Lab 01: Creating our First Android Application

Prerequisites

You will need a Windows PC with the Android SDK and Xamarin tools installed. We will be using the Android emulator to test the code we are building, so make sure to have a virtual device already configured and ready to run. See the Xamarin.Android setup documentation if you need help getting your environment setup.

Downloads

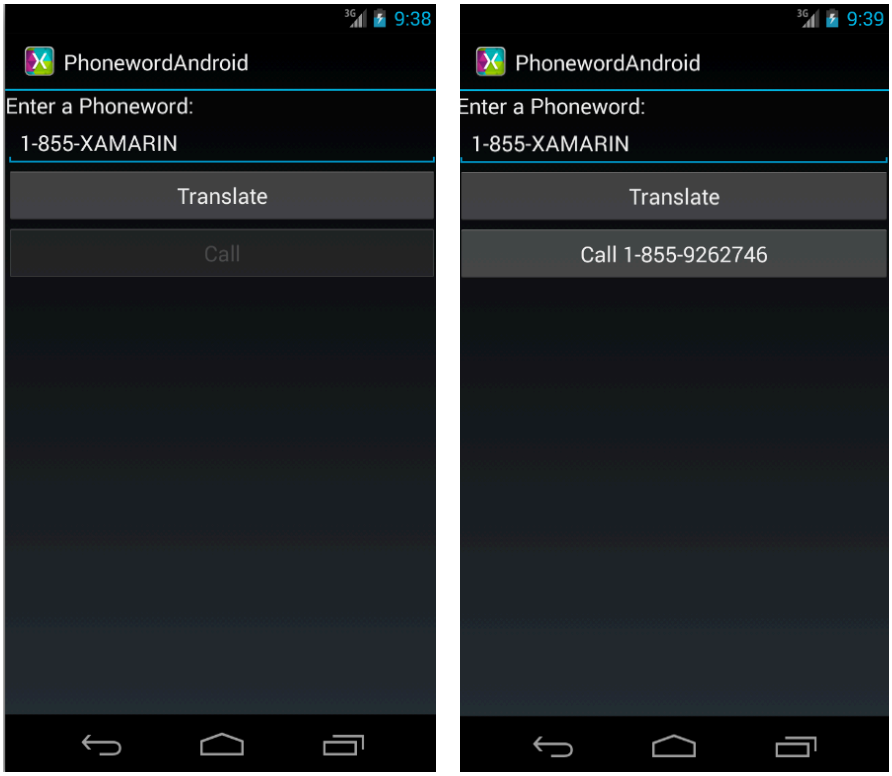
Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is **Lab 01 Resources**. Make sure you have this folder before you begin.

Lab Goals

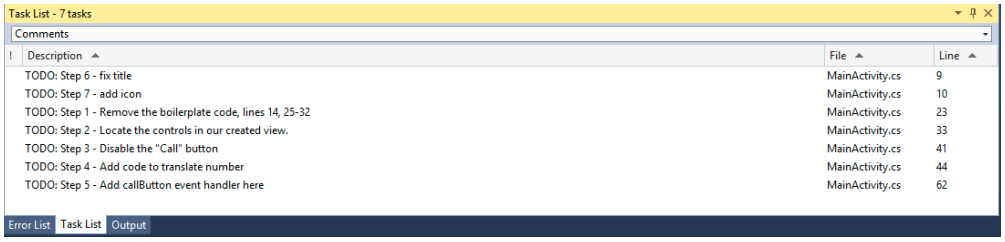
The goal of this lab will be to create our first **Xamarin.Android** application. During the process we will become familiar with the Xamarin tools within Visual Studio, and the various parts of our application. By completing this lab you will gain experience with the following tools:

- **Xamarin.Android Tools for Visual Studio** – Introduction to the Xamarin.Android Tools added to Visual Studio and how to use it to create **Xamarin.Android** applications.
- **Xamarin Studio Designer** – How to use Xamarin Studio's Designer to create your application's user interface (UI).
- **Android Emulator** – How to use the Android SDK emulator to test your application.

In addition, by completing this lab, you will be introduced to the core parts of a **Xamarin.Android** application through the creation of a simple application that translates a phone number with letters in it, i.e. `1.855.XAMARIN`, and then allows the user to dial the number as shown in the following screen shots:



The lab has been provided as a starter solution with most of the code already filled in for you – as you following along with the instructor you will make small changes for each step, either writing a little code or uncommenting a block of code. Most of these steps are clearly marked in the supplied solution with `// TODO:` comments. These comments are picked up by Visual Studio and shown in the Task List, which you can make visible through the **View > Task List** menu item. When the Task List is open, it will look like this:

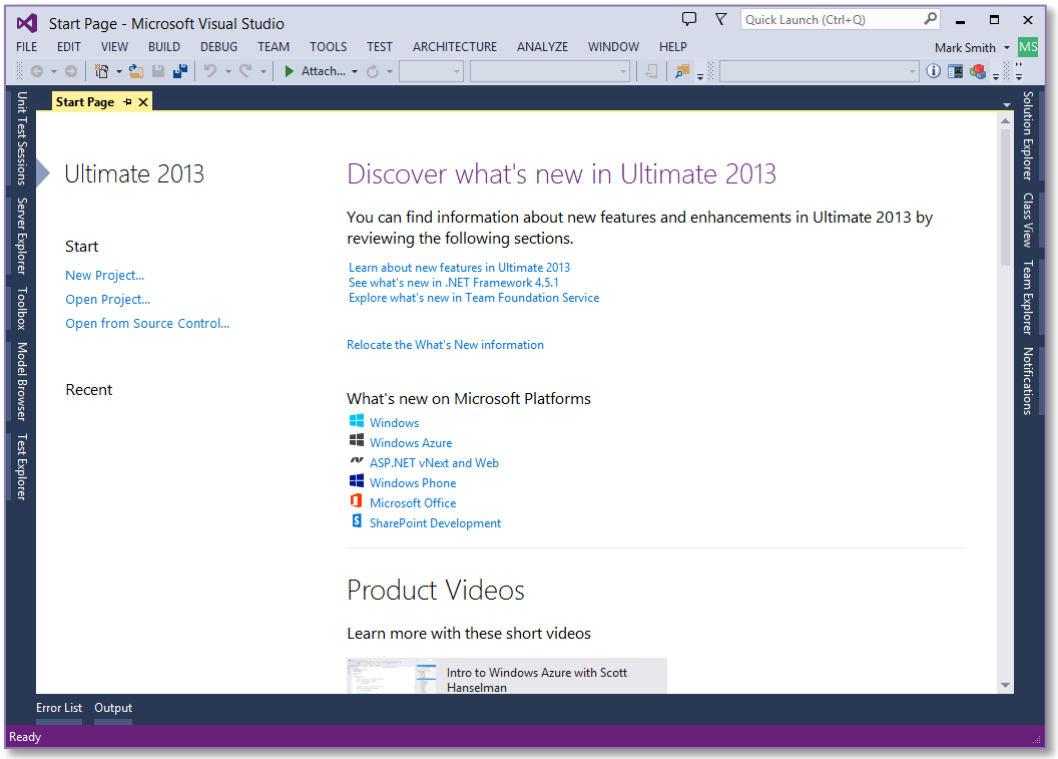


Make sure you select **Comments** in the drop-down so the tool is scanning the comments in the code. You can quickly jump to the code by clicking on the task itself to keep up with the lecture as the instructor runs through this lab. If you need additional time to complete a task or need some help please let the instructor know – the goal is for you to work through this lab in the class itself.

Steps

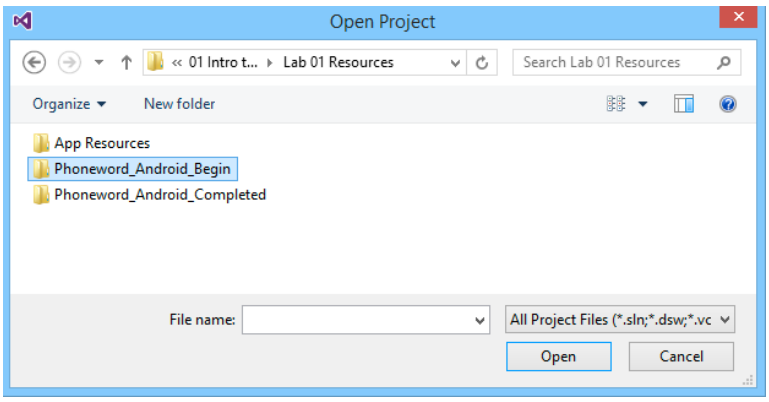
Open the Starting Solution

1. Launch **Visual Studio 2013.** It should open and look something like:



Note: the above is the Ultimate edition of Visual Studio 2013. You can actually use any version of Visual Studio except the Express edition.

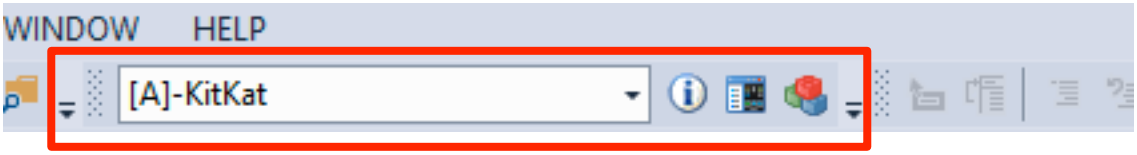
2. Click **Open Project...** on the Start Page and navigate to the **Lab 01 Resources** folder included with this document.
3. Locate the **Phoneword_Android_Begin** folder:



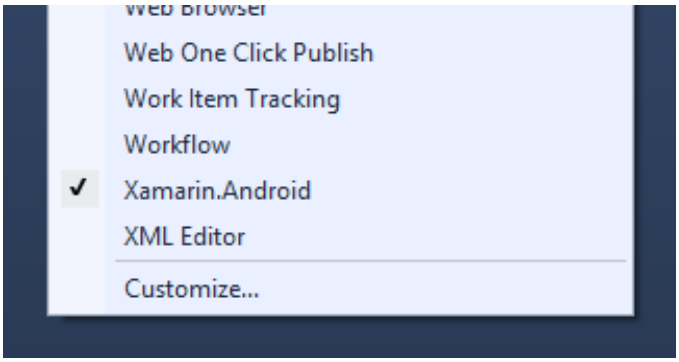
4. Inside the Phoneword_Android_Begin folder you will find a `PhonewordAndroid.sln` file – double click on this file to open the starter solution:

Testing the Initial Application with the Emulator

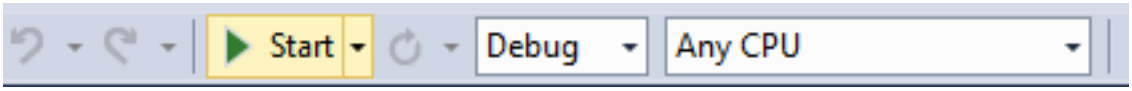
1. The Xamarin installer added a new toolbar into Visual Studio to make it easier to work with the Android SDK. It should be turned on by default and will look something like:



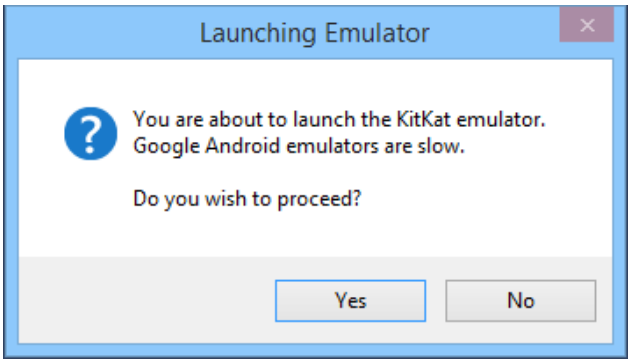
2. If you don't see the toolbar, right-click on the toolbar section in Visual Studio and make sure the **Xamarin.Android** toolbar is checked.



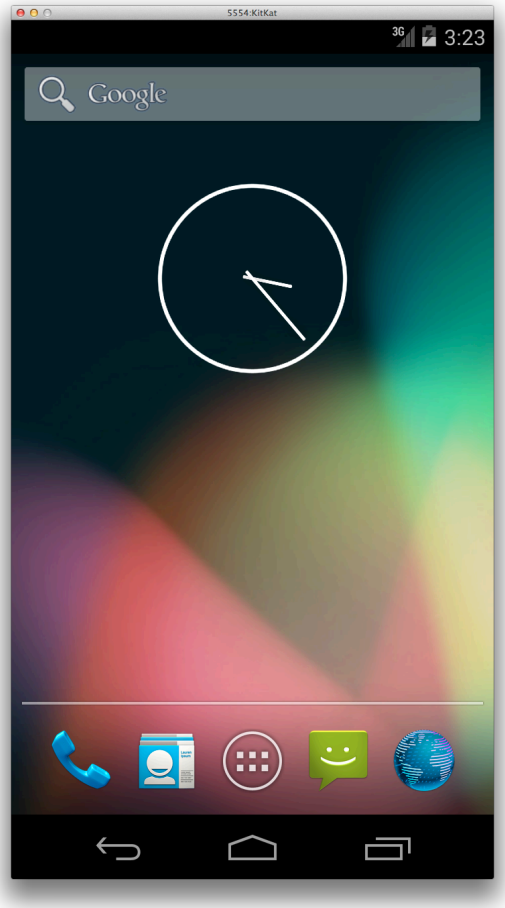
1. There is a Combo box in the **Xamarin.Android** toolbar, which lists the available devices, and emulators you can run your code in. There should be an entry for KitKat – this is Android 4.4 and, as of November 2013, is the most recent version of Android. Make sure this entry is selected and click the **Play** button in the **Debug** toolbar:



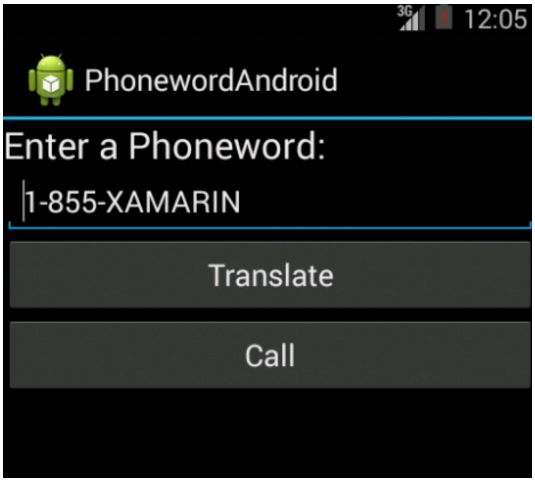
2. You should get a prompt indicating that Visual Studio is about to start the KitKat emulator:



3. Click **Yes** to dismiss the prompt. This will launch the emulator, which may take a few minutes.



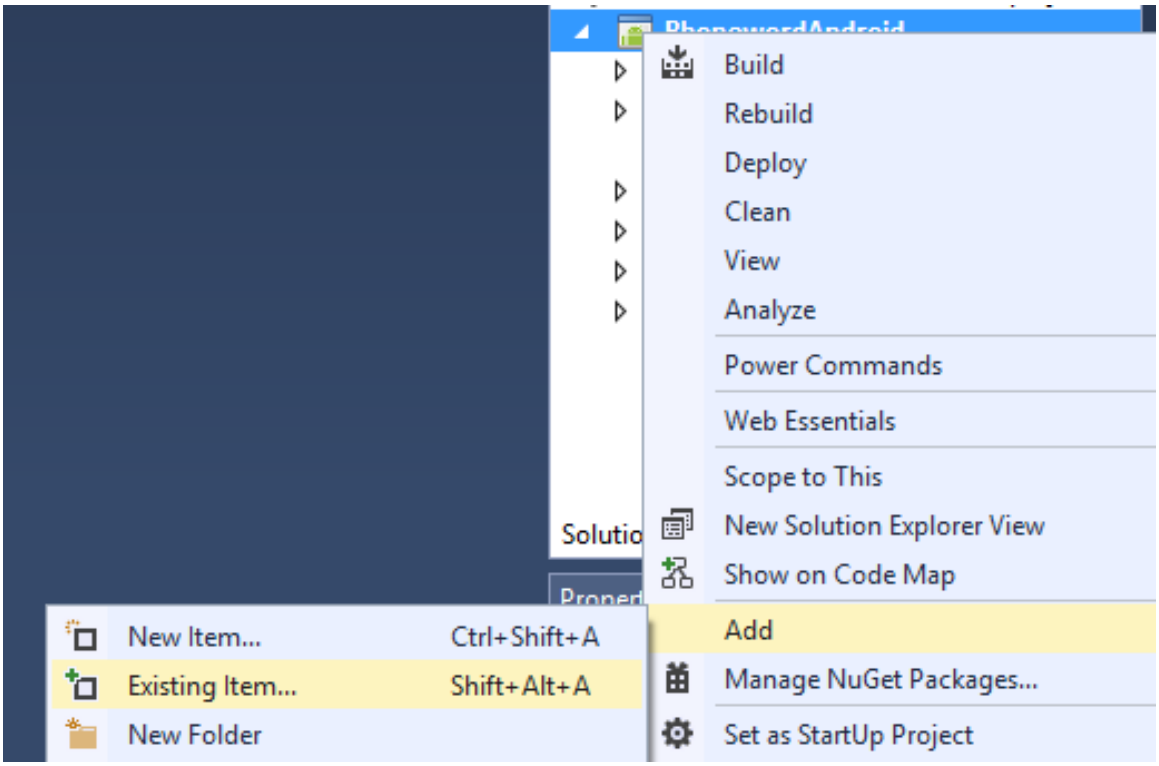
3. The application should run (it will take a minute or two the first time because it installs some shared libraries – just be patient) and you should see a screen that looks something like:



4. Notice that the UI has already been supplied for you - however if you click the **Translate** button in the UI it will change the text and increment the button count each time – this is the code from the normal Android starter project.
5. Press the **Stop** button in the debug toolbar in Visual Studio to stop running the program in the emulator. Leave the emulator running as it takes a while to start it each time.

Add PhoneTranslator.cs file to our project

1. Right-click on the **PhonewordAndroid** project folder and choose **Add > Existing Item...** as shown below:

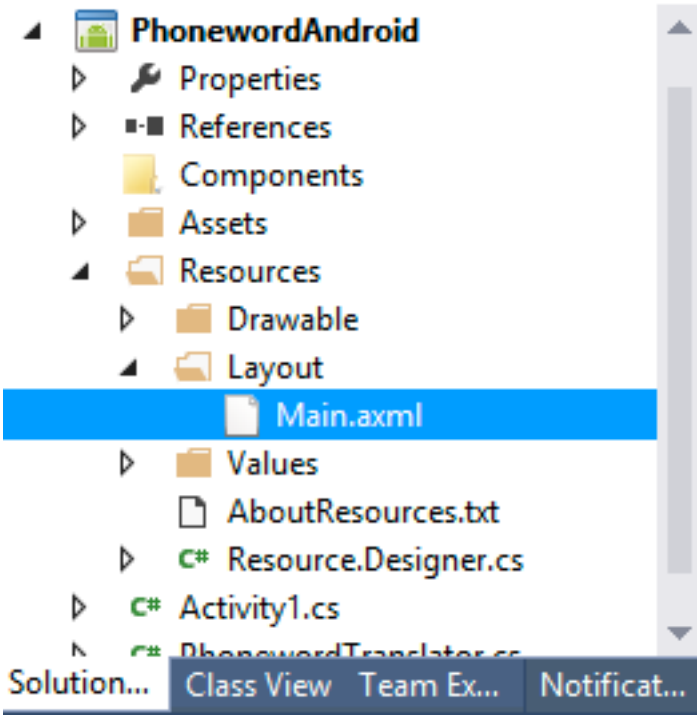


2. Navigate to the lab resources (**Lab 01 Resources**) that were included with this document.
3. Select the `PhonewordTranslator.cs` file and click the **Add** button.

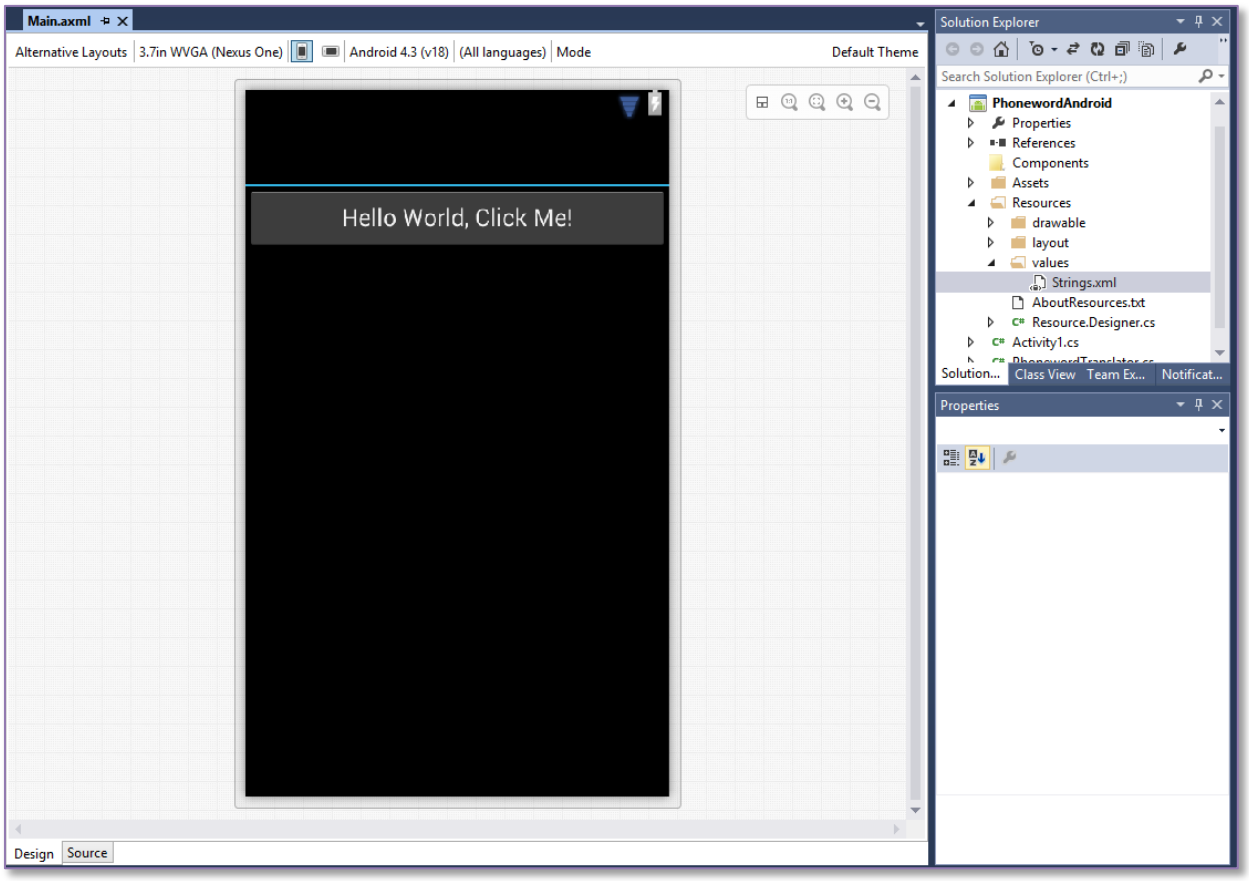
Create the User Interface

This task has already been completed for you, however the instructor will walk through these steps and create the UI from scratch so you can see how it is built – pay particular attention to how an identifier is assigned to a UI element. If you would like to build the UI on your own after class, the instructions are supplied below. Once the instructor is done with the UI, you can go to the next task: Implement the Click Event for the TranslateButton.

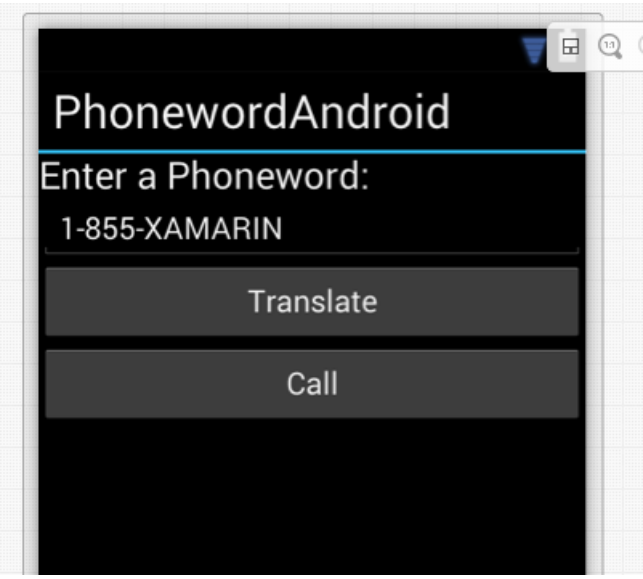
1. Expand the Resources folder, then the layout folder and double-click on the **Main.axml** file to display the UI in the Android visual designer.



2. Once it opens, it should look something like:

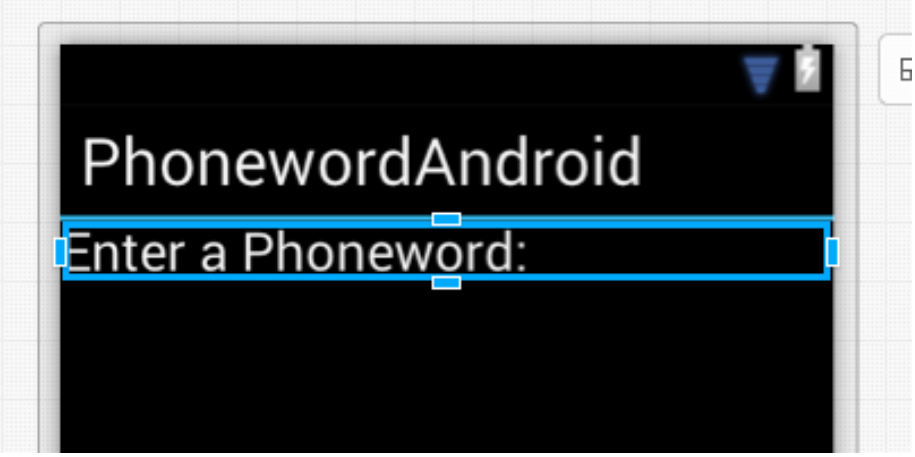


Create the following user interface using the steps below:



3. The Android project template created an initial UI, which needs to be deleted. Click on the “Hello World, Click Me!” button to select it and then press the DEL key to delete it from the design.

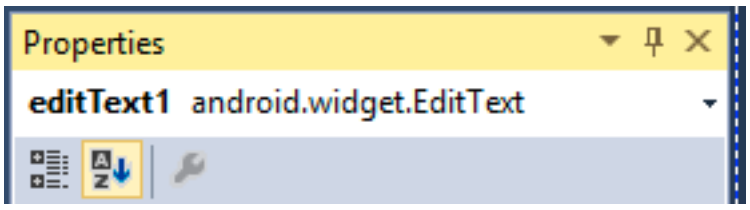
4. In the **Toolbox**, scroll down and locate the **Text (Large)** control. It is in the section titled "Form Widgets".
5. Drag a **Text (Large)** control into the **Designer** layout area and drop it – this will insert a **TextView** control and automatically line up at the top. Double-click the control and type, *Enter a Phoneword:* it should look similar to the following when complete:



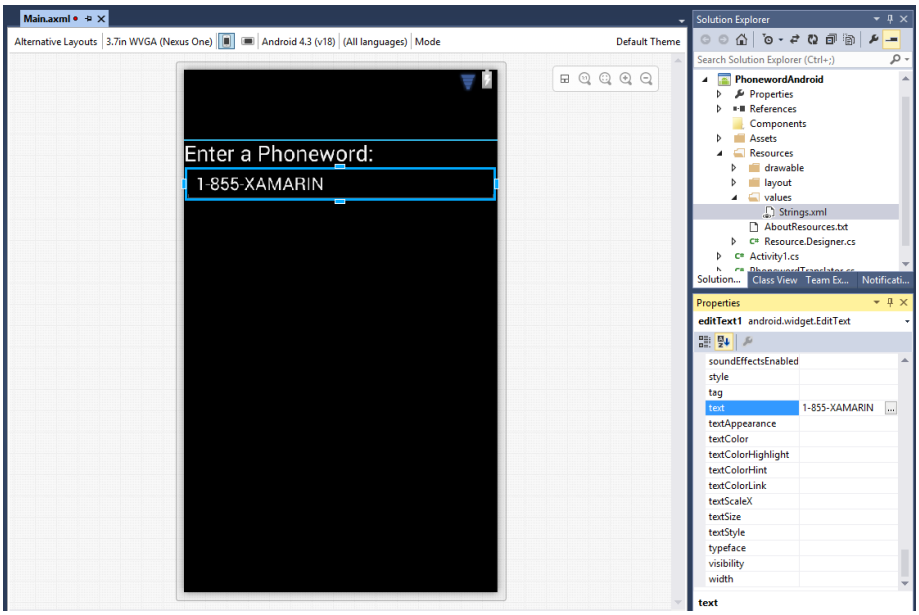
6. Next, locate and drag a **Plain Text** control under the **Text View**, this is in the Text Fields group in the **Toolbox**. This creates an **EditText** control for entering content; we will use this to enter our phone numbers.
7. Select the **EditText** control you just added and in the **Properties** tab set the following property values.

Property	Value
Id	@+id/PhoneNumberText
Text	1-855-XAMARIN

8. It is easiest to find these if you sort alphabetically in the **Properties** tab.



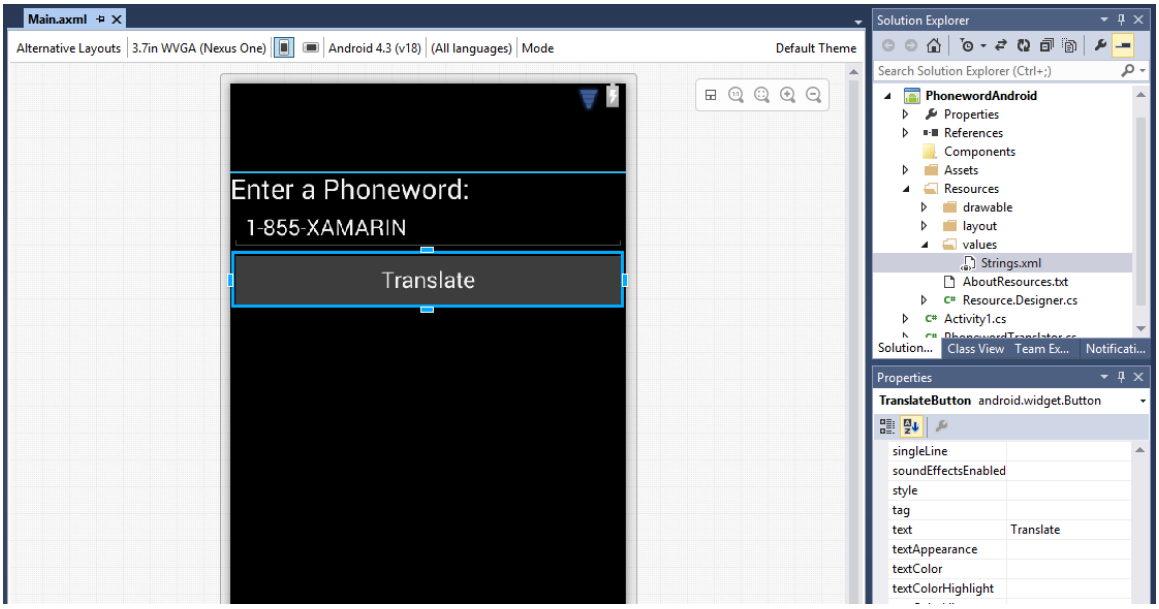
The result should look like:



9. Drag a **Button** under the **Plain Text** control. With the **Button** selected, in the **Property Tab**, make the following property changes:

Property	Value
Id	@+id/TranslateButton
Text	Translate

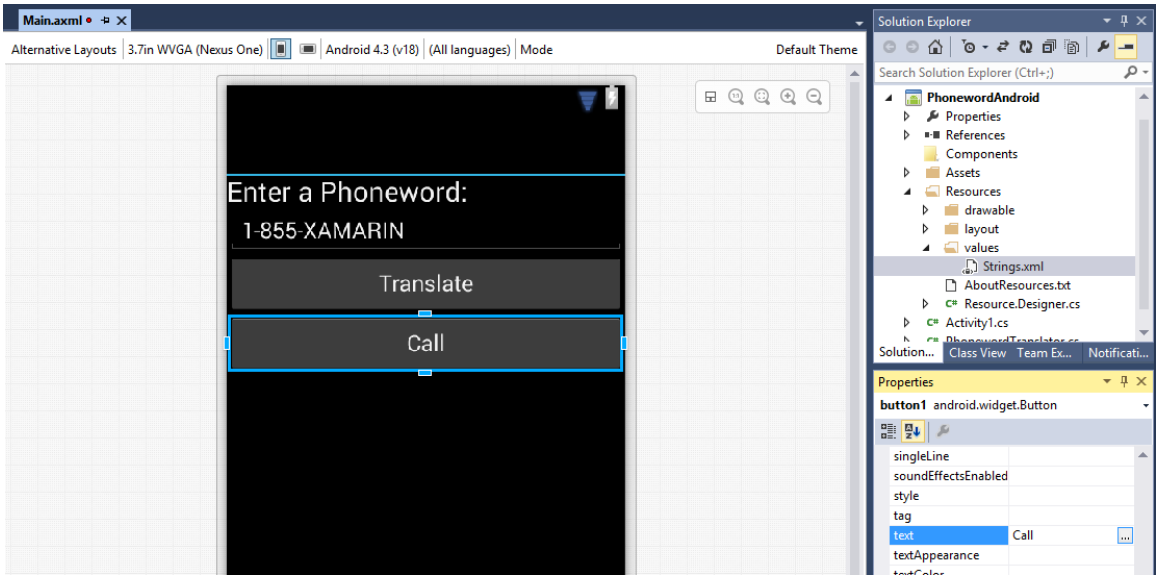
10. The result should look like:



11. Drag another **Button** under the **Translate** button and make the following changes in the **Property Tab**:

Property	Value
Id	@+id/CallButton
Text	Call

The result should look something like:



12. **File > Save All** (or **CTRL+SHIFT+S**) to save the layout.

Implement the Click Event for TranslateButton

1. Double-click `MainActivity.cs` to open the file.
2. The first step is to delete the boilerplate code which was added by the Android project template. Locate the comment `// TODO: Step 1 – Remove the boilerplate code, lines 14, 25-32.`

> **Tip**: use the Tasks Pad to quickly jump to `TODO` elements

3. Remove the `count` field on line 14, and all the code that provides the current button behavior – this should be lines 25 - 32. The specific lines to remove have been highlighted below; feel free to remove the `TODO` comments as you perform the steps as well.

```
public class MainActivity : Activity
{
    int count = 0;
```

```
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        // Set our view from the "main" layout resource
        SetContentView(Resource.Layout.Main);

        // TODO: Step 1 - Remove the boilerplate code, lines 14, 25-32

        // Get our button from the layout resource,
        // and attach an event to it
        Button button = FindViewById<Button>(Resource.Id.TranslateButt
on);
        button.Click += delegate
        {
            button.Text = string.Format("{0} clicks!", count++);
        };

        // TODO: Step 2 - Locate the controls in our created view.
```

4. Next, locate the comment `TODO: Step 2 - Locate the controls in our created view` and uncomment the block that follows it. The relevant code is shown below:

```
// TODO: Step 2 - Locate the controls in our created view.
// Get our UI controls from the loaded layout
Button translateButton = FindViewById<Button>(
                                Resource.Id.TranslateButton);
EditText phoneNumberText = FindViewById<EditText>(
                                Resource.Id.PhoneNumberText);
Button callButton = FindViewById<Button>(
                                Resource.Id.CallButton);
```

5. Next, locate the comment `TODO: Step 3 - Disable the "Call" button` and uncomment line right below it.

> Tip: use the accelerator key CTRL+K+M to comment or uncomment the selected lines in the editor.

6. Finally, locate the comment `TODO: Step 4 - Add code to translate number` and uncomment the block below it, which adds the behavior for the Translate Button in the UI. The code is shown below:

```
// TODO: Step 4 - Add code to translate number
string translatedNumber = string.Empty;

translateButton.Click += delegate
{
    translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumberTe
xt.Text);
    if (String.IsNullOrWhiteSpace(translatedNumber)) {
        callButton.Text = "Call";
        callButton.Enabled = false;
```

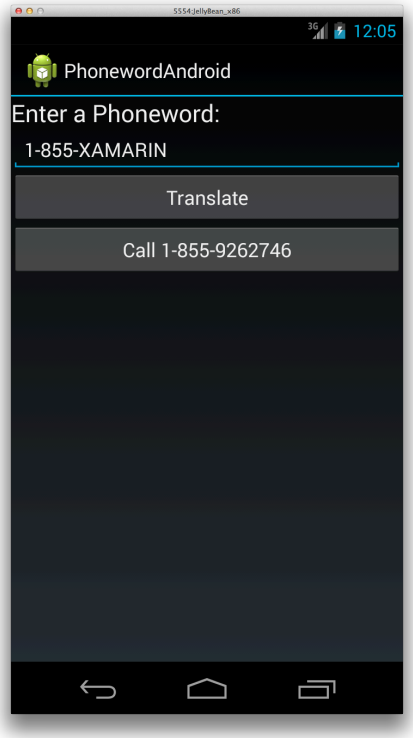
```
    }
    else {
        callButton.Text = "Call " + translatedNumber;
        callButton.Enabled = true;
    }
};
```

7. Build the application and fix any compile errors that occur. The code should now look like: (the TODO comments we have performed have been removed)

```
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        // Set our view from the "main" layout resource
        SetContentView(Resource.Layout.Main);

        // Get our UI controls from the loaded layout
        Button translateButton = FindViewById<Button>(
                            Resource.Id.TranslateButton);
        EditText phoneNumberText = FindViewById<EditText>(
                            Resource.Id.PhoneNumberText);
        Button callButton = FindViewById<Button>(
                            Resource.Id.CallButton);

        callButton.Enabled = false;

        // TODO: Step 4 - Add code to translate number
        string translatedNumber = string.Empty;

        translateButton.Click += delegate
        {
            translatedNumber = Core.PhonewordTranslator.ToNumber(
                            phoneNumberText.Text);
            if (String.IsNullOrWhiteSpace(translatedNumber)) {
                callButton.Text = "Call";
                callButton.Enabled = false;
            }
            else {
                callButton.Text = "Call " + translatedNumber;
                callButton.Enabled = true;
            }
        };

        // TODO: Step 5 - Add callButton event handler here
```

Testing the Application

1. The Translate Button is now ready to be tested. Go ahead and launch the application in the emulator by pressing the Play button in the toolbar. If you stopped the emulator earlier you will need to re-launch it.
2. Click the **Translate** button in the application and verify the **Call** button becomes enabled and the title changes to **Call 1-855-9262746**, as shown below:



3. Press the **Stop** button in the Vistual Studio toolbar to halt the application.

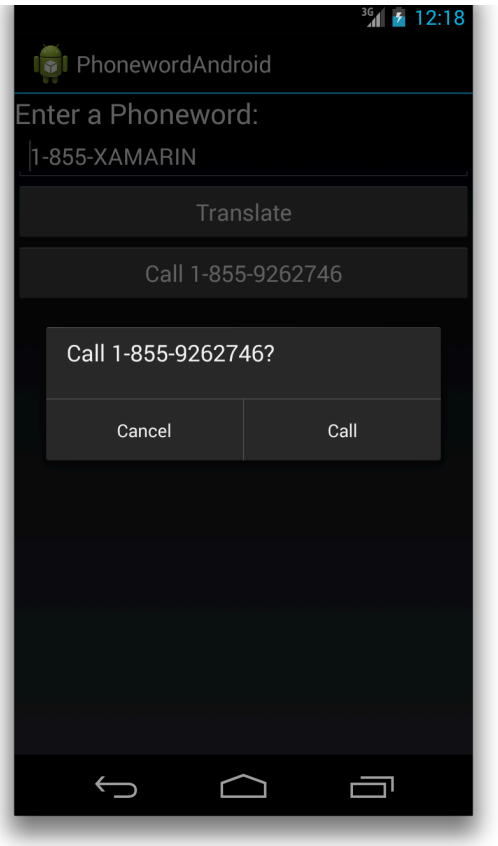
Implement the Click Event for CallButton

1. Switch to the `MainActivity.cs` source file and locate the comment TODO: `Step 5 - Add callButton event handler here` and uncomment the block of code that follows it. The code is shown below:

```
// TODO: Step 5 - Add callButton event handler here
callButton.Click += (sender, e) =>
{
    // On "Call" button click, try to dial phone number.
    var callDialog = new AlertDialog.Builder(this);
    callDialog.SetMessage("Call " + translatedNumber + "?");
    callDialog.SetNeutralButton("Call",
        delegate
        {
            // Create intent to dial phone
            var callIntent = new Intent(Intent.ActionCall);
```

```
            callIntent.SetData(Android.Net.Uri.Parse(
                                "tel:" + translatedNumber));
            StartActivity(callIntent);
        });
    callDialog.SetNegativeButton("Cancel", delegate {});

    // Show the alert dialog to the user and wait for response.
    callDialog.Show();
};
```

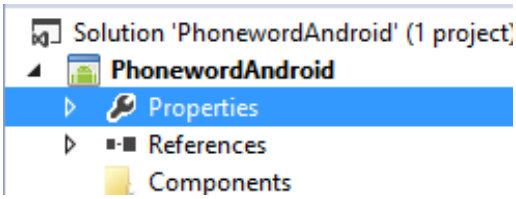
2. Click the **Play** button to build and launch the application in the emulator.
3. Tap the **Translate** button to populate our **Call Button** with **Call 1-855-9262746**.
4. Press the **Call Button** and an **Alert Dialog** should appear with a prompt to call the number:



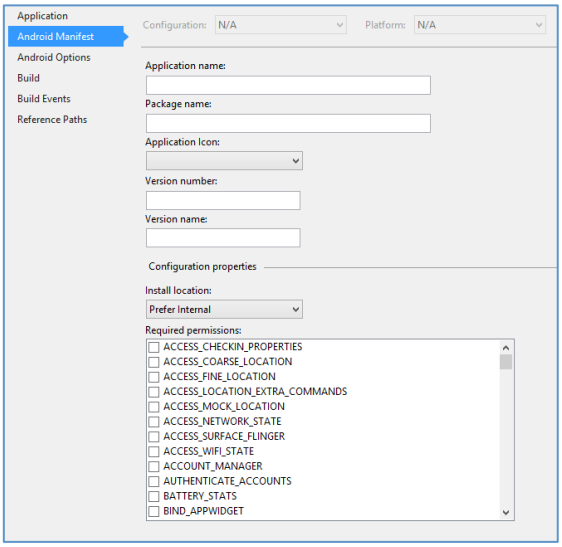
5. Clicking **Call** will result in a simulated call taking place, or rather it would if we had the proper permissions setup in this application. Instead, it produces a `SecurityException`. To fix this security exception, we need to request the proper privileges for our application to be able to place an outgoing call.
6. Go ahead and stop the program (click the **Stop** button in the toolbar).

Requesting Make Call privileges

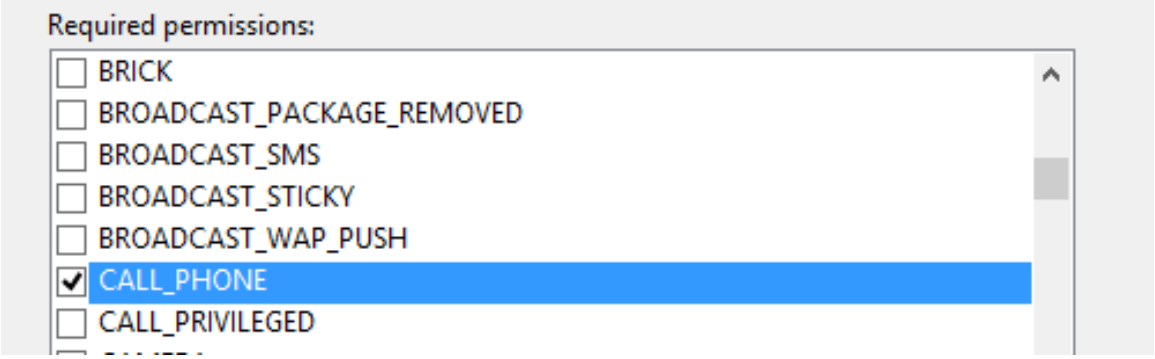
1. Double-click on the **PhonewordAndroid** project in the **Solution Explorer** (or right-click the project element in the tree and select **Properties**).



2. Select the “Android Manifest” entry:



3. Find the **CALL_PHONE** entry in the Required Permissions section and check the checkbox next to it:

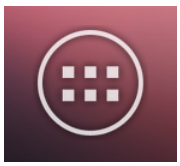


4. Click **OK** to dismiss the dialog.
5. Run through the steps to test your application again and press the **Call** button. This time, it should show a simulated call:

Finishing Touches – Application Name

1. While the **Android Emulator** is still running, click the center **Home Button** on the emulator screen. You can also use the **Home Button** in the skin if you have that enabled. The **Home Button** looks like this:

2. On the Home screen, click the **All Applications Button** to display all the installed applications and locate our new app we just created:



3. Our app name defaults to the name of our project, **PhonewordAndroid**, which does not wrap properly under the app icon. Additionally, the icon is the default app icon, as shown below:

4. Switch back to **Visual Studio** and open the `MainActivity.cs` source file.

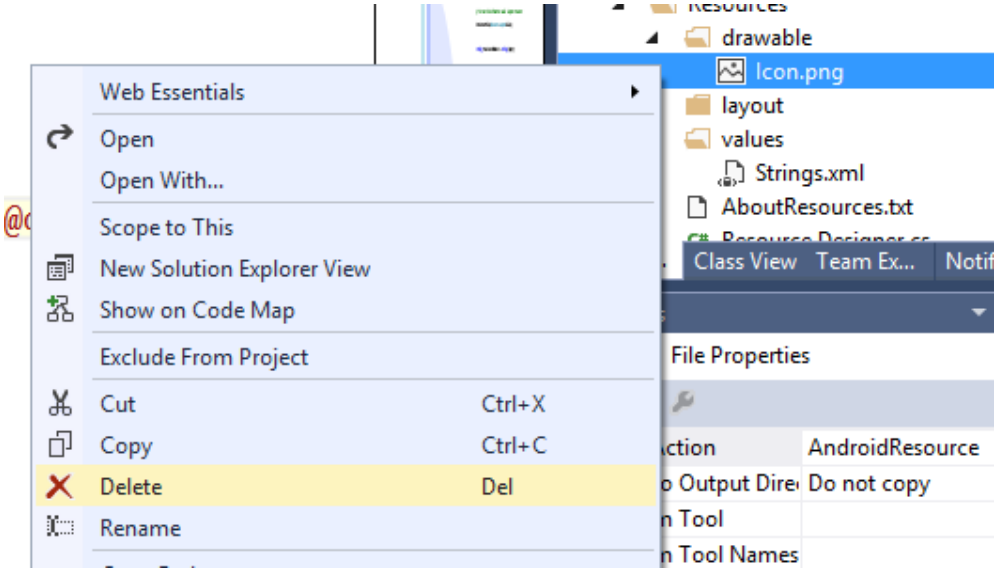
5. On top of the class there will be an **Activity** attribute applied which has a **Label** property. That label is used for the title of the application in this case – change it to have a space between **Phoneword** and **Android**. There is a `TODO: Step 6 – fix title comment` in the source file. Go ahead and add a space into the name - it should look like:

```
[Activity(Label = "Phoneword Android", MainLauncher = true)]
public class MainActivity : Activity
{
```

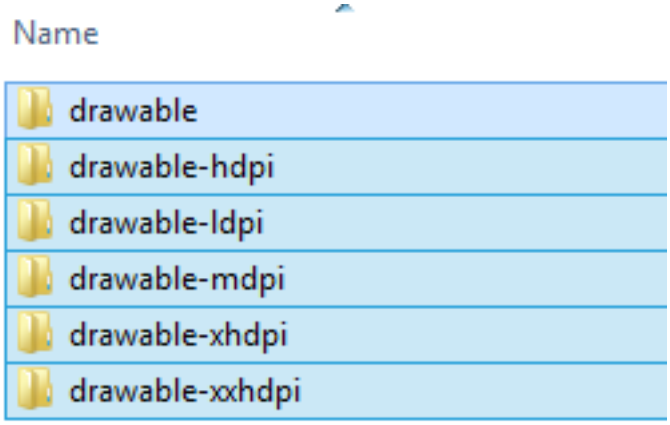
6. Click the **Play** button to rerun our app.
7. Verify our new app name is in the title bar while it is running, and by clicking on the **Home Button** to stop the application, so we can see the new **app name** under the icon.

Finishing Touches – Updating the Icon

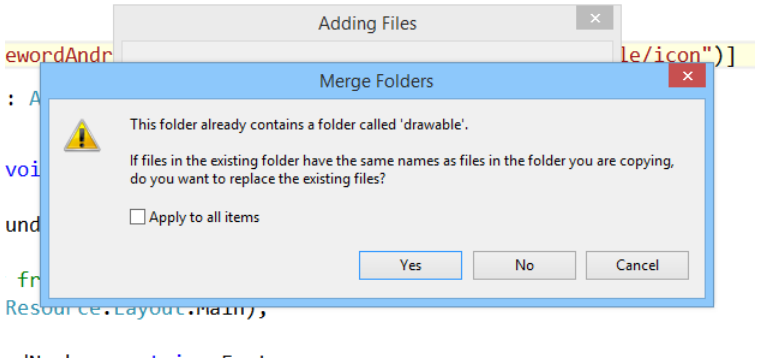
1. Let's add some custom app icons and launch images to our project.
2. Expand the **Resources** node in the **Solution Explorer** and then the **drawable** folder. Remove the **Icon.png** file, which is currently in the folder by right clicking on it and selecting **Delete**.



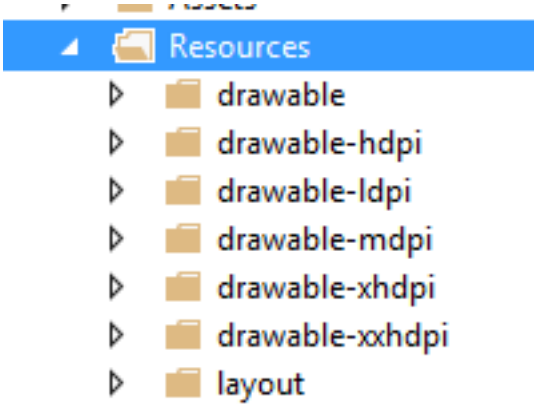
Next, open a new Explorer window and navigate to the **App Resources** folder under your lab resources (**Lab 01 Resources**). Multi-select all the `drawable-*` folders you find there (there are six of them) and drag them onto the **Resources** folder in the **Solution Explorer** in Visual Studio – this will copy all the folders into your project in one operation. Alternatively, you can right-click on the **Resources** node and create each folder individually and then add all the icons separately but this takes a lot longer.



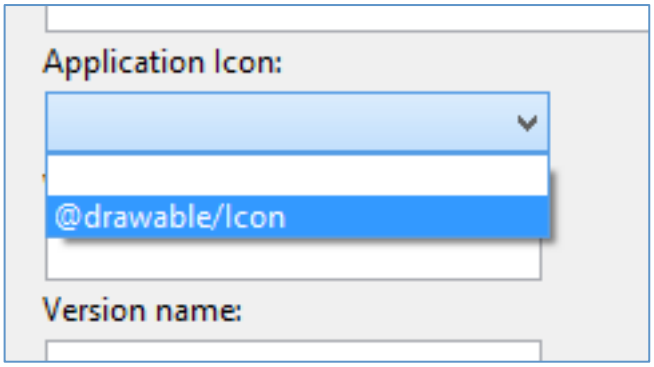
3. Visual Studio will prompt you to merge the **drawable** folder, just select **Yes**:



4. Once you are done, you should see each folder under the **Resources** node in the **Solution Explorer**, in each folder should be an **Icon.png** file which will be the icon for a specific pixel density device:



5. Next, open the **Project Options** by either double-clicking on the **PhonewordAndroid** project element in the **Solution Explorer**, or right clicking on the project node and selection **Properties**.
6. In the **Android Manfest** element, use the **Application Icon** drop-down to select the `@drawable/icon` entry:



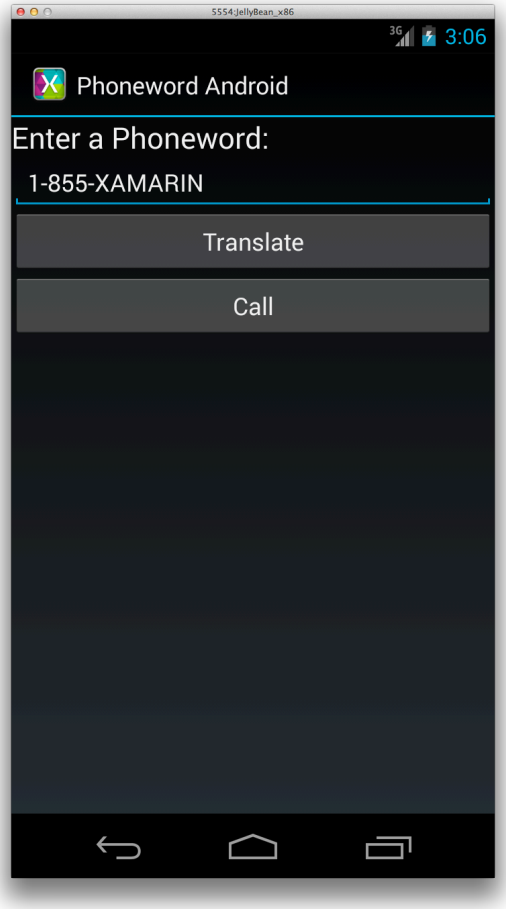
7. Next, open the `MainActivity.cs` source file and locate the `Activity` attribute applied to the top of the `MainActivity` class. There is a `TODO: Step 7 - add icon` comment here. Add a new `Icon` property to the attribute and set its value to `@drawable/icon` as shown below:

```
namespace PhonewordAndroid
{
    [Activity(Label = "Phoneword Android", MainLauncher = true,
              Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
```

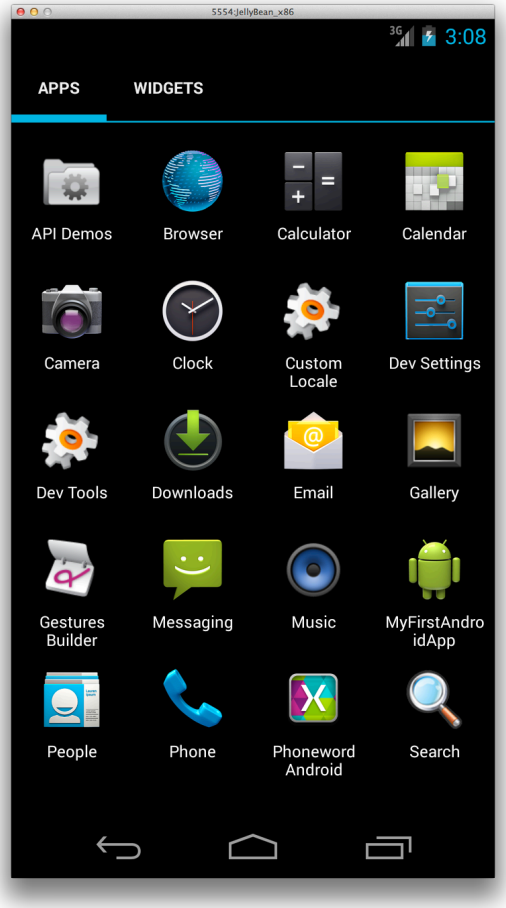
Final Testing of the Application

1. Build and Run the application by clicking the **Play** button in the toolbar.
2. You should see the new Xamarin icon in the title bar while the application executes:

> **Note**: some versions of Android do not display an icon – so if all you see is a Title, you are likely running an older version of the operating system.



3. Then, if you click the **Home** button to go to your home screen, and then the **Show All Apps** button on the Home screen to find the application you should also see our new icon on the application itself:



Congratulations!

We covered a lot of ground here. You now have a solid understanding of the components of a **Xamarin.Android** application, as well as the tools used to create them.

Summary

In this lab, we have built our first **Xamarin.Android** application and learned how to use the Xamarin.Android Tools for Visual Studio. In the next lab, we're going to build on to this application by adding another screen.