Introduction to Android with Xamarin Studio

Lab 1

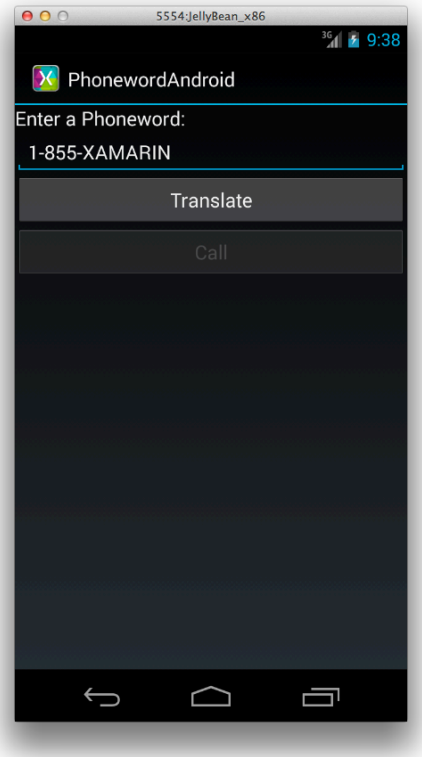
Lab Goals

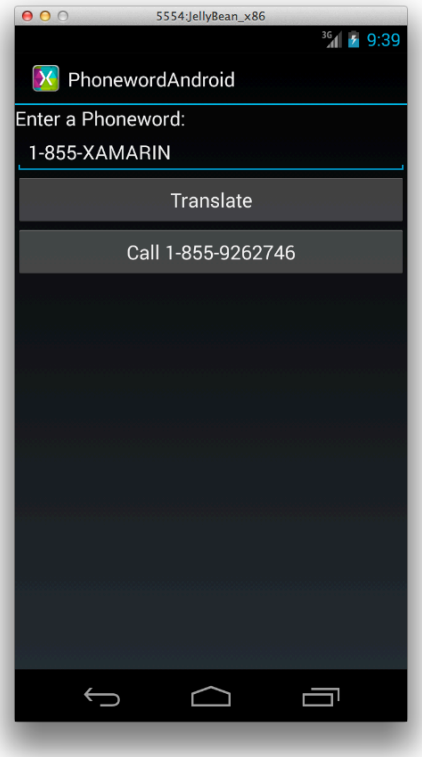
The goal of this lab will be to create our first **Xamarin.Android** application. During the process we will become familiar with the tools within Xamarin Studio, and the various parts of our application.

In this lab, we're going to walk through how to create a simple application from start to finish. We'll cover the following items:

- **Xamarin Studio** – Introduction to Xamarin Studio and how to use it to create **Xamarin.Android** applications. It is also possible to use Visual Studio 2010, 2012 or 2013 if you are running on Windows.
- **Anatomy of a Xamarin.Android Application** – Introduce the core parts of a Xamarin.Android application.
- **Xamarin Studio Designer** – How to use Xamarin Studio's Designer to create your application's user interface (UI).
- **Android Emulator** – How to setup and use the Android SDK emulator to test your application.

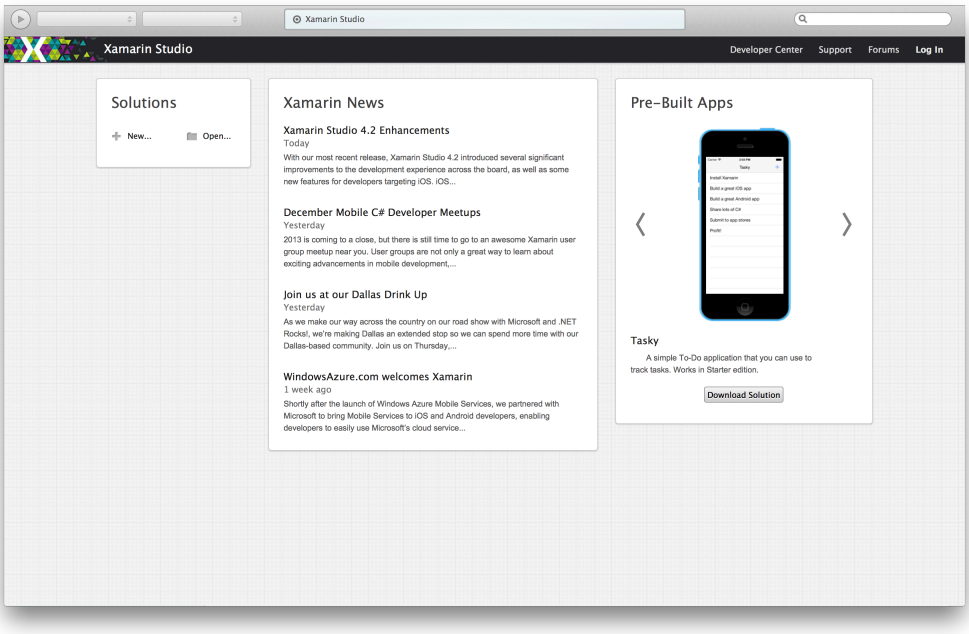
We're going to create a simple application that translates a phone number with letters in it, i.e. `1.855.XAMARIN`, and then allows the user to call it:





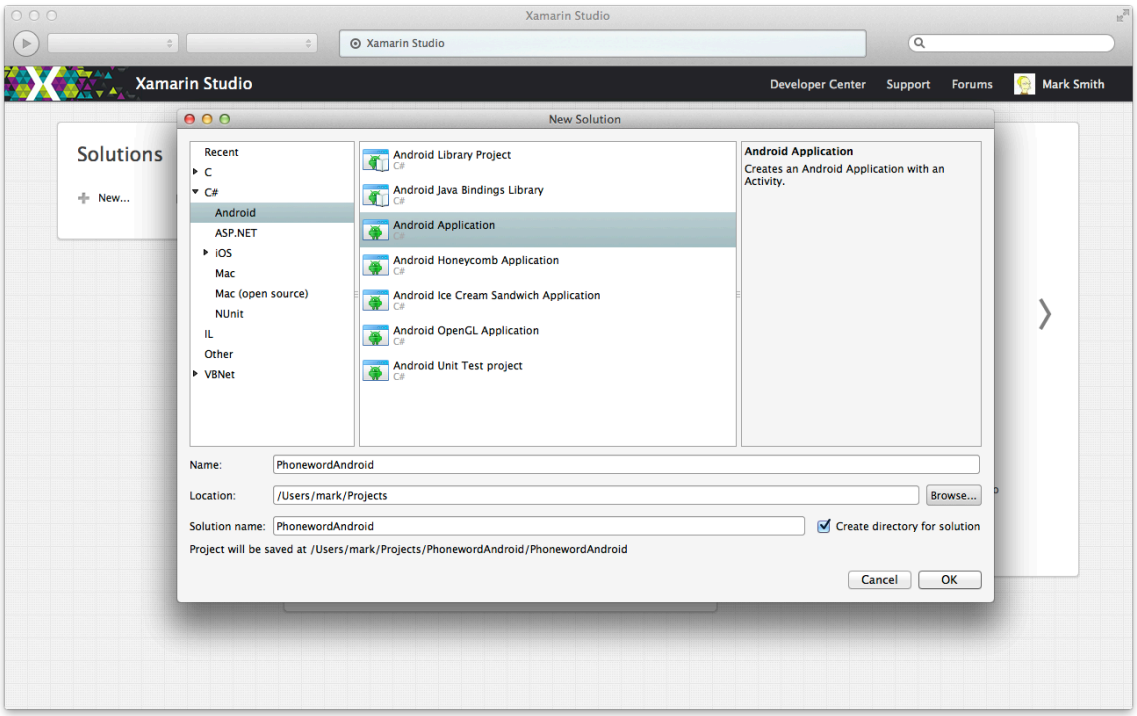
Create the solution with Xamarin Studio

1. Open your virtual environment or log onto your physical development environment with the Android development tools. Please contact your instructor, if there are any issues.
2. Double-click the **Xamarin Studio** icon provided for you on the **Desktop** of your environment. Xamarin Studio will open up and look something like the following:



Create a New Solution

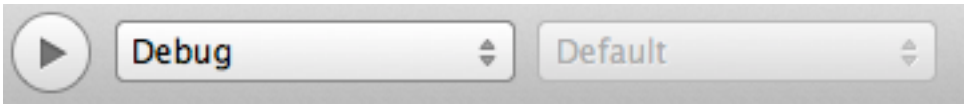
1. Click **+New...** within the Xamarin Studio Welcome screen, this action opens the **New Solution** dialog.
2. In the left-hand pane, choose **C# > Android**
3. Select the **Android Application** template.
4. In the **Name** field, enter `PhonewordAndroid` as shown below:



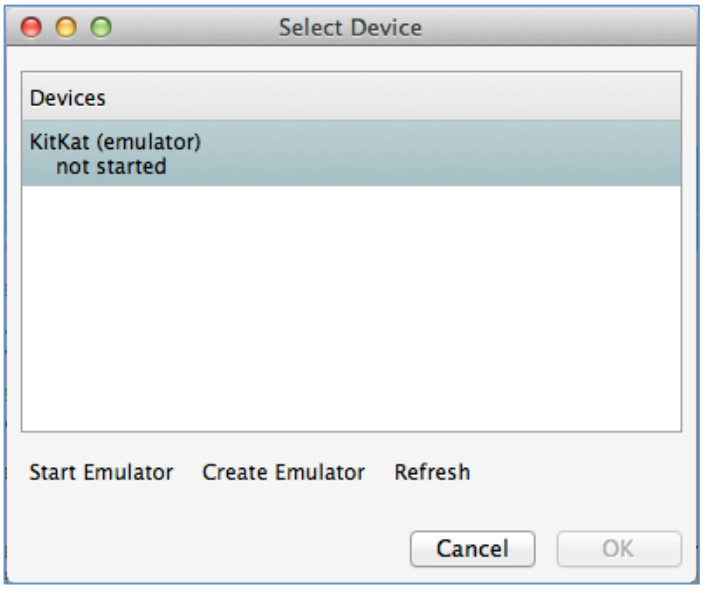
5. Click the **Browse…** button and navigate to the **Desktop** on your VM and click the **Open** button. This will place the **Solution folder** for our new app on the **Desktop** for convenience in the lab.
6. Click the **OK** button to create our new solution, `PhonewordAndroid`. Xamarin Studio will create a new solution containing a **Xamarin.Android** application.

Testing the Initial Application with the Emulator

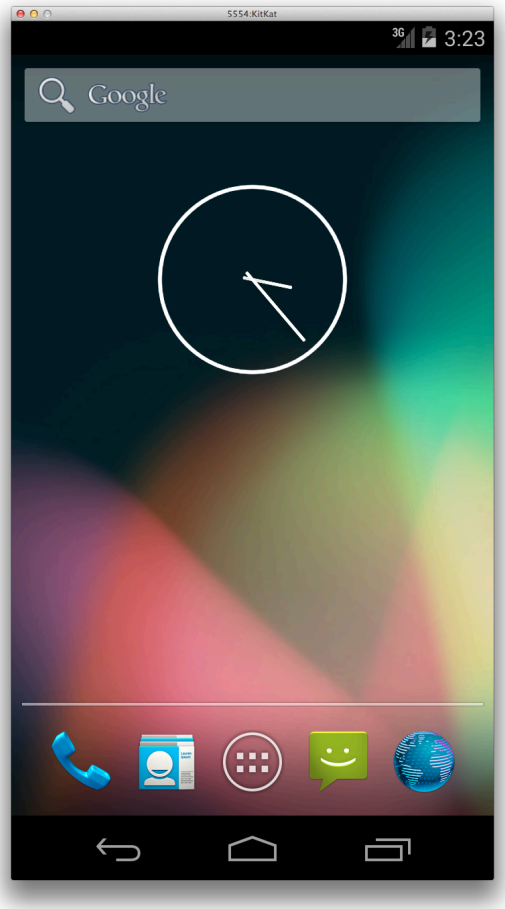
1. In Xamarin Studio, choose **Debug** in the toolbar at the top and click the **Play** button:



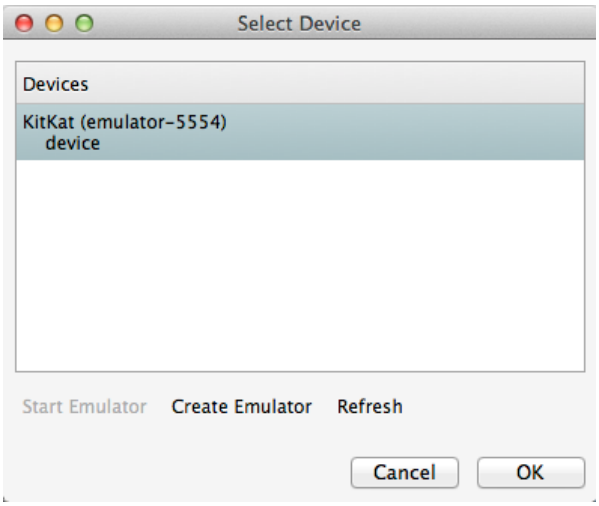
2. The **Select Device** dialog should be shown allowing you to select the Emulator instance to start or execute the application in. There should be an entry for KitKat – this is Android 4.4 and, as of November 2013, is the most recent version of Android.



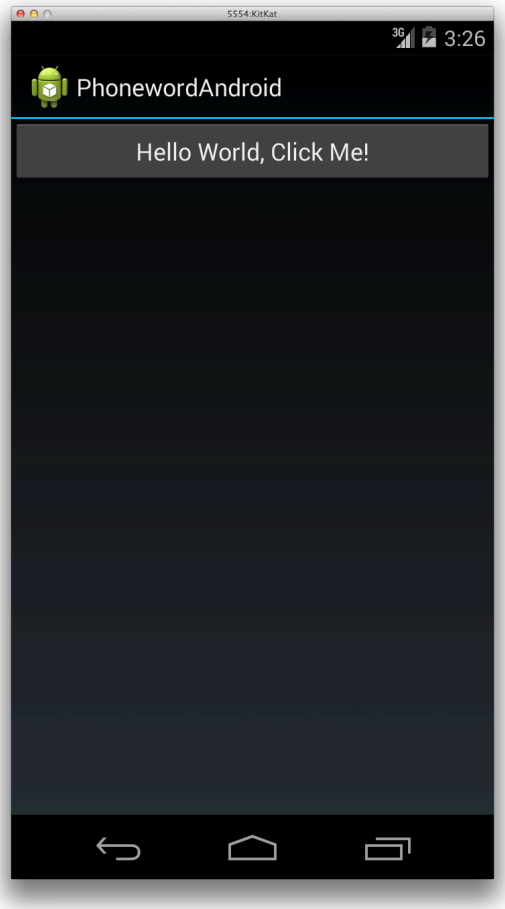
3. Select the KitKat emulator and start it by clicking the "Start Emulator" button. This will launch the emulator, which may take a few minutes.



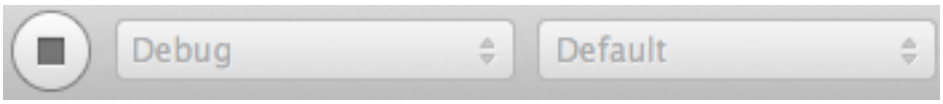
4. Once it's launched and running, unlock it if necessary by sliding the lock and then switch back to the **Select Device** dialog, select the Emulator and click OK to launch your application.



5. 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:

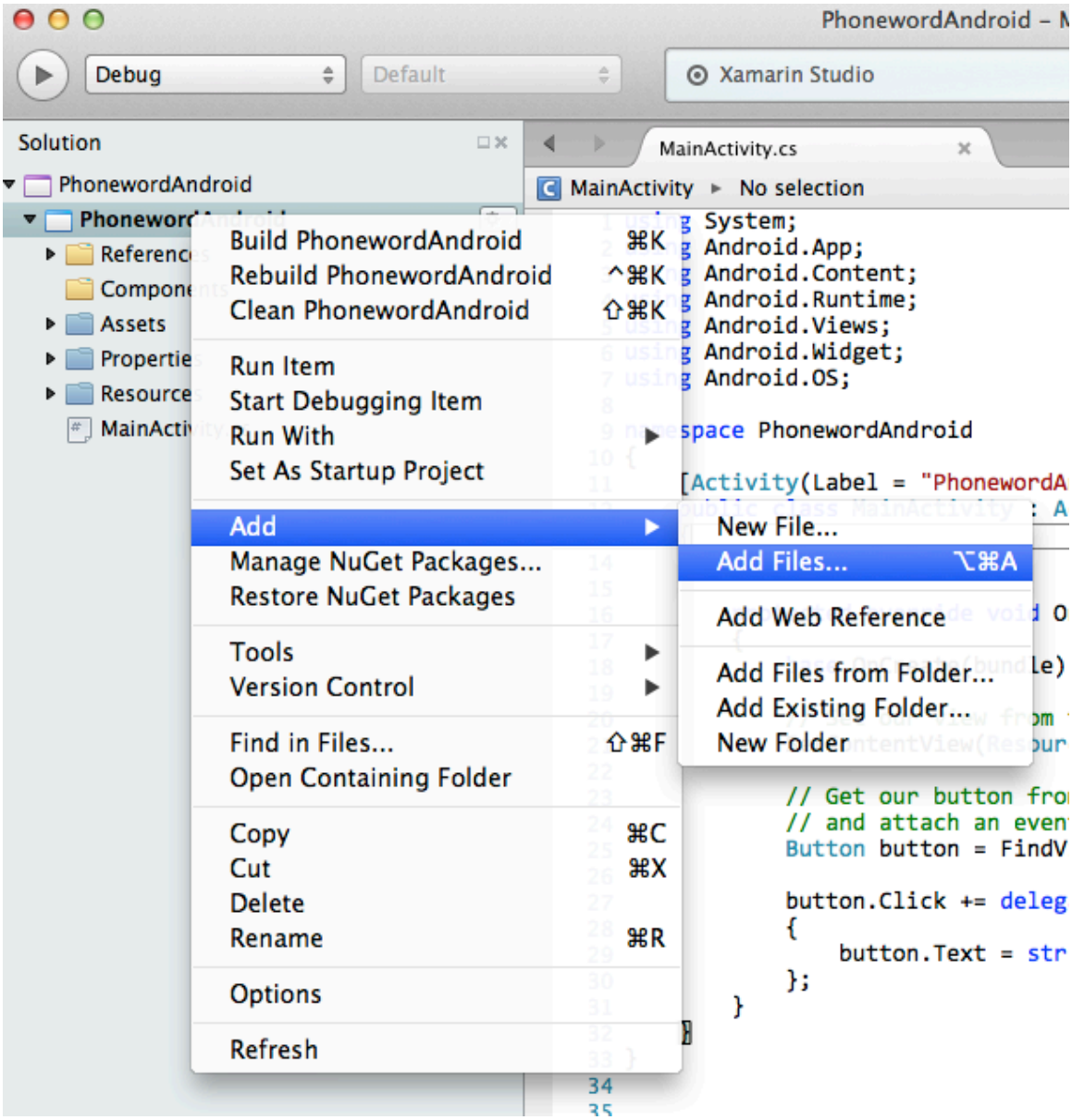


5. Press the **Stop** button in Xamarin Studio to stop running the program in the emulator. Leave the emulator running as it takes a while to start it each time.



Add PhoneTranslator.cs file to our project

1. Right-click (or Control-Click) on the **PhonewordAndroid** project folder and choose **Add > Add files...** as shown below:



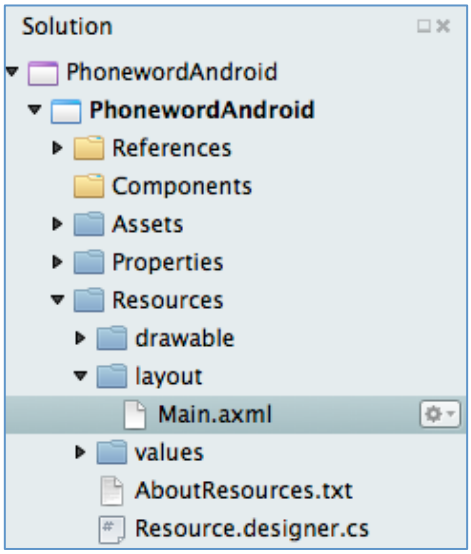
2. Navigate to

   `Xamarin Studio Labs/Intro to Android/Lab 01 Resources/`

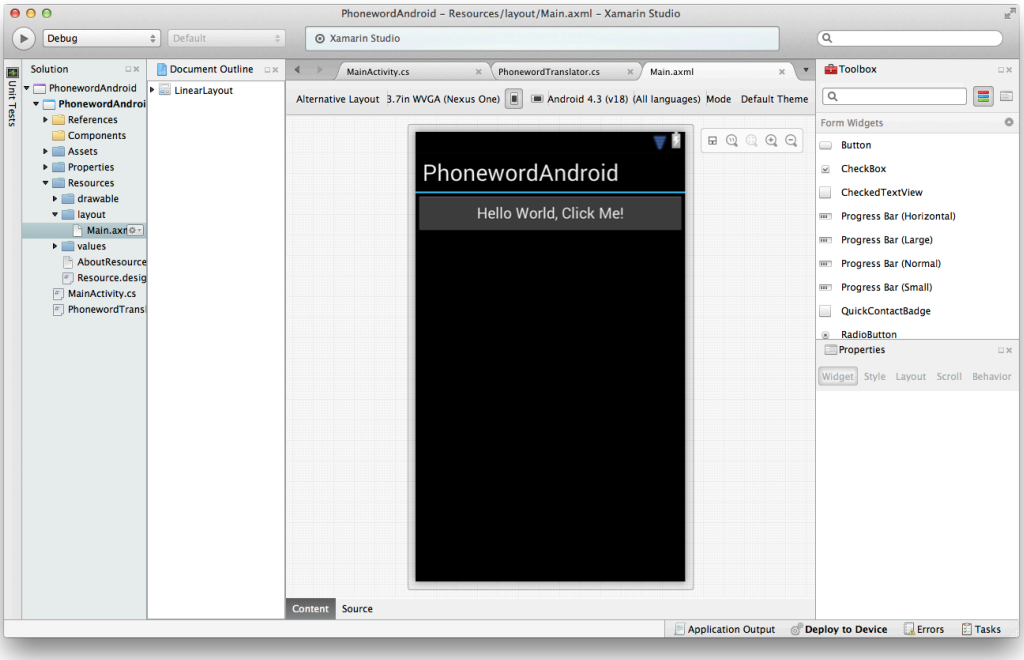
3. Select `PhonewordTranslator.cs` and click the **Open** button.
4. Choose `Copy the file to the directory` and click the **OK** button.

Create the User Interface

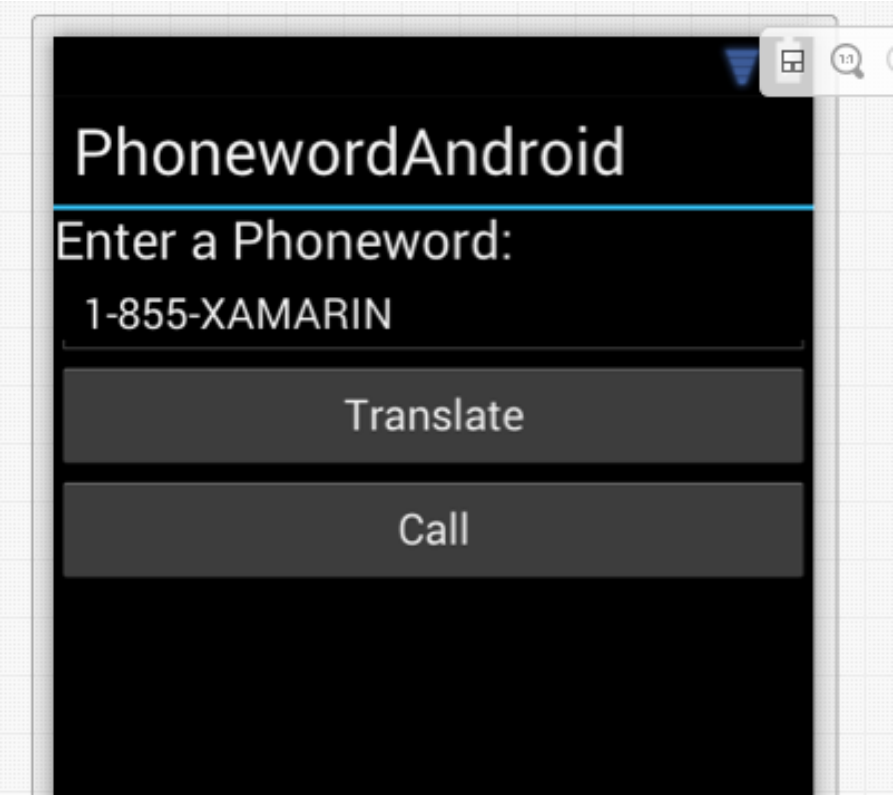
1. Expand the Resources folder, then the layout folder and double-click on the **Main.axml** file to display the UI in the Android visual designer.



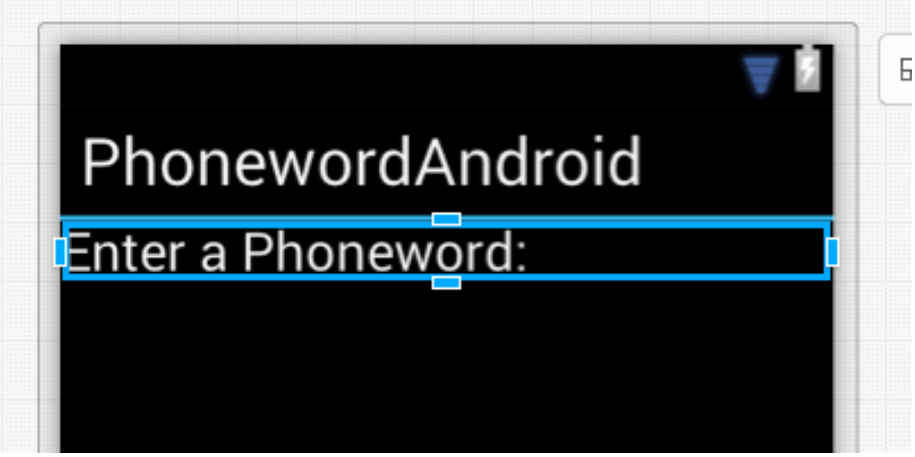
2. Once it opens, it should look something like:



Create the following user interface using the steps below:



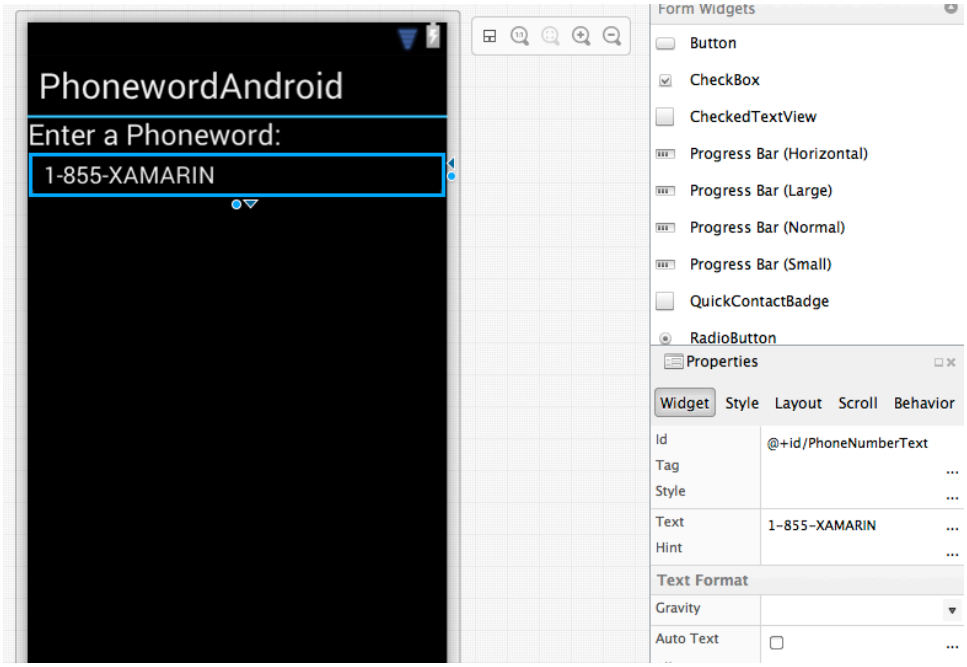
3. The Android project template created an initial UI, which needs to be deleted. Click on the "Hello World, Click Me!" button to select it and then press the DEL (or Fn-DEL) key to delete it from the design.
4. In the **Toolbox Pad**, scroll down and locate the **Text (Large)** control. It is in the section titled "Form Widgets" which should be the first group in the Toolbox.
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 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 Pad** set the following property values, these are all on the "Widget" tab.

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

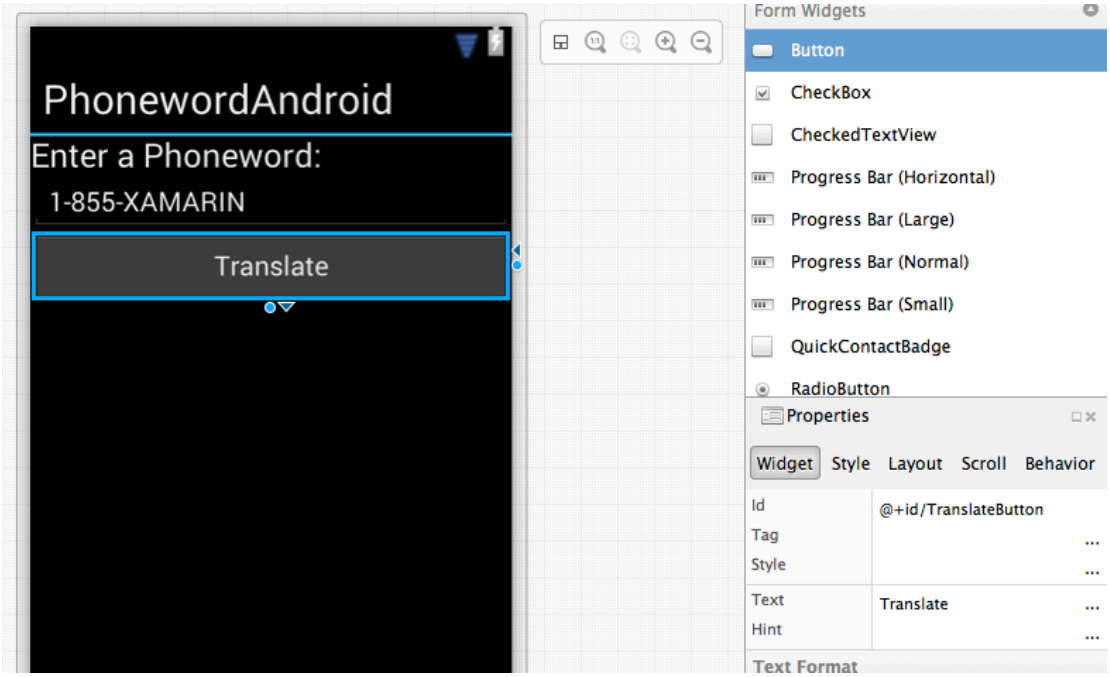
The result should look like:



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

Property	Value
Id	@+id/TranslateButton
Text	Translate

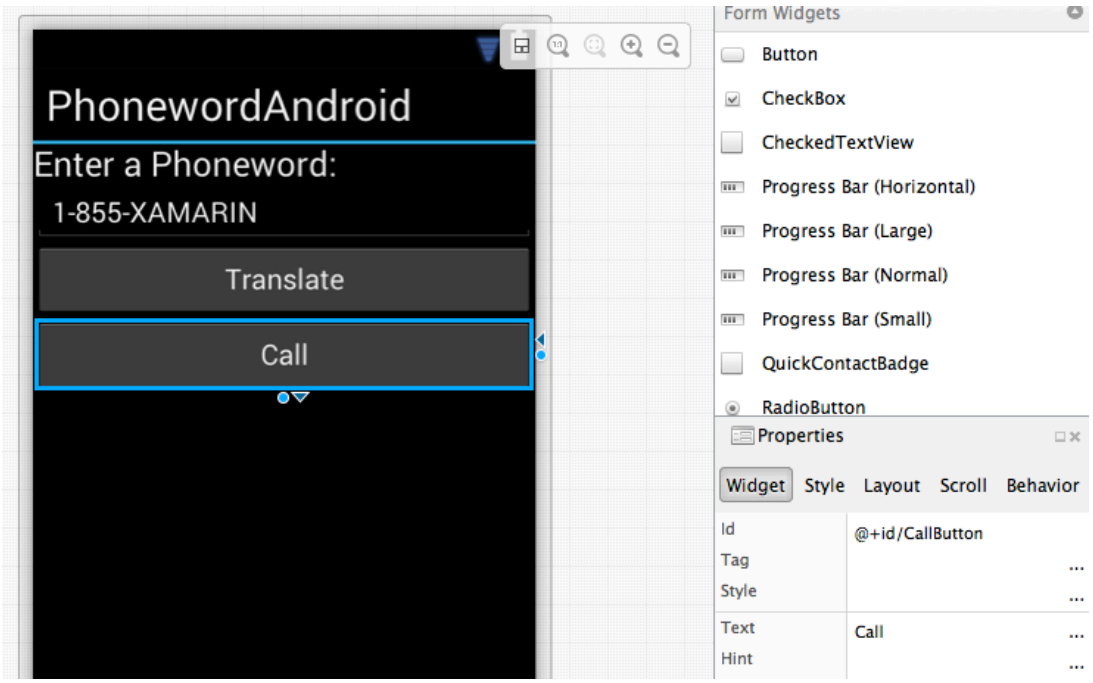
9. The result should look like:



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

Property	Value
Id	@+id/CallButton
Text	Call

The result should look something like:



11. **File > Save** (or **Command-S**) to save the layout.

Implement the Click Event for TranslateButton

1. Double-click `MainActivity.cs` to open the file.
2. Delete the count integer field in the class.
3. Remove the implementation for the `OnCreate` method and replace it with the following code:

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

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


    string translatedNumber = string.Empty;

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

    callButton.Enabled = false;
```

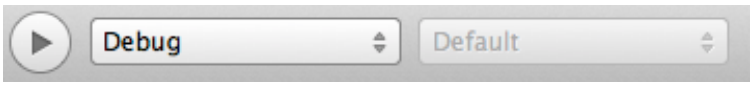
```
    translateButton.Click += delegate
    {
        // *** SHARED CODE ***
        translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumb
erText.Text);
        if (String.IsNullOrWhiteSpace(translatedNumber)) {
            callButton.Text = "Call";
            callButton.Enabled = false;
        }
        else {
            callButton.Text = "Call " + translatedNumber;
            callButton.Enabled = true;
        }
    };

    // TODO: Step 1 - Add callButton event handler here
}
```

4. Build the application using **Build > Build Solution** (or **CTRL+SHIFT+B**). The project should build properly. If there are any errors, check the error messages and compare the code to the lab instructions. It may also help to check the completed version of the lab under

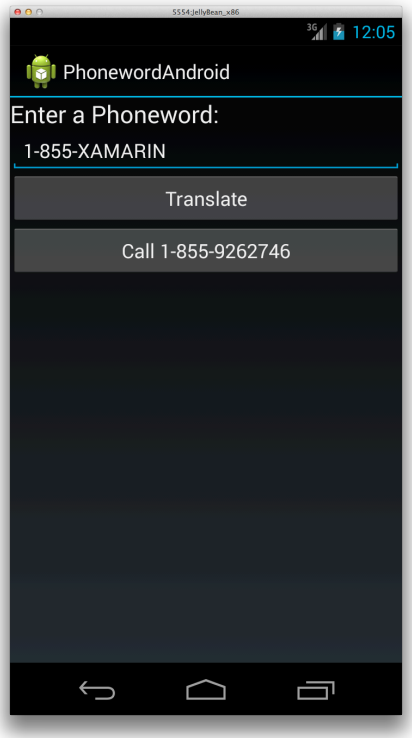
   `Xamarin Studio Labs/Intro to Android/Lab 01 Resources/Phoneword_Android_Completed.`

Testing the Application

1. Our `TranslateButton` is now wired and ready. Let's test it. In Xamarin Studio, choose **Debug** and click the **Play** button to launch it in the emulator as before, if you stopped the emulator you will need to re-start it before the application will run.



2. The application should start if everything has gone well.
3. Click the **Translate** button in our app and verify the **Call** button becomes enabled and the title changes to `Call 1-855-9262746`, as shown below:



4. Press the **Stop** button in Xamarin Studio to halt the application.

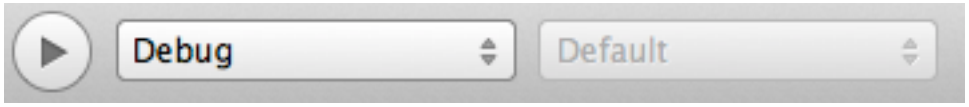
Implement the Click Event for CallButton

1. Double-click `MainActivity.cs` and add the following code for the CallButton to the `OnCreate` method under the **TODO Step 1** comment

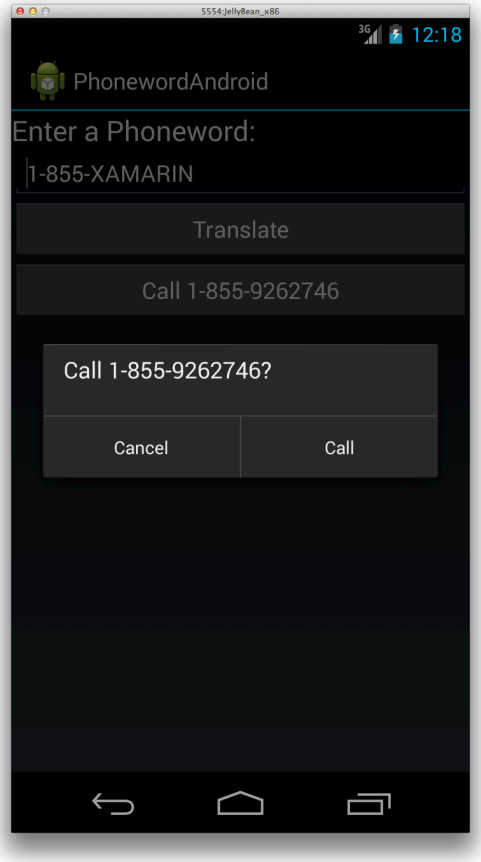
```
// TODO: Step 1 – Add callButton event handler here

// On "Call" button click, try to dial phone number.
callButton.Click += (object sender, EventArgs e) =>
{
    var callDialog = new AlertDialog.Builder(this)
        .SetMessage("Call " + translatedNumber + "?")
        .SetNeutralButton("Call",
            delegate {
                var callIntent = new Intent(Intent.ActionCall);
                callIntent.SetData(Android.Net.Uri.Parse(
                                    "tel:" + translatedNumber));
                StartActivity(callIntent);
               })
        .SetNegativeButton("Cancel", delegate {});

    callDialog.Show();

};
```

Testing the Application

1. Click the **Play** button:



2. Click the **Translate** button to populate our CallButton with `Call 1-855-9262746`.
3. Press the **Call 1-855-9262746** button and an *Alert Dialog* should appear with a prompt to call the number:



4. 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` -

```
    callButton.Click += (object sender, EventArgs e) =>
    {
        var callDialog = new AlertDialog.Builder(this)
            .SetMessage("Call " + translatedNumber + "?")
            .SetNeutralButton("Call",
                delegate {
                    var callIntent = new Intent(Intent.ActionCall);
                    callIntent.SetData(Android.N
                    StartActivity(callIntent);
                })
            .SetNegativeButton("Cancel", delegat

        callDialog.Show();
    };
}
```

has been thrown

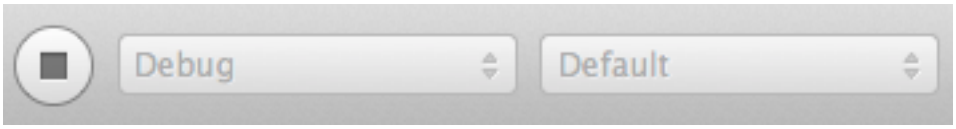
Permission Denial: starting Intent { act=android.intent.action.CALL dat=tel:x-xxx-xxxxxxx cmp=com.android.phone/.OutgoingCallBroadcaster } from ProcessRecord{b1752d38 3958:PhonewordAndroid.PhonewordAndroid/u0a10050} (pid=3958, uid=10050) requires android.permission.CALL_PHONE

Show Details

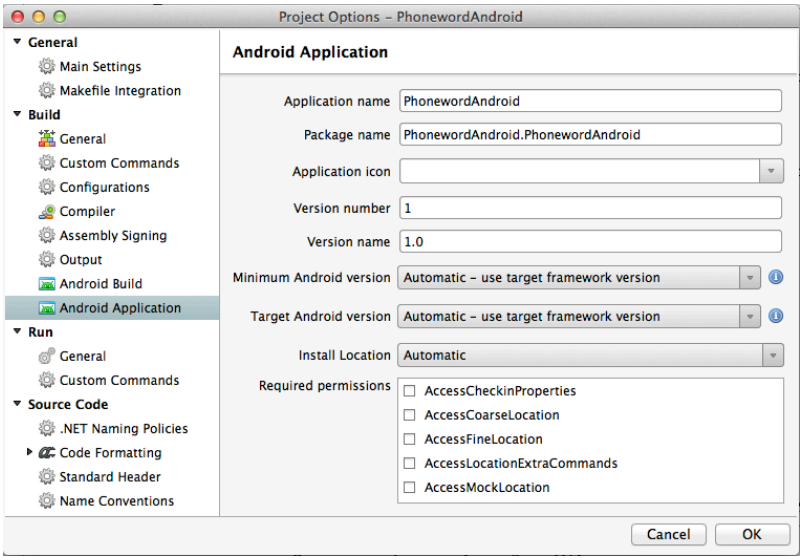
To fix this security exception, we need to request the proper privileges for our application to be able to place an outgoing call.

Requesting Make Call privileges

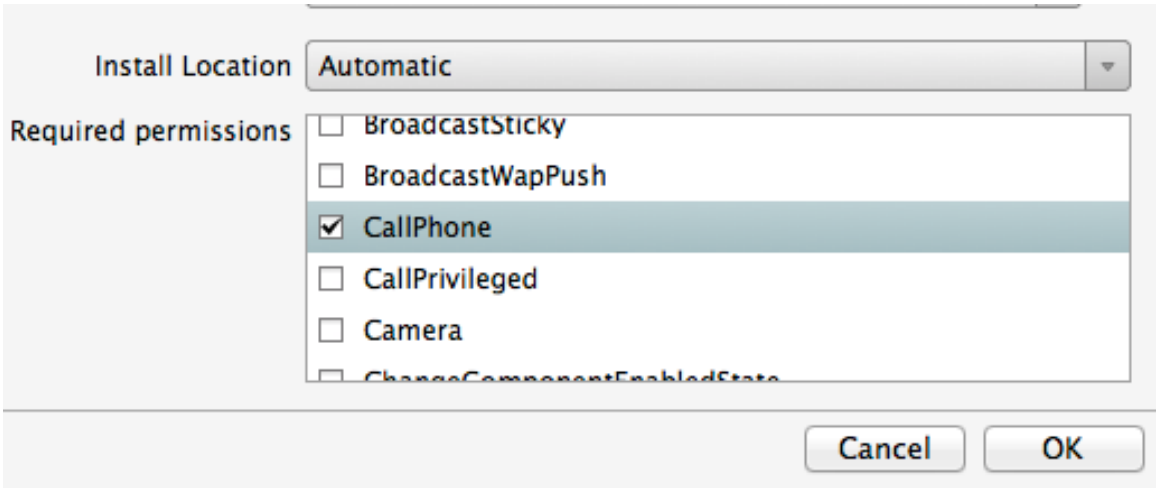
1. Stop the program by clicking the **Stop** button.



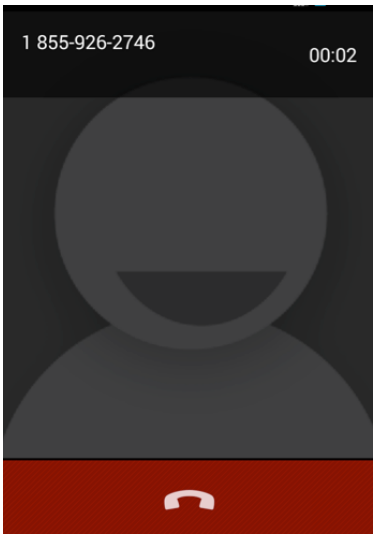
2. Double-click on the **PhonewordAndroid** project in the **Solution Pad** (or right-click the project element in the tree and select **Options**)
3. Select the "Android Application" entry under the Build node:



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



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



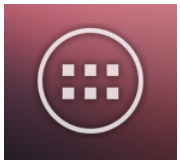
Congratulations

You've now completed your very first **Xamarin.Android** application!

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 Apps" 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. In Xamarin Studio, double-click on the `MainActivity.cs` 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**.

   It should look like:

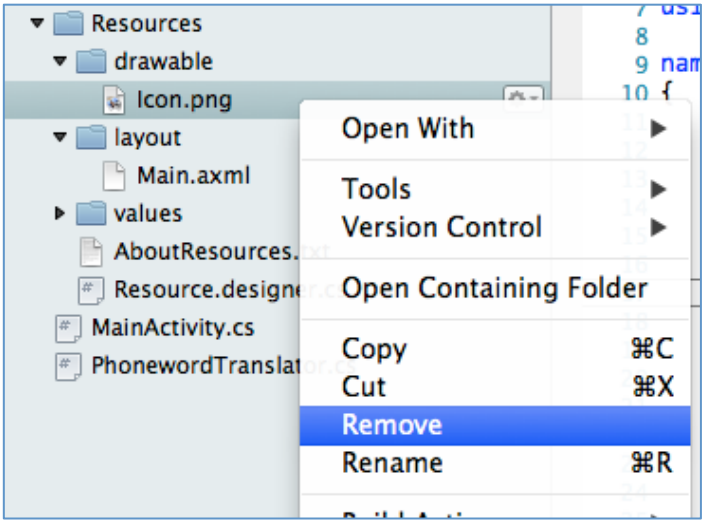
```
[Activity(Label = "Phoneword Android", MainLauncher = true)]
public class MainActivity : Activity
{
```

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.

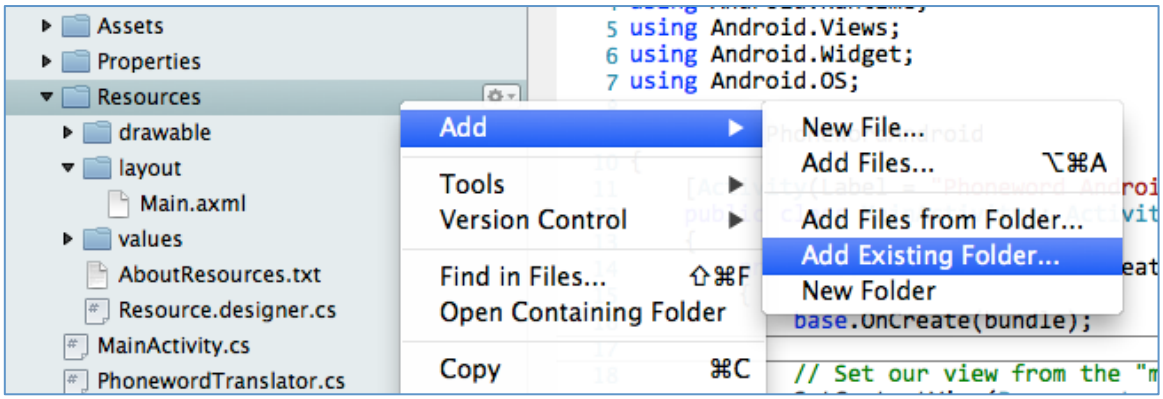
Finishing Touches – Updating the Icon

1. Let's add some custom app icons and launch images to our project.

2. Expand the **Resources** node in the **Solution Pad** and then the **drawable** folder. Remove the **Icon.png** file, which is currently in the folder by right clicking on it and selecting **Remove**. You can select **Delete** when it prompts you.



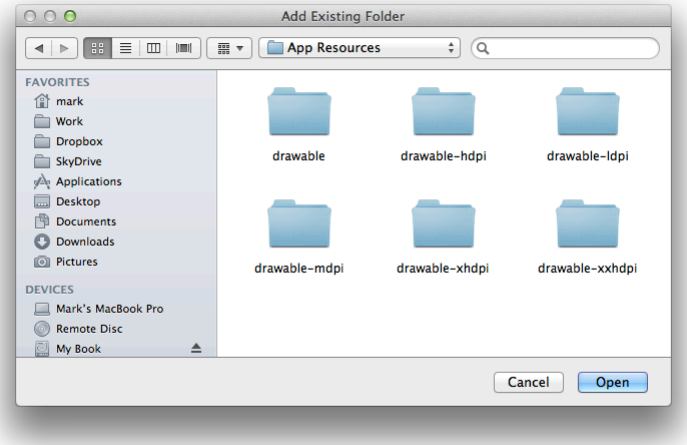
3. Next, right-click on the **Resources** folder in the **Solution Pad** and choose **Add Existing Folder**.



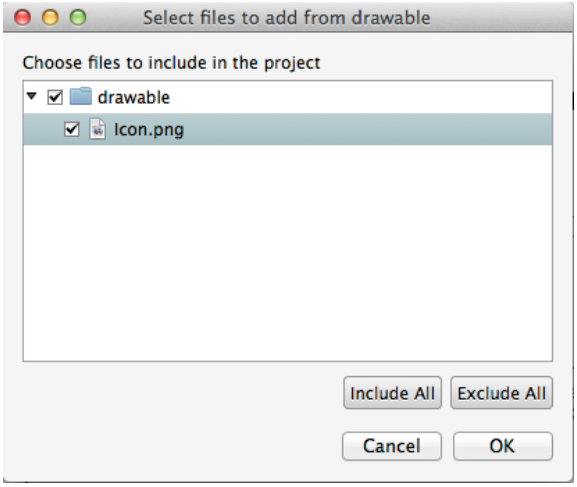
4. Navigate to

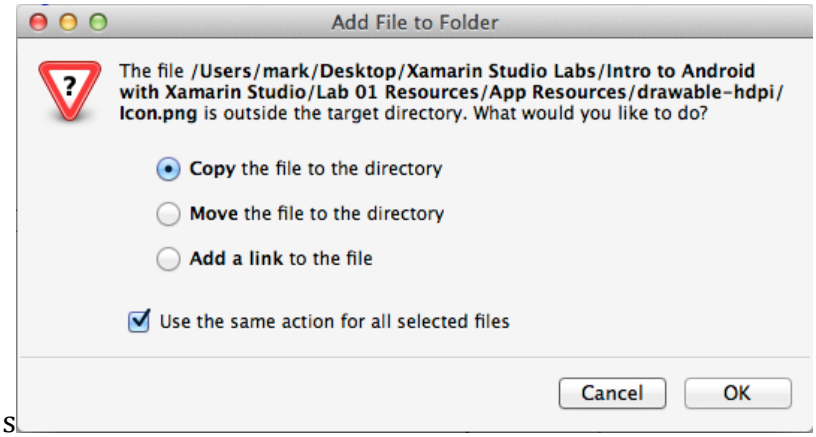
   `Xamarin Studio Labs/Intro to Android/Lab 01 Resources/App Resources/`

   And add all the `drawable-*` folders you see in there.



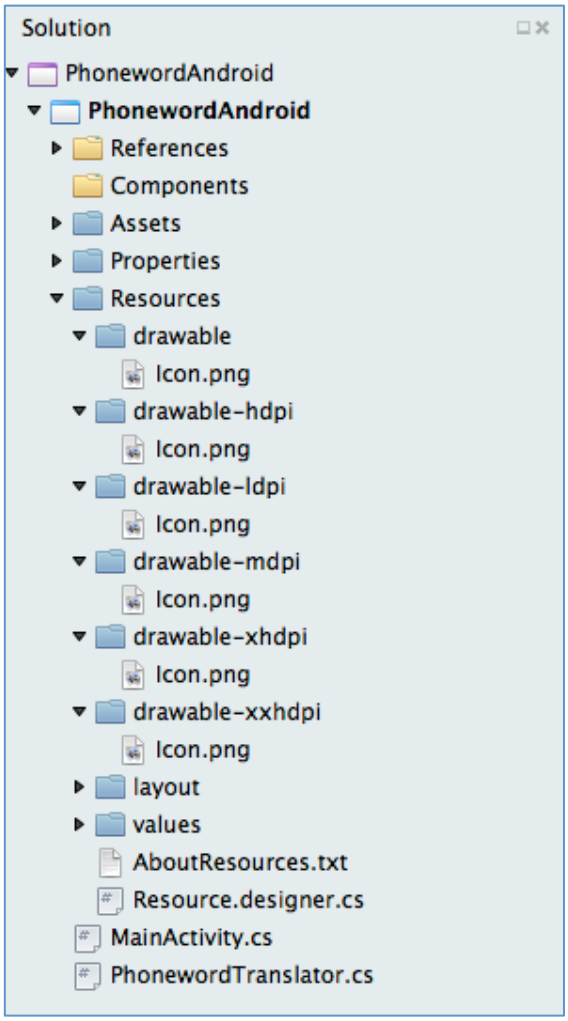
5. There are six folders and you will need to do this step for each folder (it must be done once per folder). Make sure to include all files and to **Copy** the resources into the project:



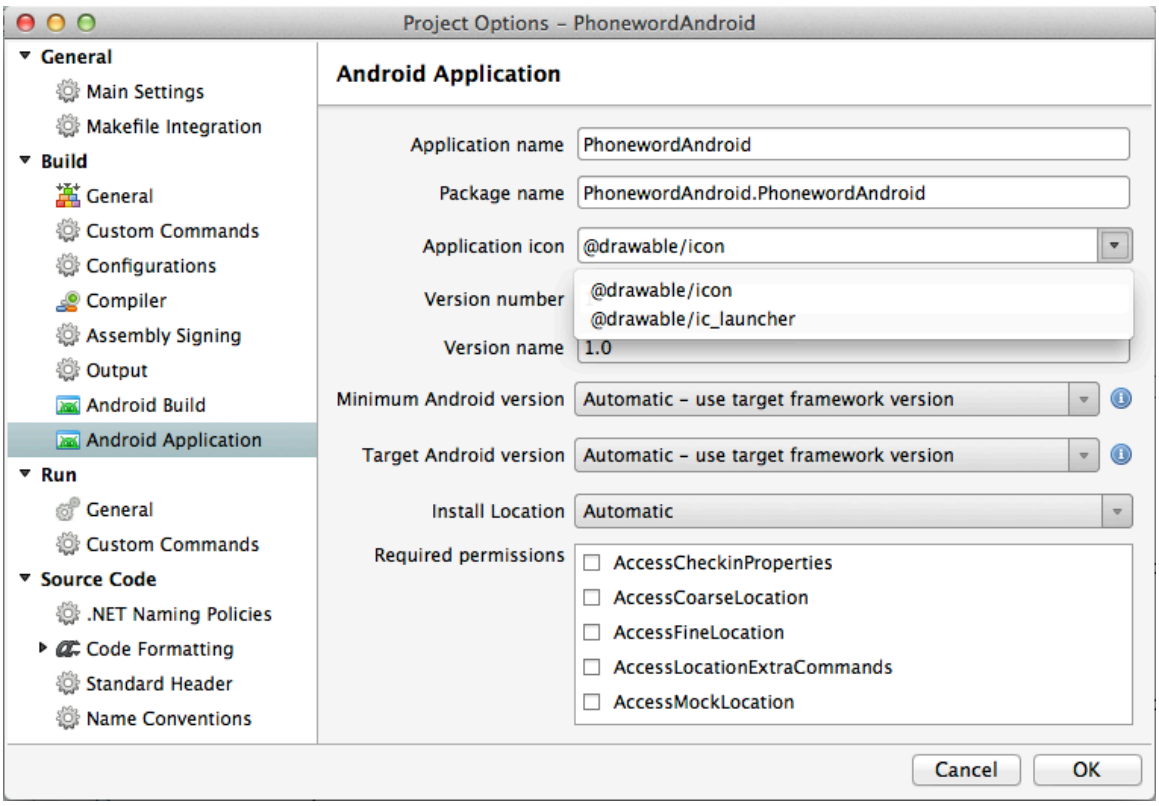


S

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



7. There are two places where we will need to specify the icon to use.
8. First, open the Project Options by either double-clicking on the **PhonewordAndroid** project element in the **Solution Pad**, or right-clicking on the project node and selection "Options".
9. In the "Android Application" element under **Build**, use the **Application Icon** drop-down to select the `@drawable/icon` entry:

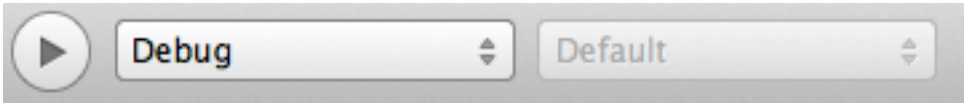


10. Next, open the `MainActivity.cs` source file and locate the Activity attribute applied to the top of the `MainActivity` class. Add a new Icon property to the attribute and set its value to `"@drawable/icon"`:

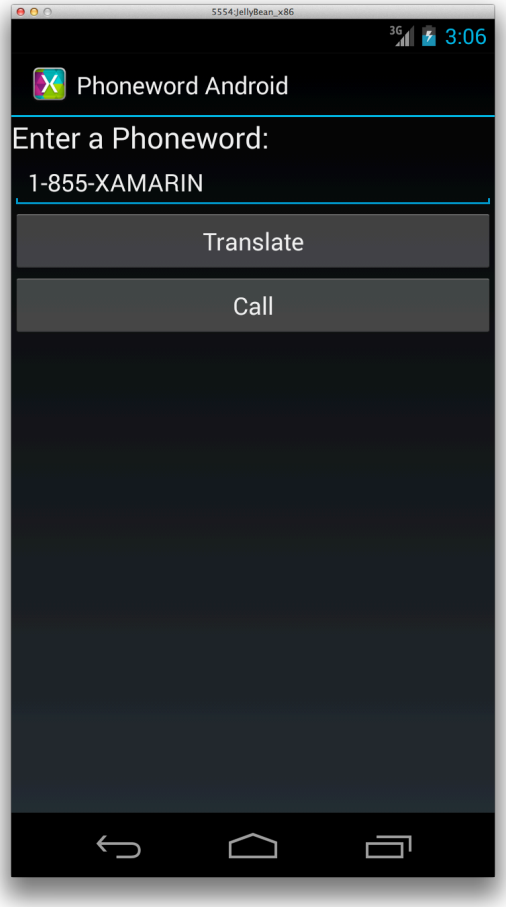
```
namespace PhonewordAndroid
{
    [Activity(Label = "Phoneword Android", MainLauncher = true,
              Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
```

Testing the Application

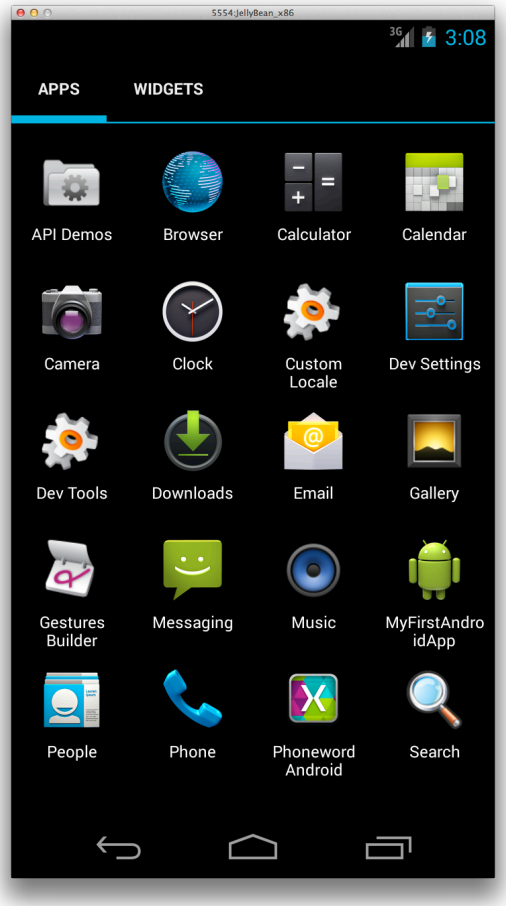
1. Build and Run the application by clicking the **Play** button:



2. You should see the new Xamarin icon in the title bar while the application executes:



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 Xamarin Studio. In the next lab, we're going to build on to this application by adding another screen.