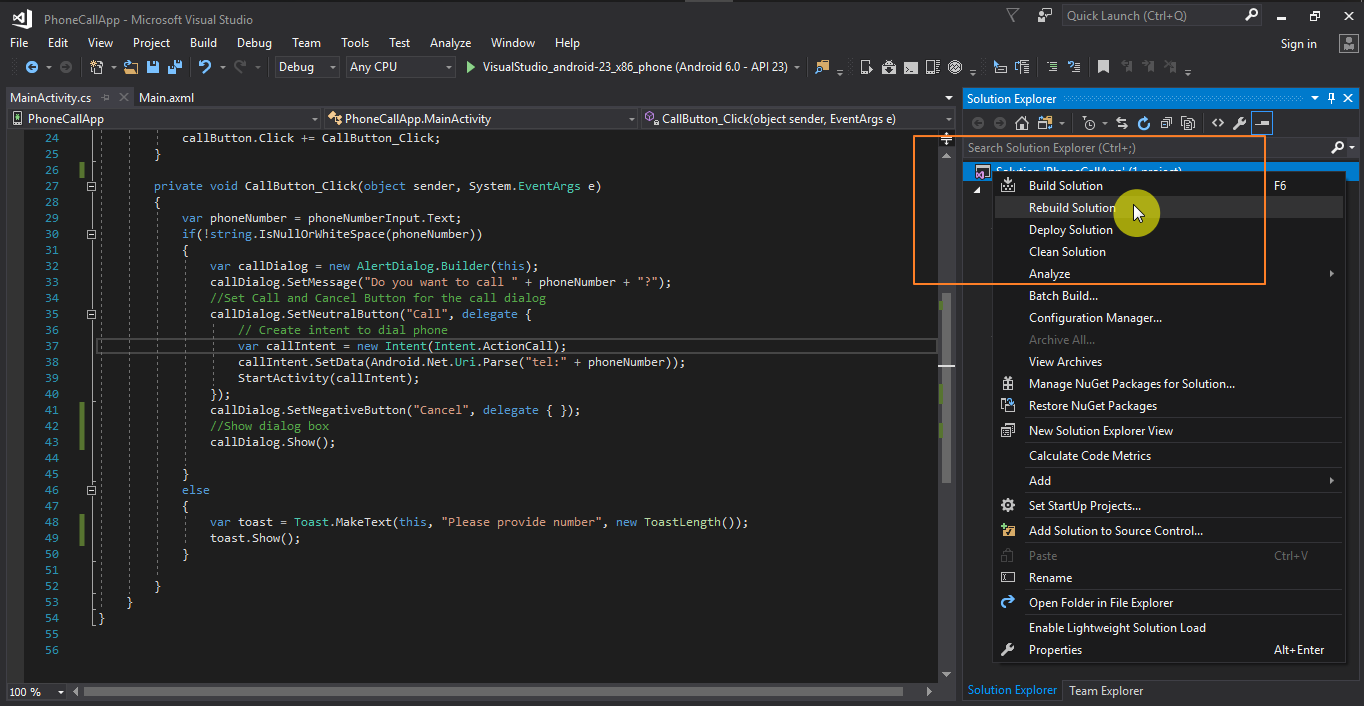
* Open Solution Explorer
* Double Click on Properties under the project
* This should open a UI to edit project properties
* Now from the left side menu click on Android Manifest to open it



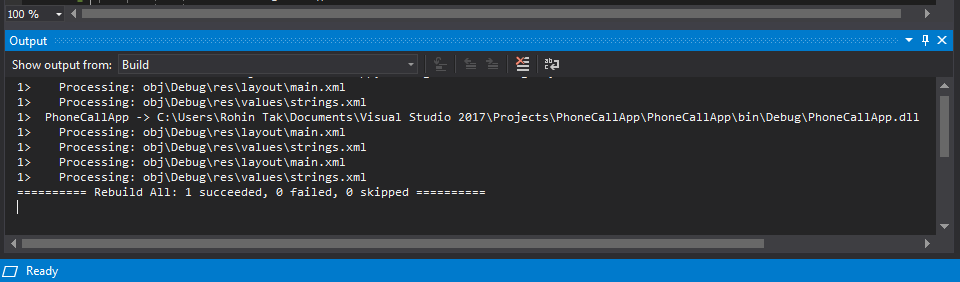
* In the Required Permissions section, scroll down and find CALL\_PHONE permissions and select it.



* Press Ctrl + Shift + S to save all the changes to the project
* Close the Properties window
* We are done adding permissions to application
* We need to build the solution now, so the resulting installation file has all the changes we made
* Rebuild the project right click on Solution -> Rebuild Solution



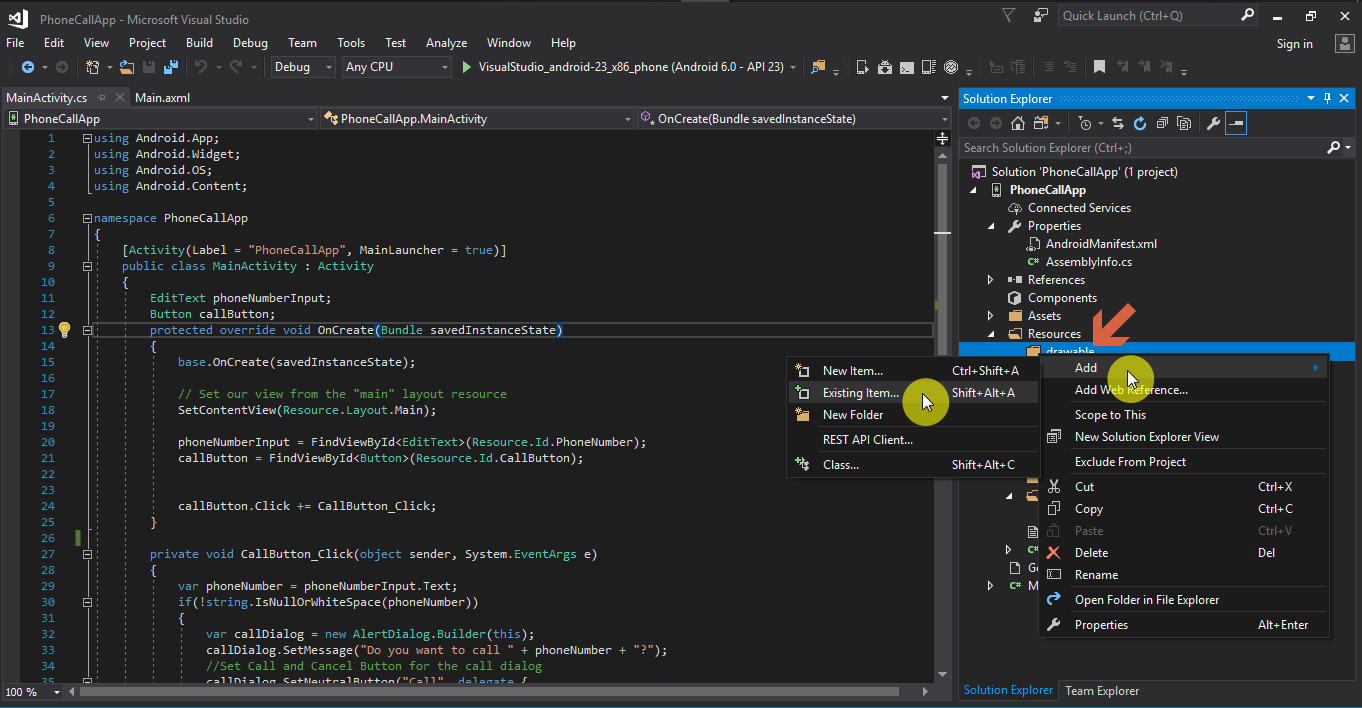
* If everything is fine, we should be able to see in the output window as Rebuild Succeeded, if you get errors, go to previous steps and compare the code and rebuild



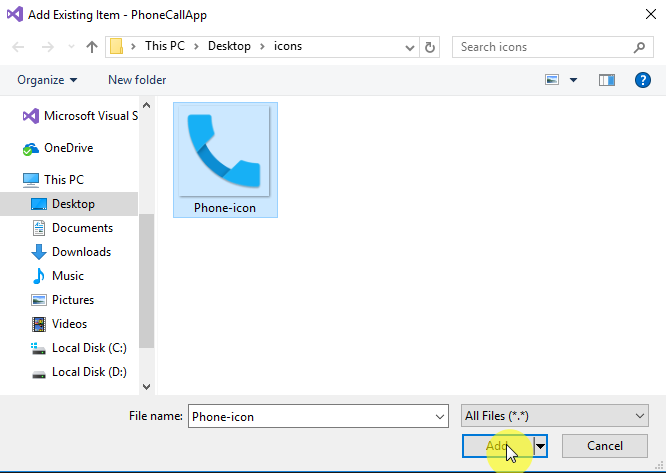
**Add Icon for Android App**

App permissions are set and it’s ready to run, let’s add an icon for our app

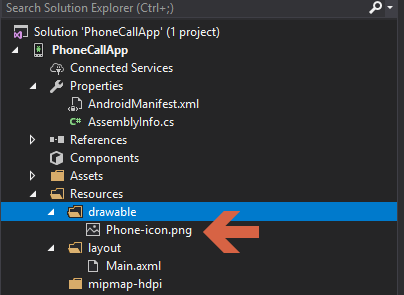
* Download an icon file that you like and suits best for our phone call app.
* Go to Solution Explorer and add the downloaded file to the “drawable” folder under Resources
* Right click on drawable -> Add -> Existing Item



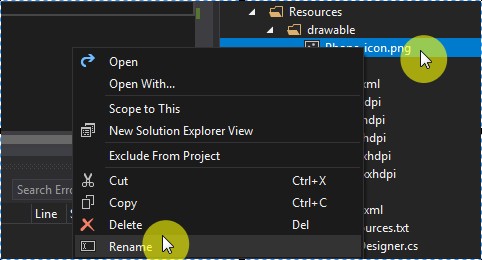
* File Explorer window will open.   
  Navigate to the icon file location, select the icon file and click add.

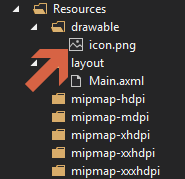


* The icon should now be added to the drawable folder of the project

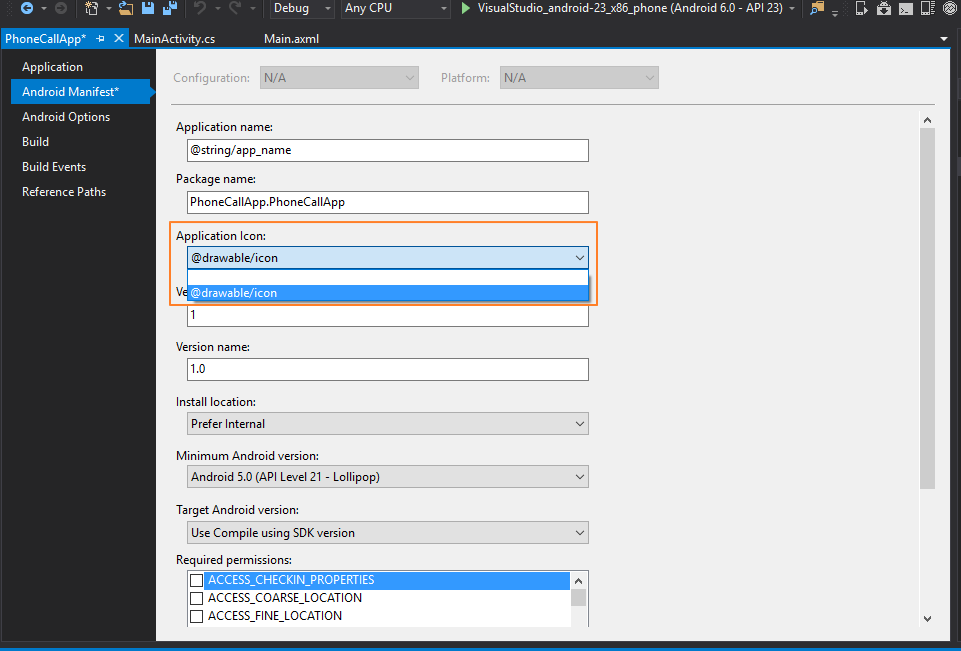


* Rename the icon file to icon.png by right clicking on the file and then click rename

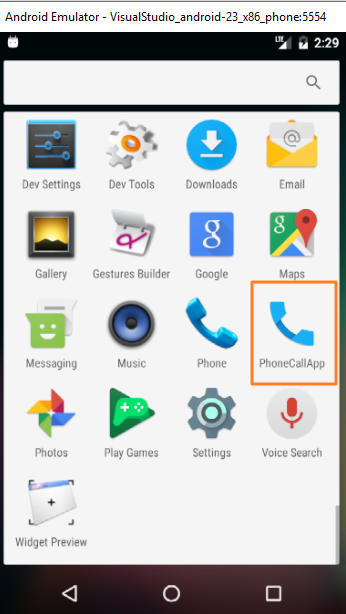




* After renaming the file, rebuild the project like we did in the previous steps.
* Once rebuild is done successfully, let’s add the icon to application’s Manifest file
* Double click on the Properties from solution explorer and open Manifest
* And chose @drawable/icon from the Application Icon dropdown



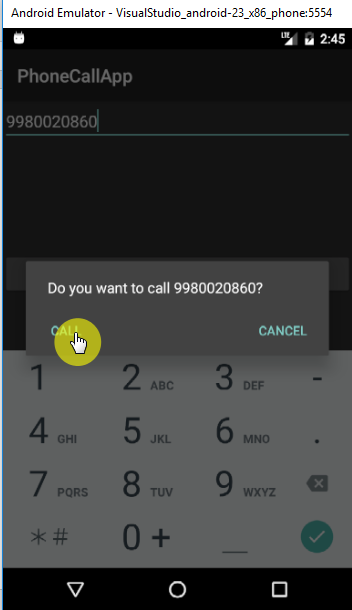
* Do Ctrl + Shift + S to “Save All” and rebuild the solution to make sure everything works fine.
* Now let’s run the application on emulator.
* If we go to the app drawer and scroll down to the app name, we can see app icon we just added, now showing there:



* Congratulations, you’ve successfully added an icon for the new Android application.
* Now that we have added the permissions and icon to the Manifest, It is time to test the main functionality of our app, that is “making a call”.

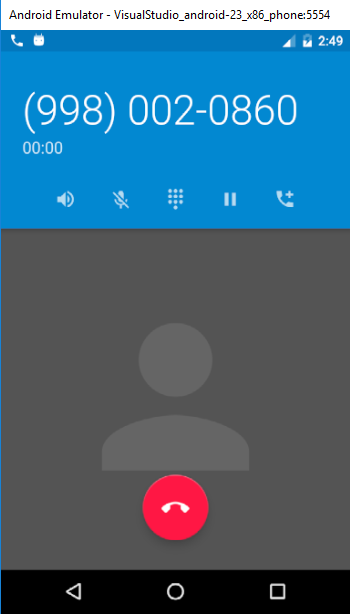
**Test User Interaction**

Click on the app on Android Emulator and run it. Repeat the previous steps of testing the application and at the end press call button to make a call.



This time application has all the permission and we have written the code to handle call button interaction and we are creating a callIntent in the MainActivity.cs to make a call.

So the call should be placed on click of the “Call” button and we should get a screen like below:



Awesome! You just created your first working Android application using Xamarin and C# on Visual Studio.

Now that we have done the difficult part, let’s understand some fundamentals of an Android Application we just developed and see how it all comes together.

**Application Fundamentals**

There are many topics that can be covered while explaining Android application fundamentals. But for the scope of this book we’ll try to understand the most important ones that we used in the development of our PhoneCallApp.

1. **Android APIs**:

Android has different API levels for different versions of Android. These API levels basically tells which version of Android libraries our code uses and which all versions of Android OS our app is compatible with.

There are different configurations to be specified while developing an Android application. These configurations include:

* Target Framework
* Minimum Android Version
* Target Android Version

You’ll read about these configurations in more details.

1. **Resources:**

Resource is a general word that encapsulates many features used in Android to make a better Android application. An Android application uses many resources like

* the icon we used
* the layout file that makes the UI for the user,
* String files to store strings for application localization/internationalization etc.

1. **Activities:**

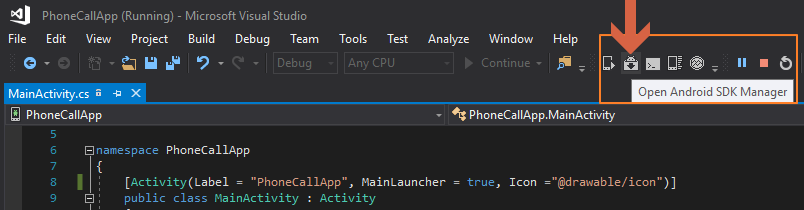
Activities are the main building block of applications in Android. Every UI element and its interactions are connected to an activity. Whenever we click on a button and open a new page, a new activity is called and control gets transferred.  
An Activity in Android can have different state based on the current operation being performed.

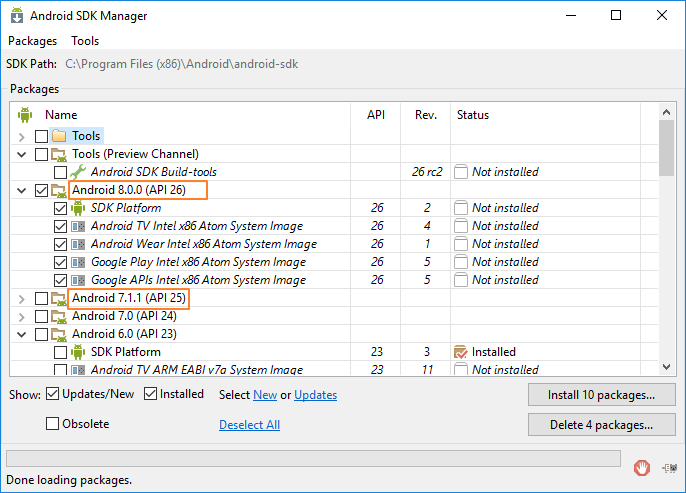
you’ll learn more about Activities in detail in coming topics.

**Android APIs**

Android APIs are known with an API level, e.g. API level 23.

An API level represents a specific Android release. If you open Android SDK Manager in Visual Studio:





Each API level is specific to an Android release. Any release of Android is known as multiple names:

* The API level, such as API level 23
* The Android version, such as Android 6.0
* A code name, such as Marshmallow

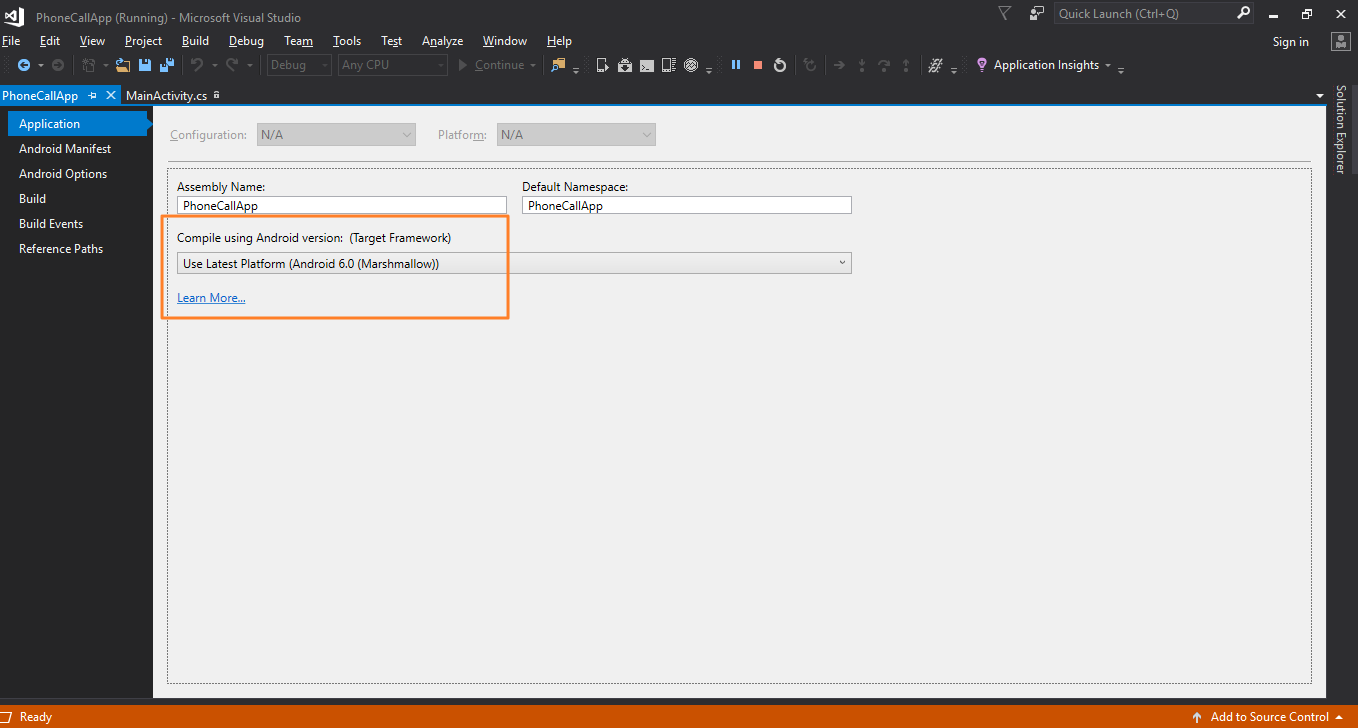
So, we can say that API is an integer value, a number to identify the release.

Because with each release this API level changes, and users upgrade their Android versions as they get released.  
An Android app should be able to run on different APIs and should be compatible with previous versions of releases, so that old devices can run those applications as well.  
And when user updates their OS version to a new one, existing apps don’t break on their phones.

To support multiple API levels, Android project property has configuration to define:

* **Target Framework**

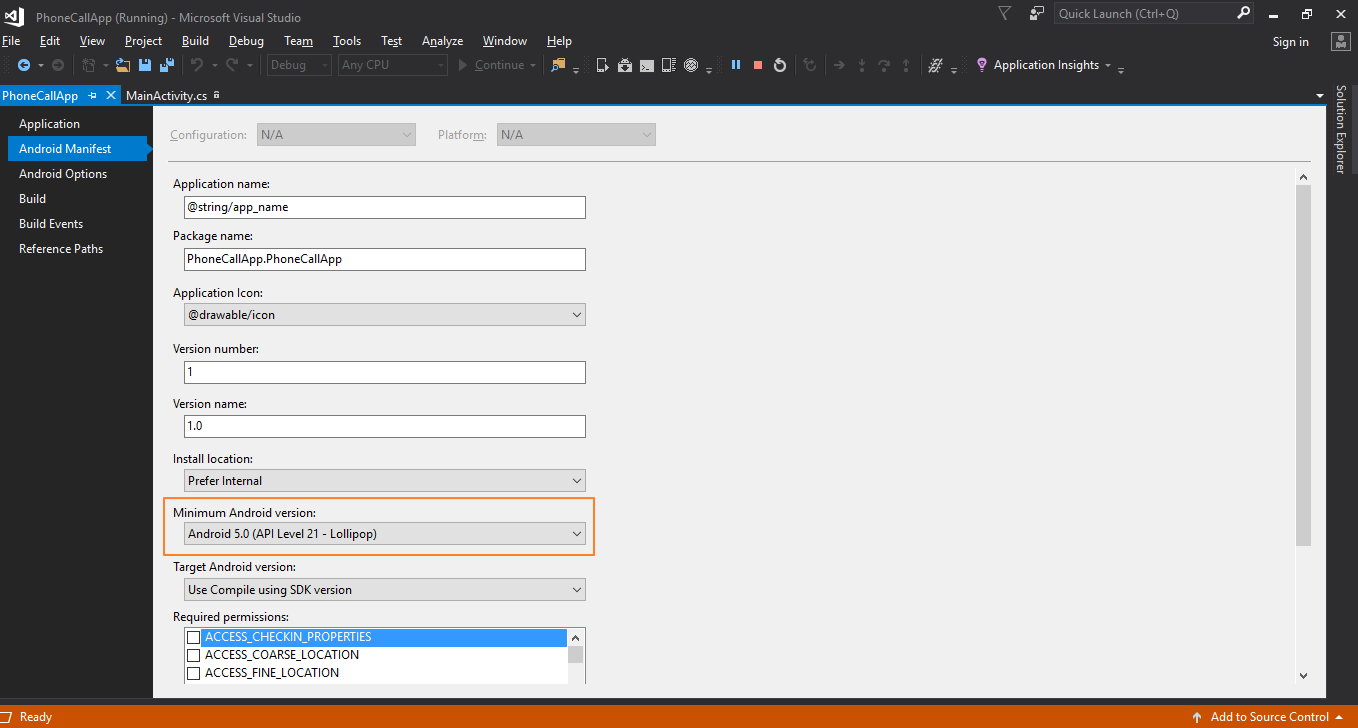
This setting can be found in under Application menu inside properties.  
This tells the Xamarin.Android to compile the project using specific API level libraries.  
While compiling/building the application, Xamarin.Android uses API level specified in this setting to load the libraries and build the application with.



* **Minimum Android version/API level**

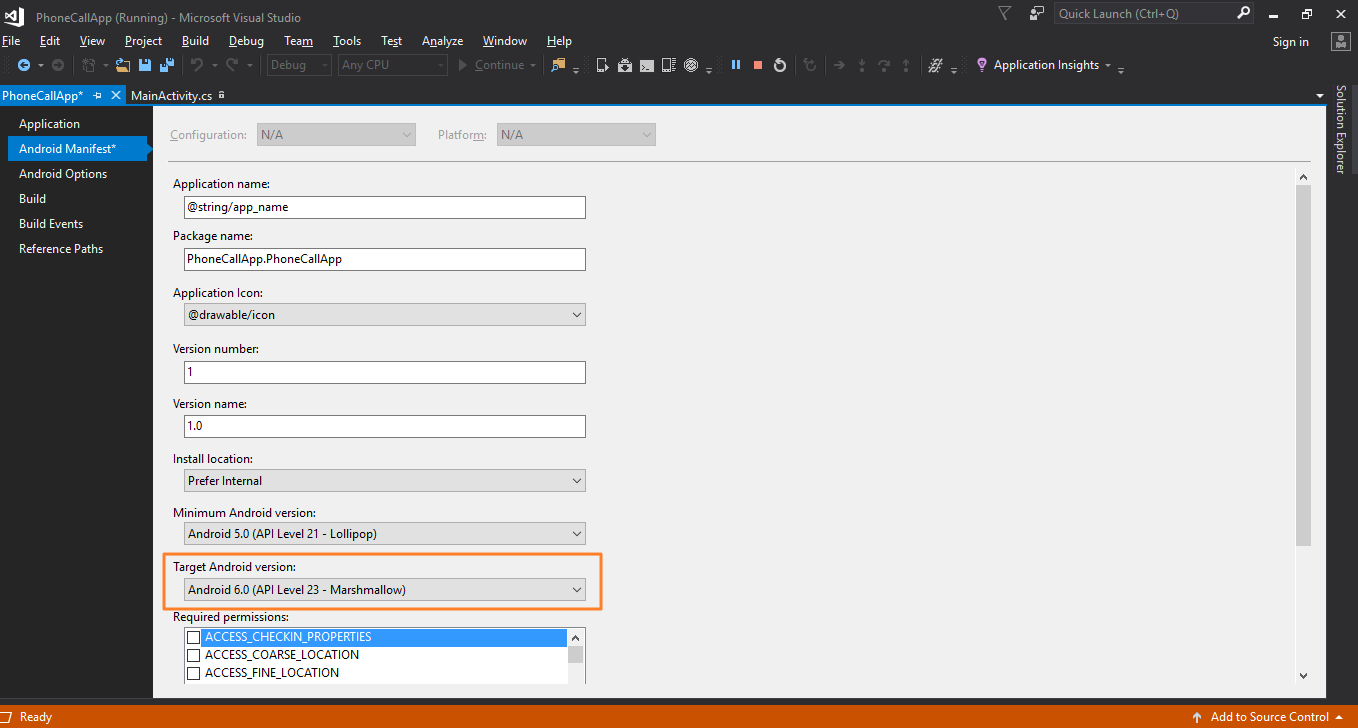
This is the minimum Android version that application can run on, this tells Android system if the app is supported on the specific OS version.  
Specifying a lower minimum version means your application can be installed on all the versions between minimum and target specified.   
But Be careful, because even if the application compiles and gets installed on a lower version of Android, it does not mean it would run successfully as well.

There might be some higher-level APIs that your application is using and could not be run on an older version.  
This setting can be found under Android Manifest inside properties.



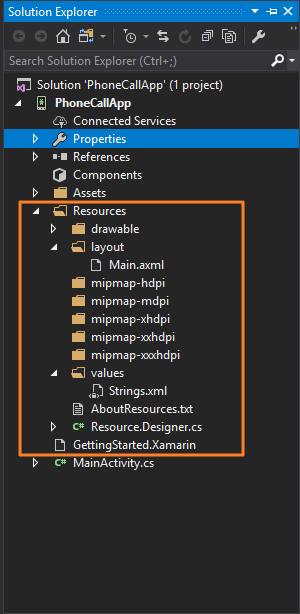
* **Target Android version/API level**

This is the OS version the app is developed to run on. Android uses this configuration to check whether it needs to enable any compatibility behaviors while running the application.  
This configuration can also be found in Android Manifest inside project’s properties.



**Resources**

When we created new Xamarin.Android application project, a folder named Resource was created in the solution explorer.



Let’s analyze the folder structure of our Resource folder in detail.

For an Android application structure, almost everything other than the actual code is a resource.

**A resource can be any of the following but not limited to**

* **Images** 
  + Any image or icon used in the application
  + They go under drawable folder
* **Application View**
  + View files for the application, i.e. Main.axml file that we created
  + Goes under layout folder
* **Strings**
  + These are text strings that are used across the application
  + Like some “Call” text on the text button
  + It helps keep consistency throughout the application
  + Goes under values folder

**Resources we used in the application:**

The main files that we used in our application in the Resources folder:

* **Icon.png**

The icon for the application we downloaded and added

* **Main.axml**

The default user interface layout file for our application. We only edited this file in the designer, but you can also go ahead and open the file in XML view and try to understand the xml tags used for UI elements.

* **Resource.designer.cs**

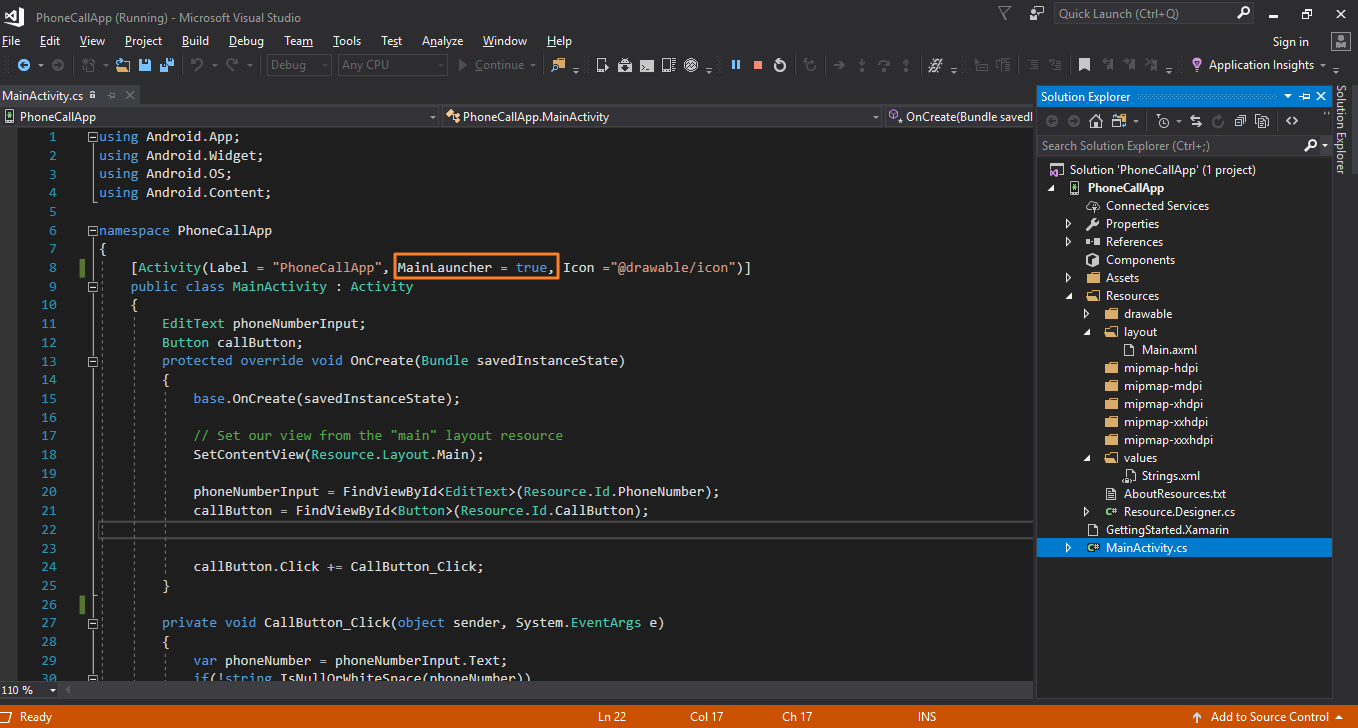
This file is automatically generated and maintained by Xamarin.Android and holds the unique ID's assigned to each resource. It is automatically created by the Xamarin.Android tools and will be regenerated from time to time.  
This is why to access certain resources in our C# code we used below code:

*phoneNumberInput = FindViewById<EditText>(Resource.Id.PhoneNumber);*

Notice the “Resource.Id.PhoneNumber” written, this information is basically stored in Resource.designer.cs file, all unique ID’s assigned to resources are stored here.

**Understanding Activities**

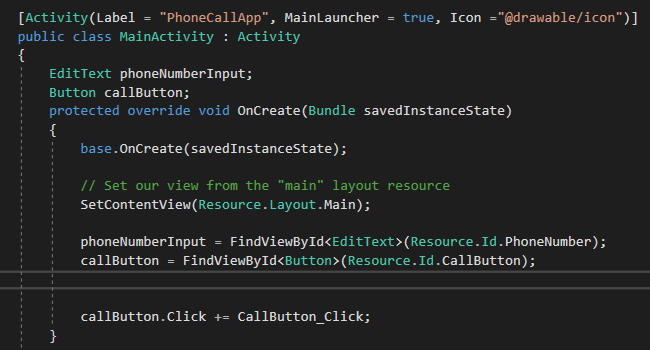
Activities are something very specific to Android application development.   
Usually in other application we have an entry point or a main method as an entry point to start the application.  
But in Android, the same purpose is fulfilled by Activities. Android application can be started from any Activity that is specified as a starting activity for the application using MainLauncher.



**Activity Class**

The Activity class contains the code that controls the user interface. Activity class is basically responsible for creating the UI and handling user interactions such as button click or touch.

Now let’s take example of our PhoneCallApp application.  
We have only one Activity in our project, and that is MainActivity.cs class.  
It is the main entry point for the OS into this application since we have set it as main launcher.



If we look closely, MainActivity class inherits Activity class, i.e. it is child of Activity class.  
That means now MainActivity is also an Activity.  
Also it is important to note that we have an Activity attribute defined above the MainActivity class which specifies the Label and MainLaucher property as well.  
This Attribute tells Android that MainActivity class is part of the application and is managed by its Manifest.

By Inheriting Activity class MainActivity gets access to methods of Activity class that provide developers the ability to perform certain actions on different states of MainActivity like:

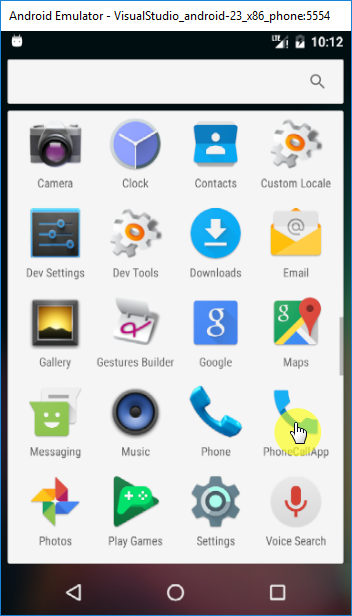
* When activity is created
* When activity is paused
* When activity is resumed etc.

When developing an application and writing code for an Activity, as discussed earlier some methods are provided by Activity class that we can use to perform some operations based on different states of an Activity.

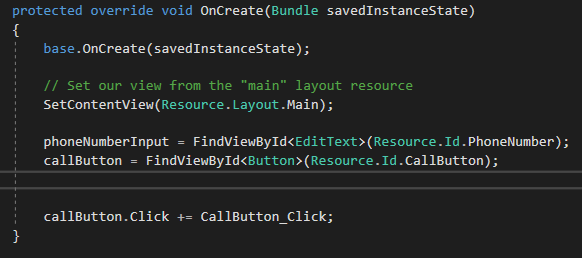
**Methods in Activity Class**

* **OnCreate():**

When user clicks on the app icon to start the application, this method is called. This method is used to perform some initial setup that might be required for the activity e.g. Creating views, initializing variables etc.



Let’s have a look at our application code where we used OnCreate method to do some initialization and setup.



Things we are doing in our OnCreate method are:

* + Setting up Layout for the view
  + Initializing variables to get reference of TextInput and call button.
  + Binding Click event to Call Button
* **OnStart():**

This method is always called right after OnCreate() method by the system

* **OnResume():**

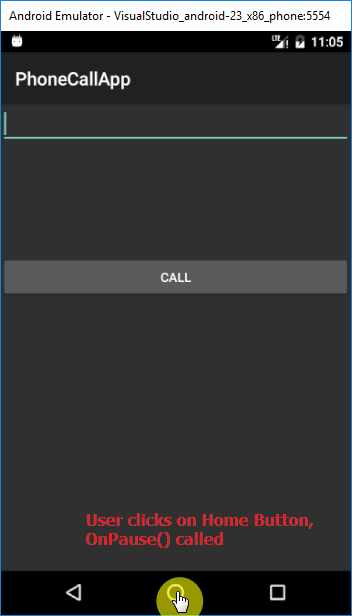
This method is called by the system when the application is up again and ready to interact with the user. OnResume is important because any operation that is done in OnPause should be un-done in OnResume, since it's the only lifecycle method that is guaranteed to execute after OnPause when bringing the activity back.

* **OnPause()**

This method is called when the system is about to put the activity into the background.

It is also an important method, because an activity should perform certain tasks like:

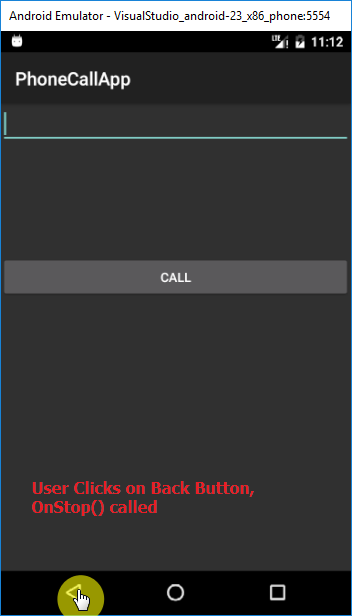
* + Saving unsaved changes
  + Freeing up resources like camera or other resource etc.



* **OnStop():**

This method is called when the activity is no longer visible to the user. This happens when one of the following happens:

* + Back button is pressed.
  + An existing activity is being opened and brought to the foreground.
  + A new activity is getting started and covering up the current activity.



* **OnRestart():**

If an activity was stopped and then it is started again, this method gets called.

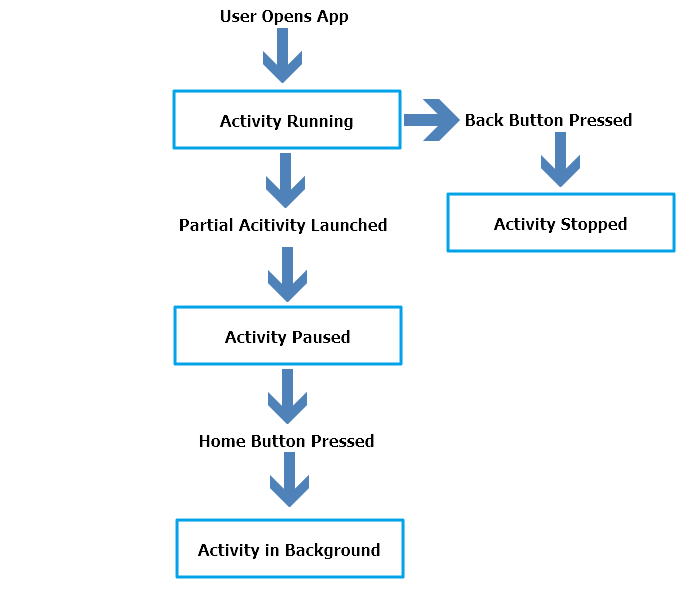
* **OnDestroy():**

This is the final method that is called on an Activity before it's destroyed and completely removed from memory. It is used to clean up resources that might cause memory misuse.

To understand more about the different activity states, let’s dive into Activity Lifecycle.

**Activity Lifecycle**

The activity life cycle is usually defined by a list of methods inside Activity class that provides us with ways to control the state of an activity. This allows developers to handle activities within Android application.  
Let’s have a look at different states of an Activity:



These states can be broken into 4 main groups as follows:

1. **Running:**

Activities are called active or running if they are in the foreground, also known as the top of the activity stack. This is known to be the highest priority activity in Android, and will only be killed by the OS in extreme situations, such as if the activity tries to use more memory than is available on the device since this could cause the application UI to become unresponsive.

1. **Paused:**

When a partial activity is called on top of currently running activity it is considered Paused. Paused activities are still alive, i.e. they maintain all state and member information, and remain in the activity stack. This is considered to be the second highest priority activity in Android and, will only be killed by the OS if killing this activity will satisfy the resource requirements needed to keep the Active/Running Activity stable and responsive.

1. **Stopped/Backgrounded:**

If an Activity is completely stopped or taken over by another activity then it is considered as stopped or in the background. Stopped activities still try to retain their state and member information for as long as possible, but stopped activities the lowest priority of the three states.

1. **Restarted/Resumed:**

If the user navigates back to the activity from another activity or by pressing app switcher icon, it must be resumed if paused or restarted, restored to its previously saved state if stopped, and then displayed to the user.

These categories are basic explanation of different states of an activity in the activity lifecycle.

**Deploying Application on Mobile Device**

Till now we have tested our application on Android Virtual Device (Android Emulator).  
But it’s always a good practice to test application on a physical device. So let’s learn how to setup an actual Android device for testing an application.

Images shown in this topic are taken using an Android device running Lollipop, your device settings may differ depending on your device version.

Steps to setup device for debugging:

1. **Enable Debugging on the Device**

We would need to enable debugging on the device. By default, it will not be possible to debug applications on an Android device.

1. **Install USB Drivers**

On our Windows computers, we would need to install USB drivers for our device

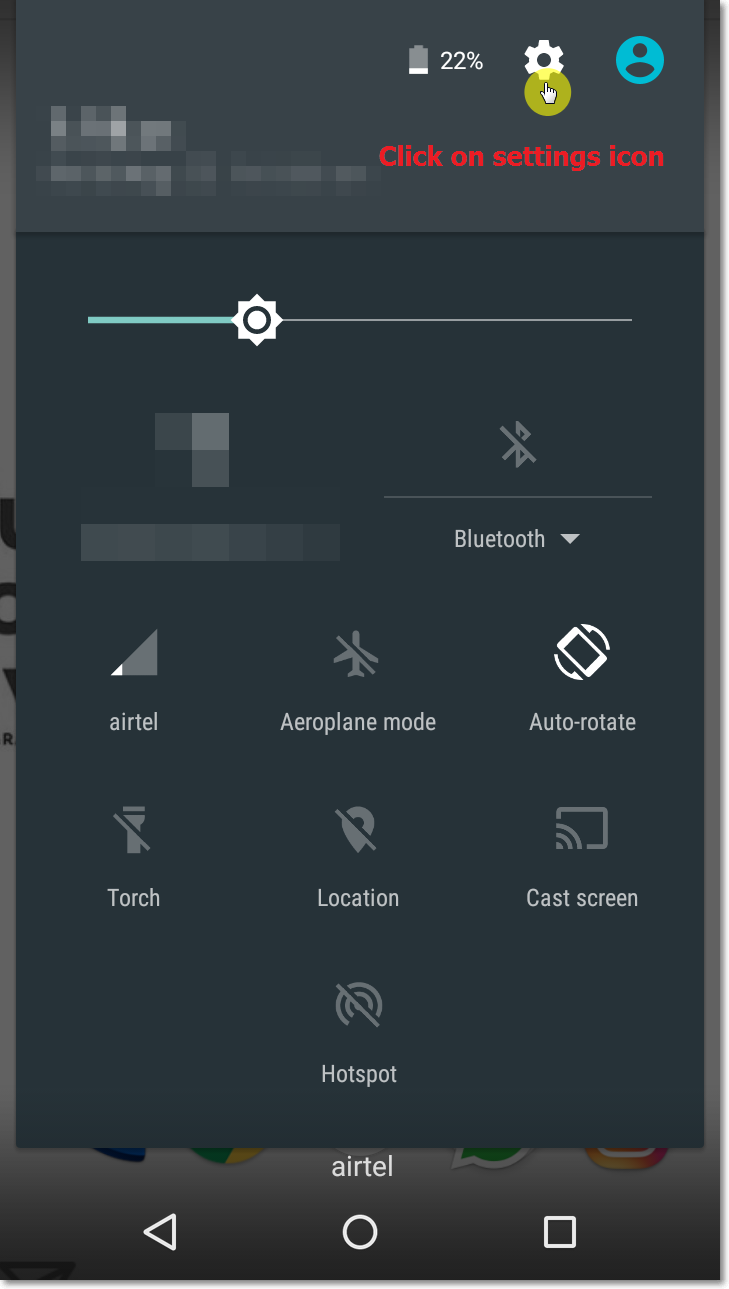
1. **Connect the Device to the Computer**

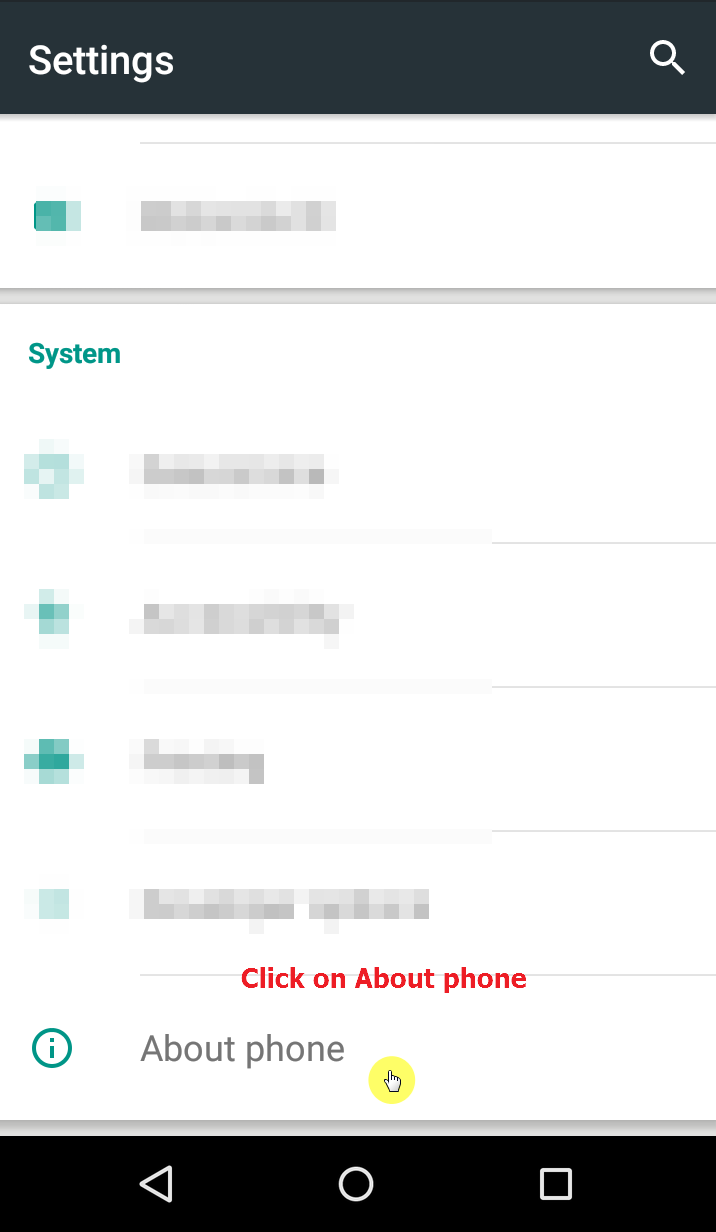
The final step involves connecting the device to the computer with a USB cable

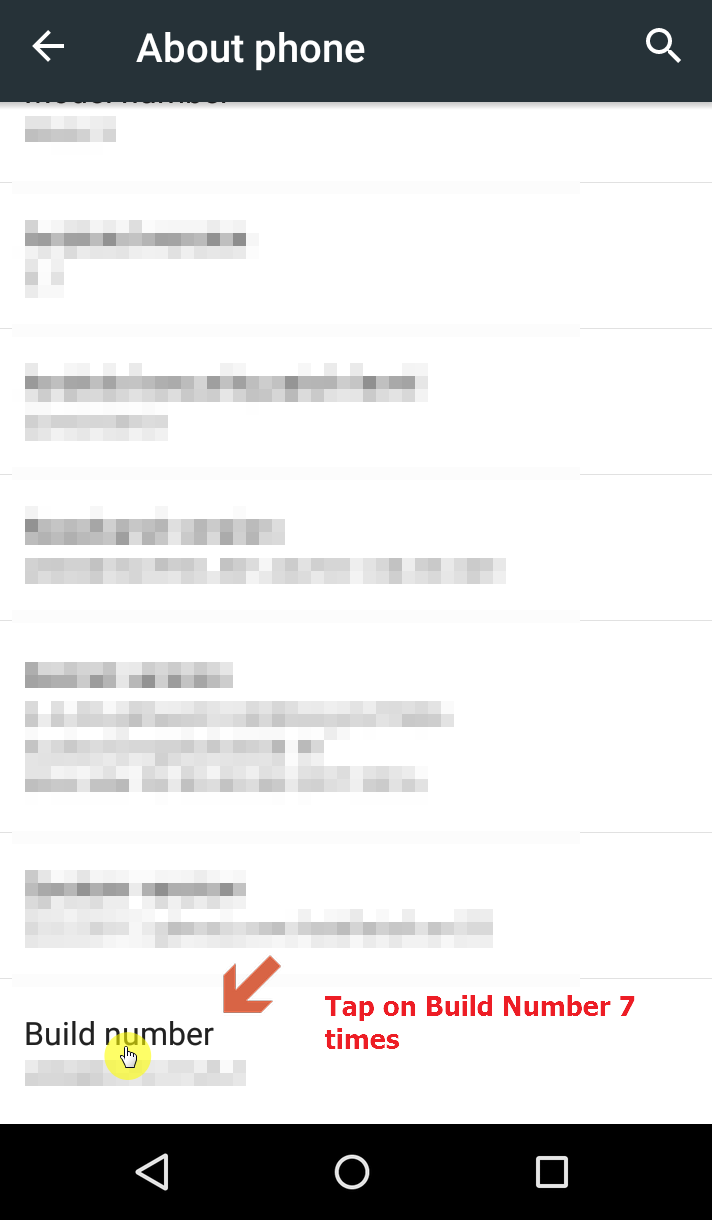
**Enable Debugging on the Device**

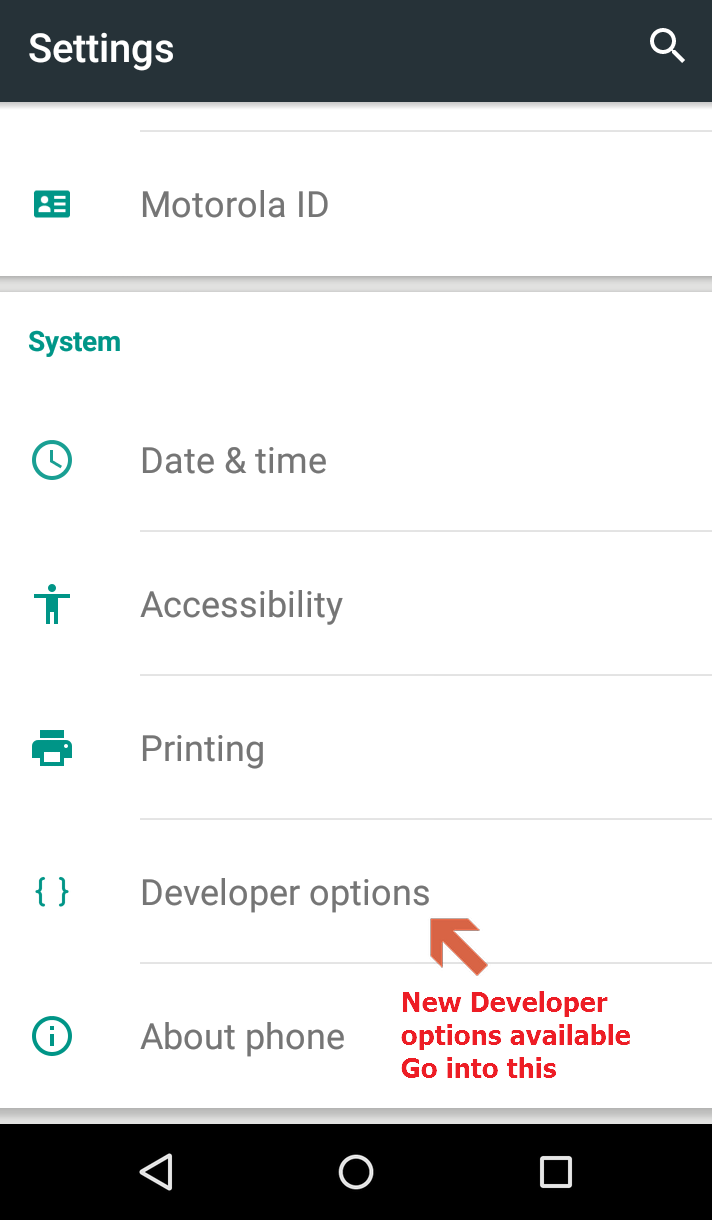
To enable debugging on the device we need to perform below steps (Follow images shown on next page for reference):

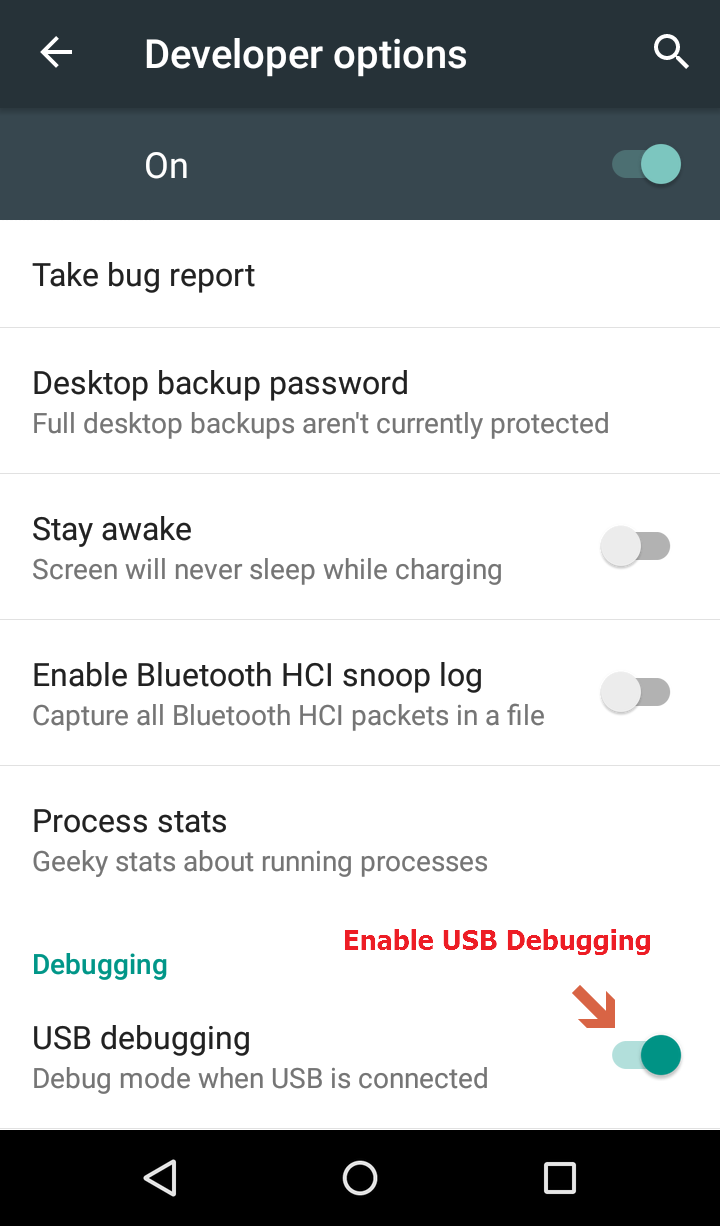
* + Click on Settings Icon from the notification bar.
  + Open Settings
  + Scroll down to the end
  + Click on About Phone
  + Scroll down to Kernel version
  + Tap on Kernel version for 7 times until it says “you are a developer”
  + Go Back to the Settings menu and scroll down till the end
  + You should be able to see a new menu now for Developer options just above About phone
  + Click on Developer options
  + Find option to enable USB Debugging
  + Enable USB Debugging











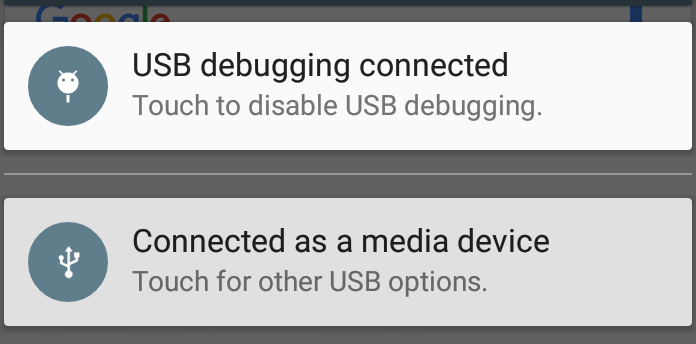
**Install USB Drivers**

For different devices, different drivers might be needed to be installed for the computer to recognize the device. Please make sure all the device drivers are properly installed and computer can recognize your device properly.  
If you are downloading the device driver and want to install them manually on the computer, follow below steps for Windows 7:

* Connect your device to the computer with a USB cable.
* Right-click on the Computer from your desktop or Windows Explorer, and select Manage.
* Select Devices in the left pane.
* Locate and expand Other Devices in the right pane.
* Right-click the device name and select Update Driver Software.
* This will launch the Hardware Update Wizard.
* Select Browse my computer for driver software and click Next.
* Click Browse and locate the USB driver folder.
* Click Next to install the driver.

**Connect Device to Computer**

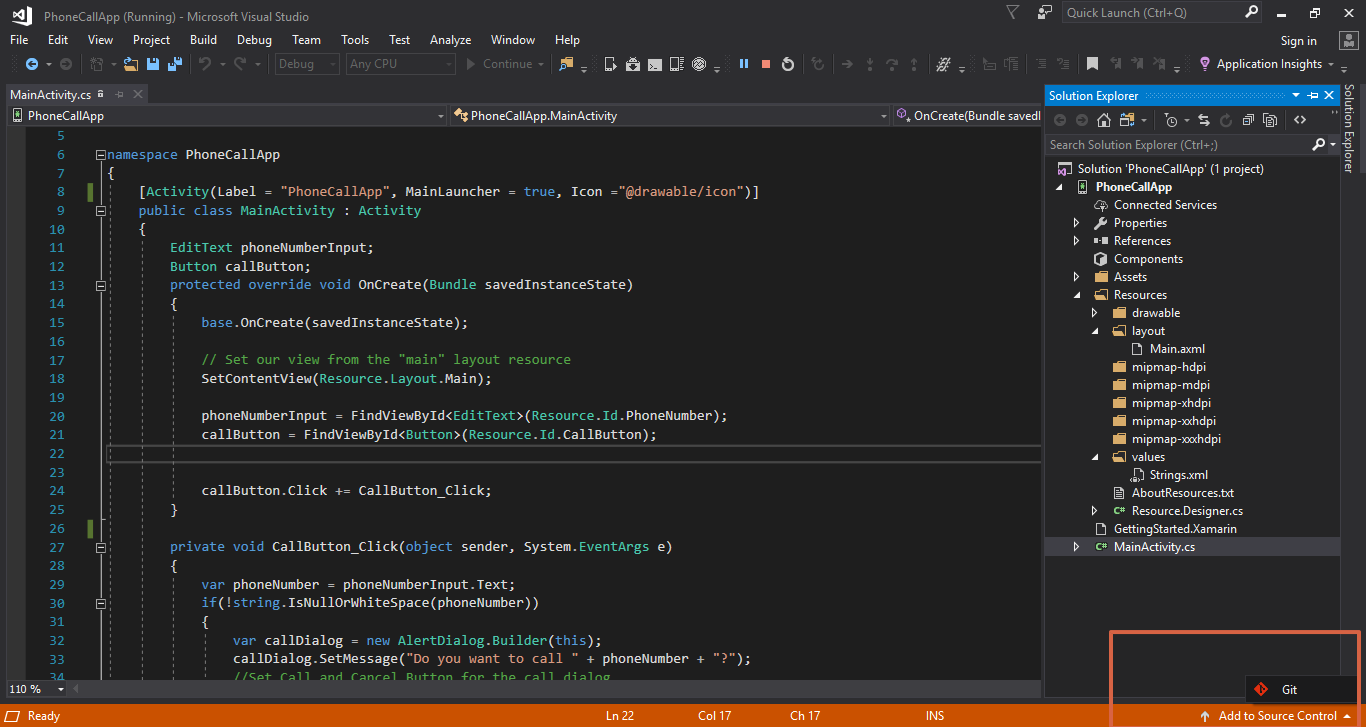
Connect the device with a USB cable to the computer and adb (android debug bridge) should be able to communicate with the device and you should see a notification on the device saying “USB debugging connected”as shown below:

  
  
Now you can go to visual studio, select your device listed in the running device list and run application. This will install application on your device and run it.

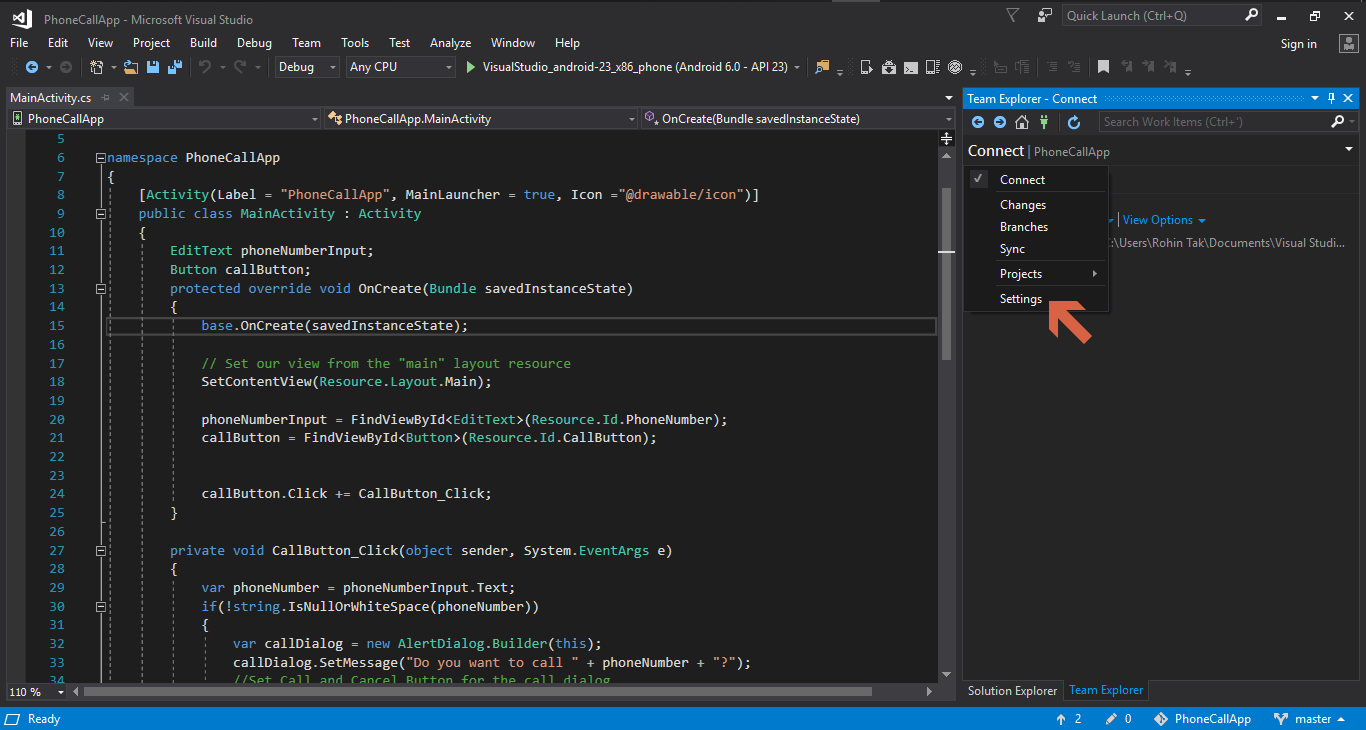
**Push Code to Git Repo**

Application development is done. Let’s save our code into Git repository so we can access the code from anywhere.

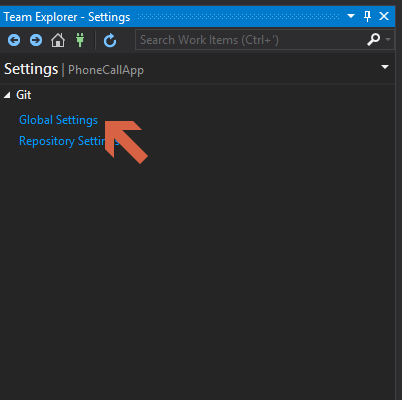
1. In Visual Studio at the bottom right corner click on Add to Source control and then Git



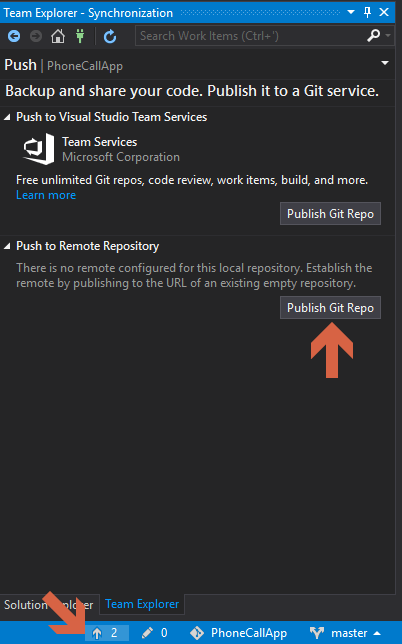
1. Click on Connect -> Settings



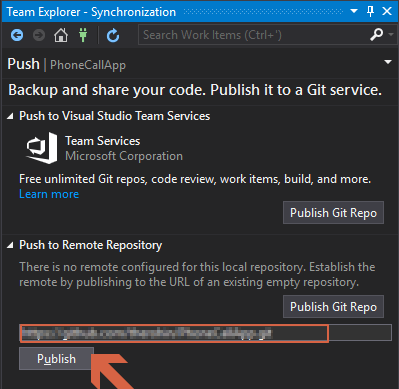
1. Click on Global Settings:



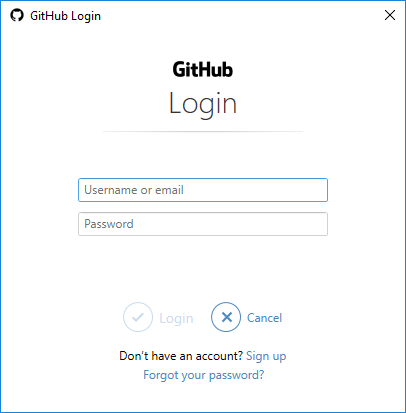
1. Enter your github account username and email and click Update
2. Click on the up arrow icon (push icon) at the bottom of the team explorer
3. Then Click on Publish Git Repo under Push to Remote Repository
4. Notice that it says there is no remote repository configured for this local repository, that is because we haven’t connected our remote github repository to our local project.



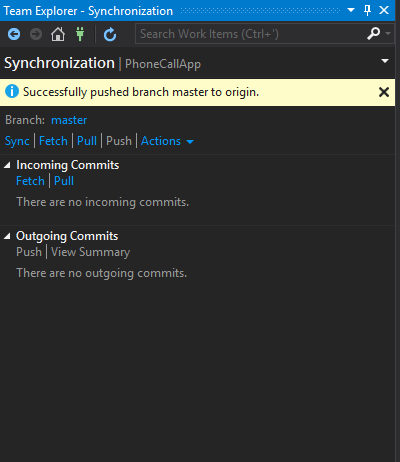
1. Let’s login to your GitHub account and create an empty git repository for our project as learned in Chapter 2 and copy that URL on the textbox as shown in the next image.



1. After clicking on Publish, A new window will open asking for GitHub credentials



1. Enter you github credentials to authenticate and click login.
2. After successful login, the code will be pushed to remote git repository and you should see a success message like below.



1. Congratulations, the code is now pushed to remote repository, same can also be checked by logging into github and going to the repository URl.

**Summary**

In this chapter, we learned to develop an Android application using Xamarin and Visual Studio.   
We also learned some detailed fundamentals of an Android application, Activities and their lifecycle. Running that application on Emulator as well as setting up an actual physical device to run the application and at the end pushing our code to a git repository.

In the next chapter, we’ll learn about implementing continuous testing using Xamarin Test Cloud.