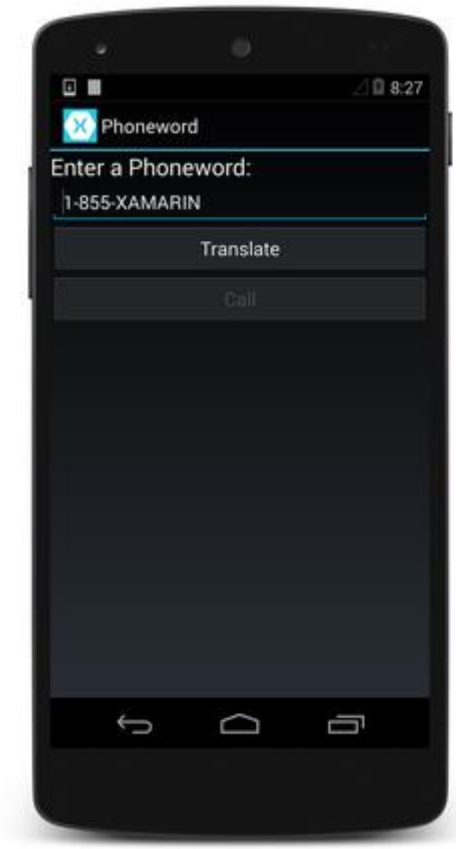
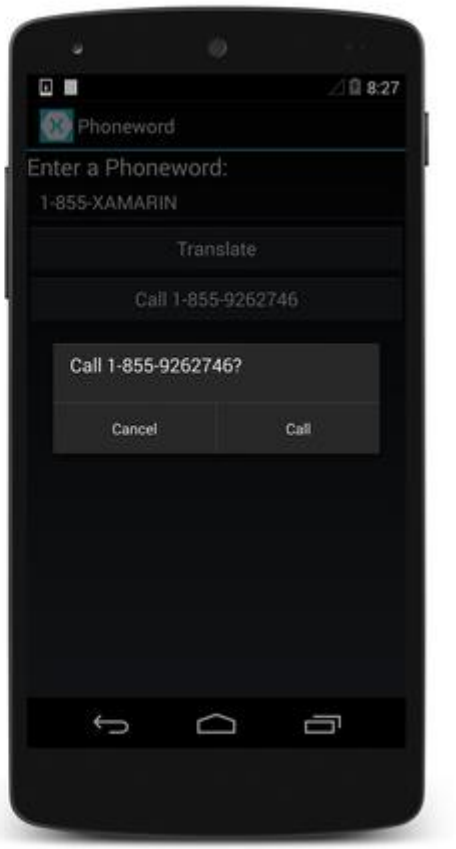
Hello, Android: Quickstart

In this two-part guide, you will build your first Xamarin.Android application (using Xamarin Studio or Visual Studio) and develop an understanding of the fundamentals of Android application development with Xamarin. Along the way, you will be introduced to the tools, concepts, and steps required to build and deploy a Xamarin.Android application.

Hello, Android Quickstart

In this walkthrough, you will create an application that translates an alphanumeric phone number (entered by the user) into a numeric phone number and then calls that number. The final application looks like this:





Requirements

To follow along with this walkthrough, you will need the following:

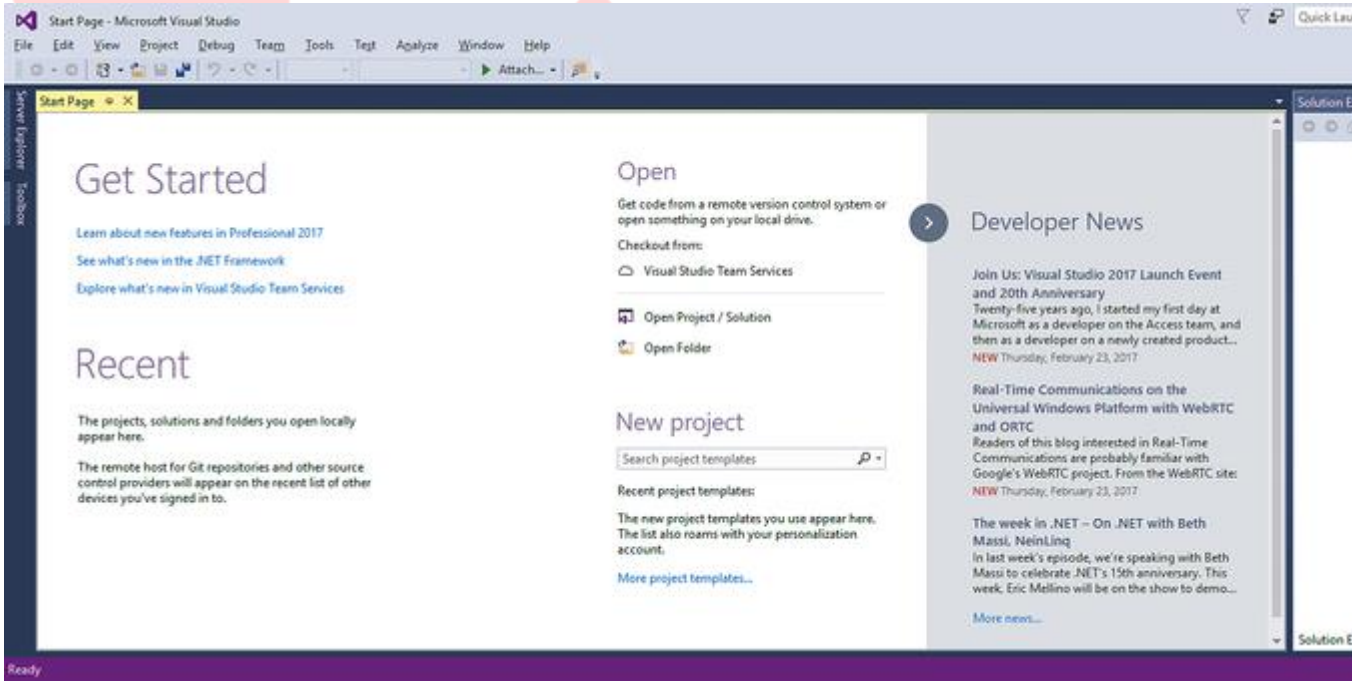
Windows 7 or later
Visual Studio 2013 Professional or later
This walkthrough assumes that the latest version of Xamarin.Android is installed and running on your platform of choice. For a guide to installing Xamarin.Android, refer to the Xamarin.Android Installation guides. Before you get started, please download and unzip the Xamarin App Icons & Launch Screens set.

Configuring Emulators

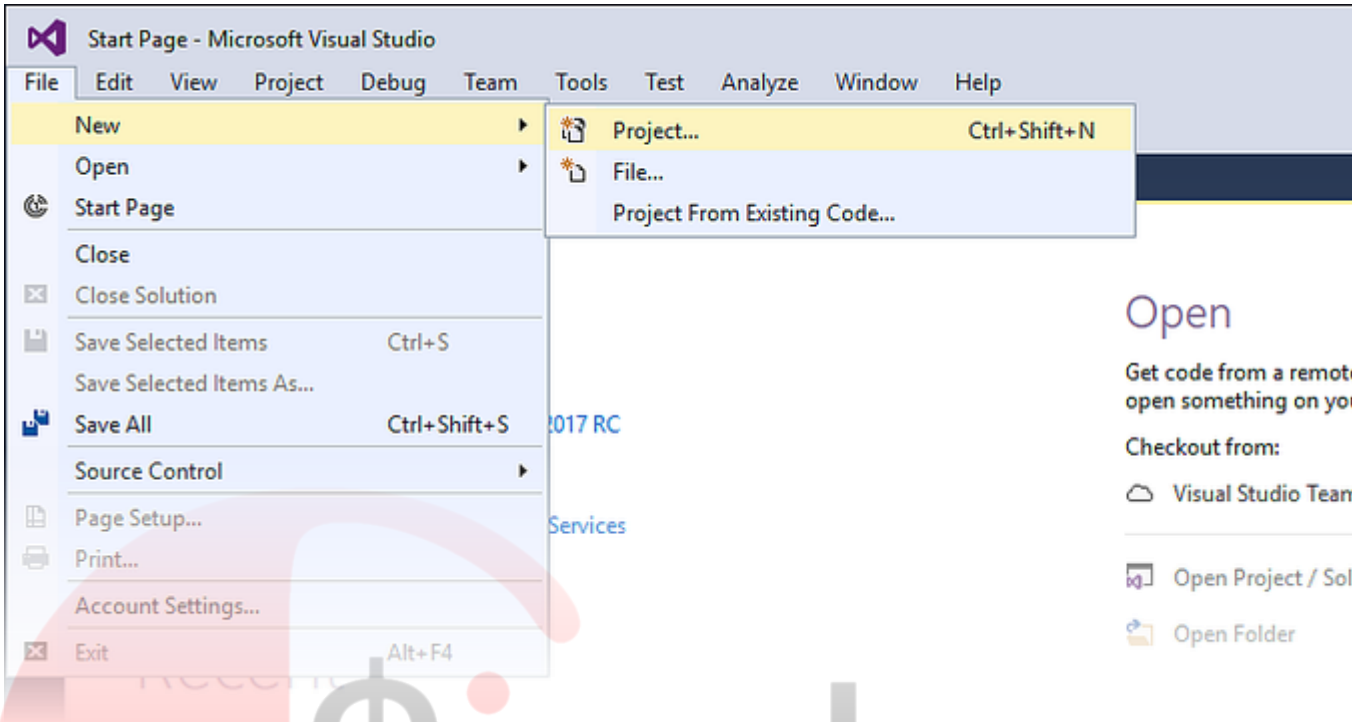
If you are using Google's Android SDK emulator, we recommend that you configure the emulator to use hardware acceleration. Instructions for configuring hardware acceleration are available in Accelerating Android Emulators with HAXM.
If you are using the Visual Studio Android Emulator, Hyper-V must be enabled on your computer. For more information about configuring the Visual Studio Android Emulator, see System Requirements for the Visual Studio Emulator for Android.

Walkthrough

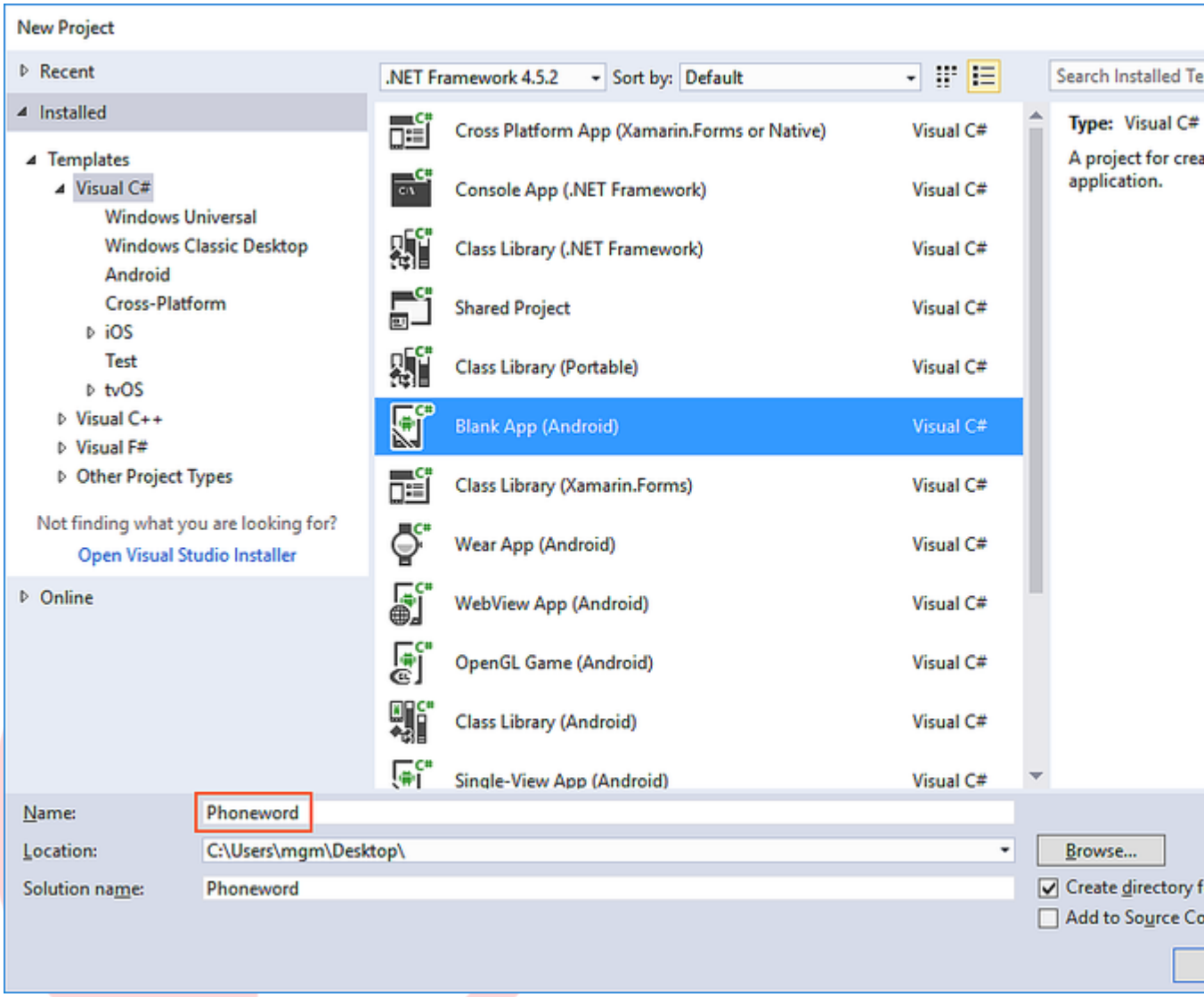
Start Visual Studio:



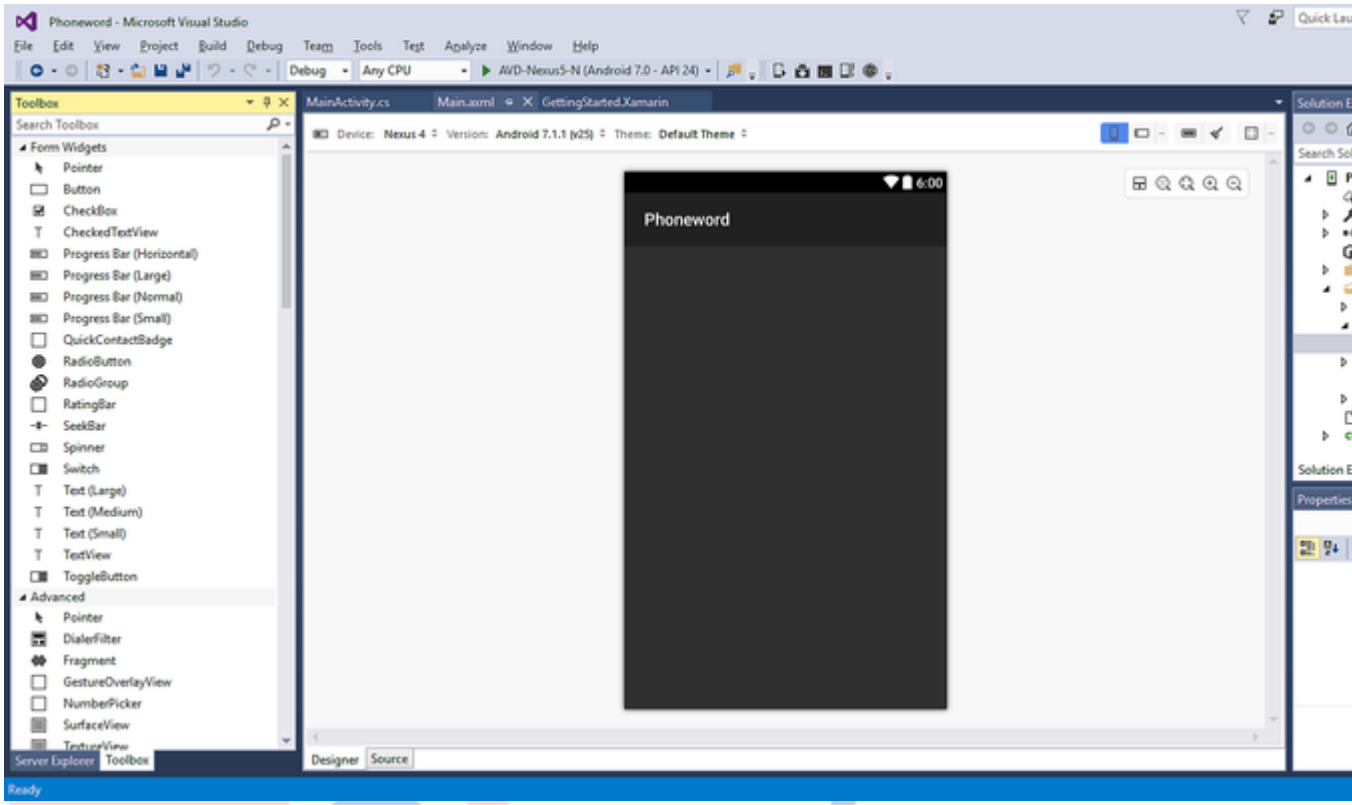
Click **File > New > Project** to create a new project:



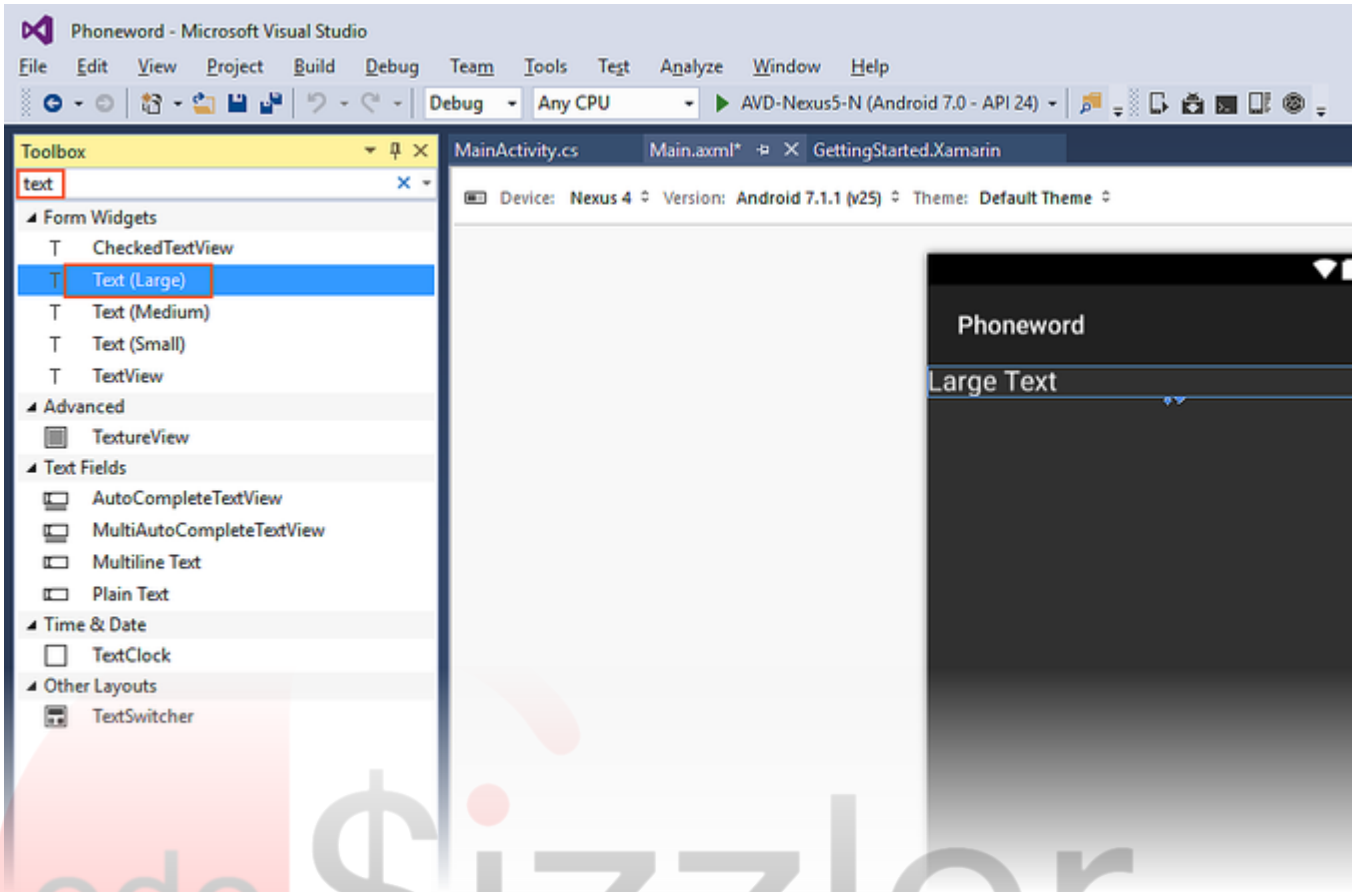
In the **New Project** dialog, click the **Blank App (Android)** template. Name the new project `Phoneword`. Click **OK** to create the new project:



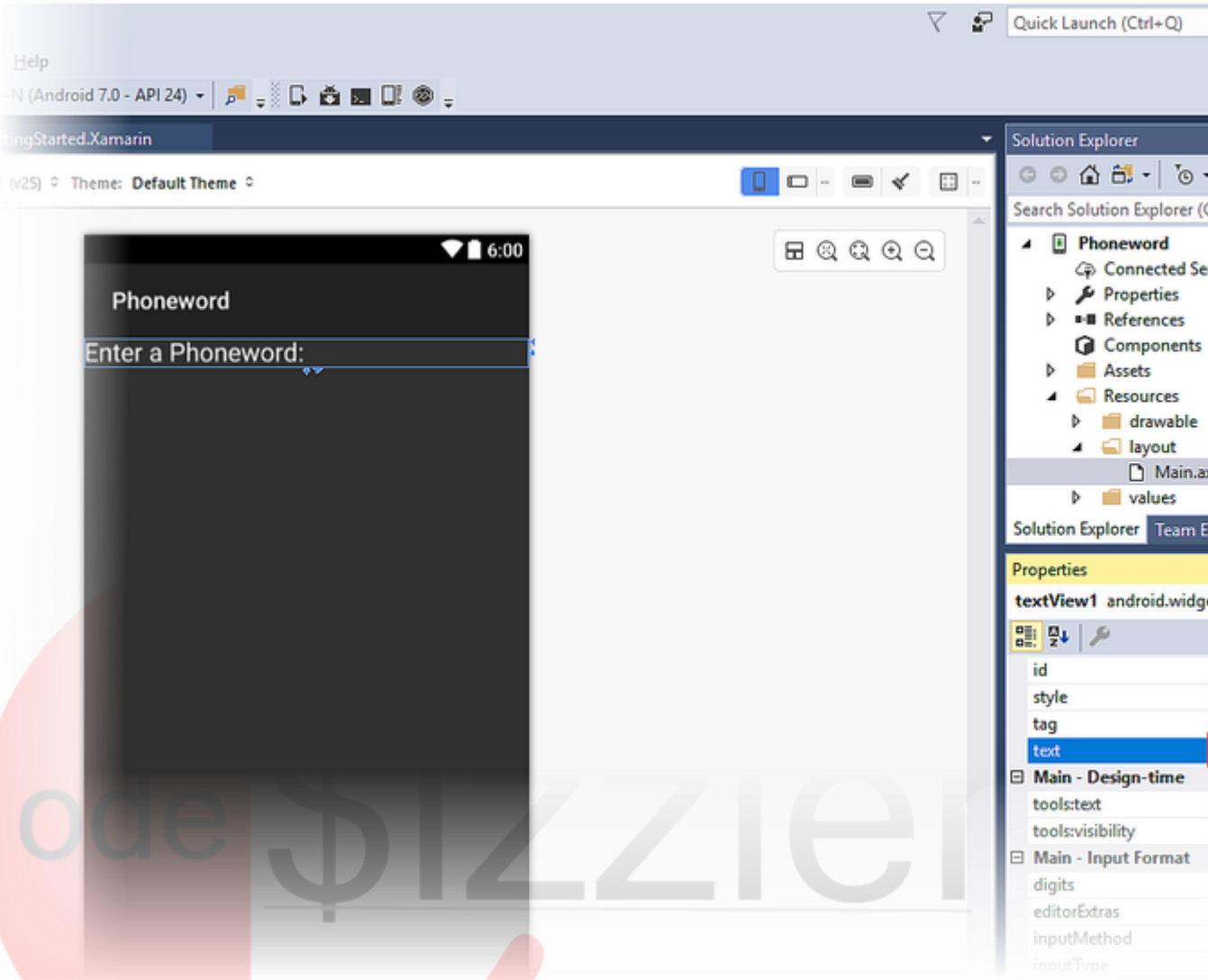
After the new project is created, expand the **Resources** folder and then the **layout** folder in the **Solution Explorer**. Double-click **Main.axml** to open it in the Android Designer. This is the layout file for the app's screen:



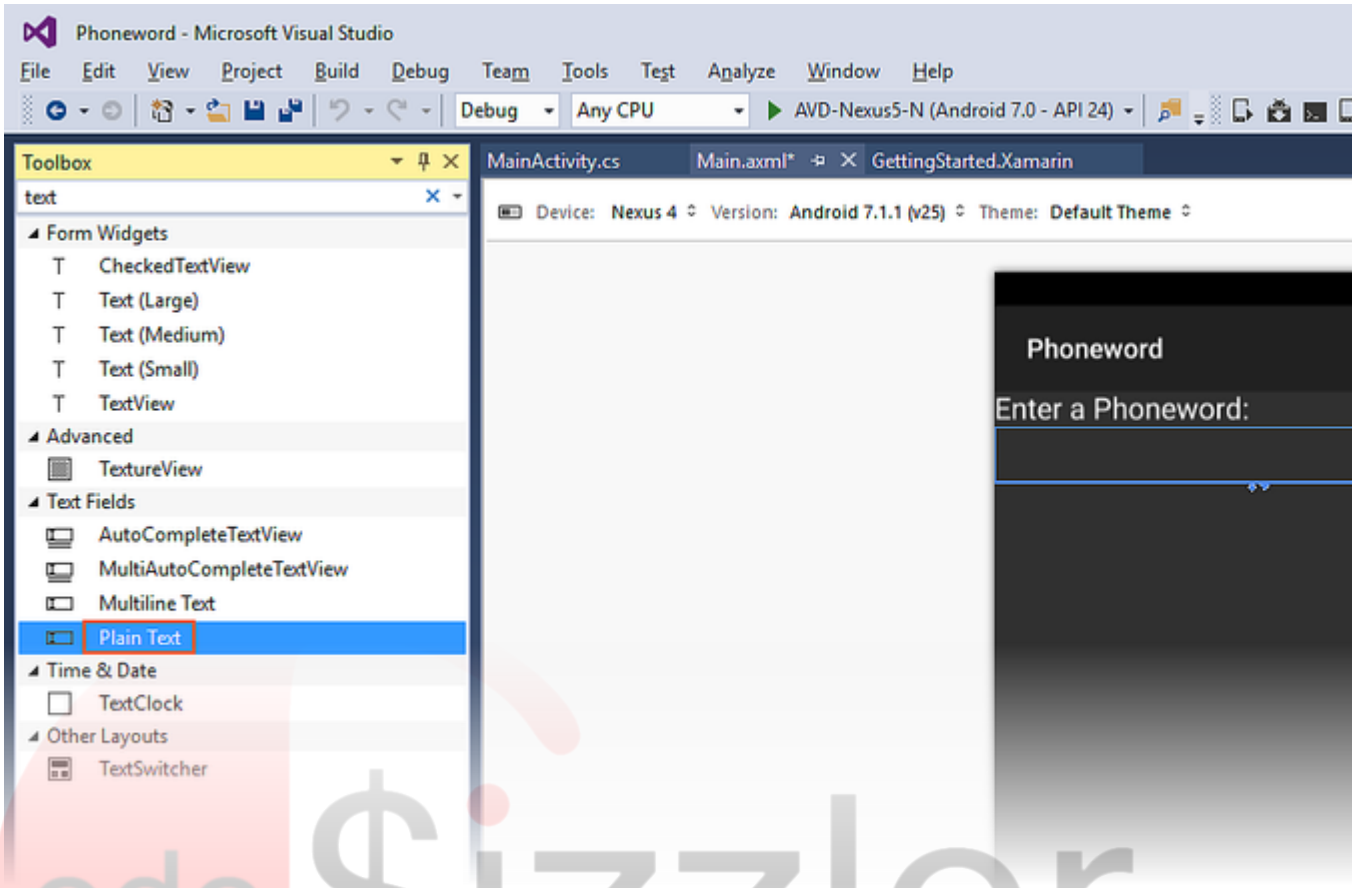
From the **Toolbox** (the area on the left), enter `text` into the search field and drag a **Text (Large)** widget onto the design surface (the area in the centre):



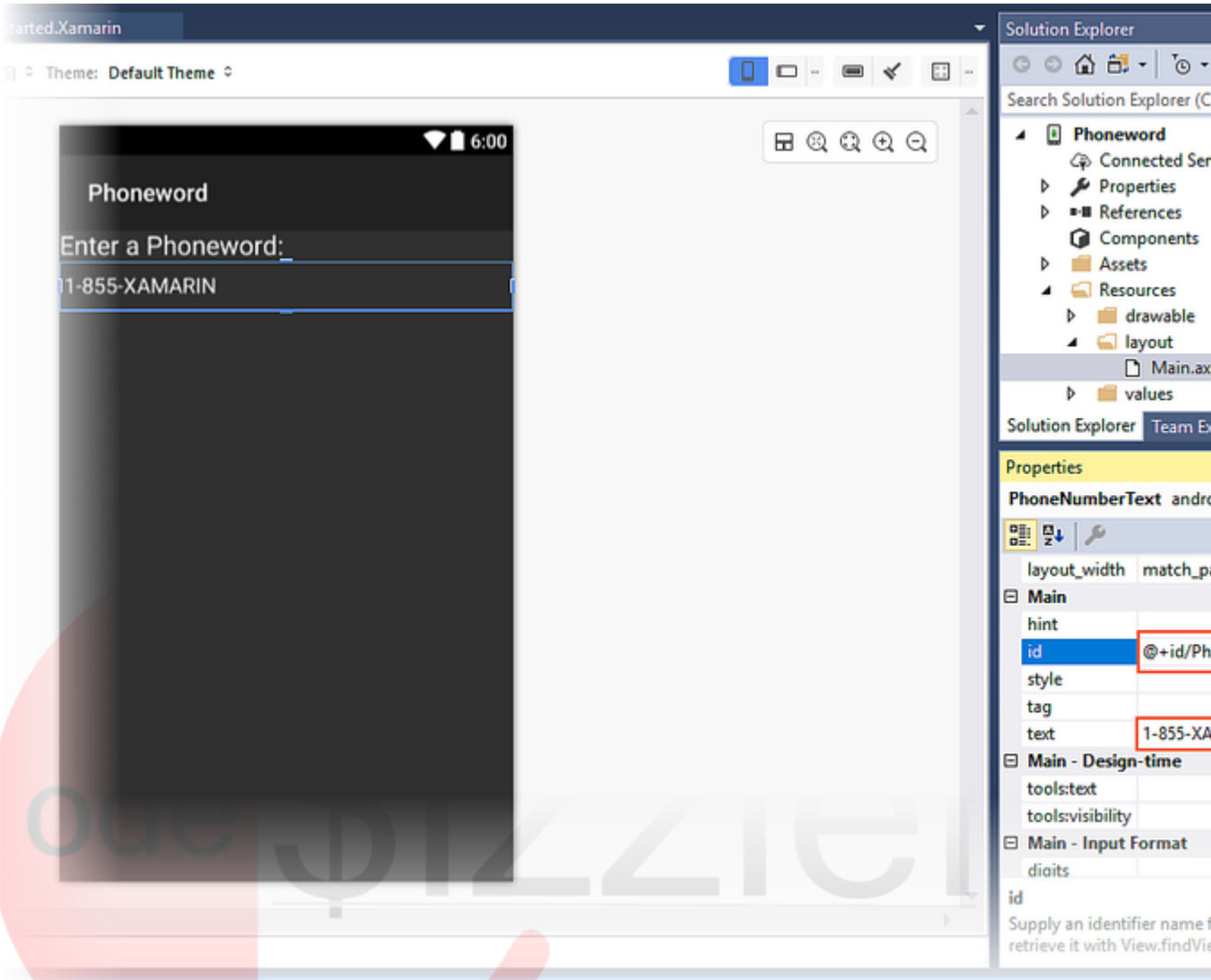
With the **Text (Large)** control selected on the design surface, use the **Properties** pane to change the `text` property of the **Text (Large)** widget to `Enter a Phoneword:` as shown here:



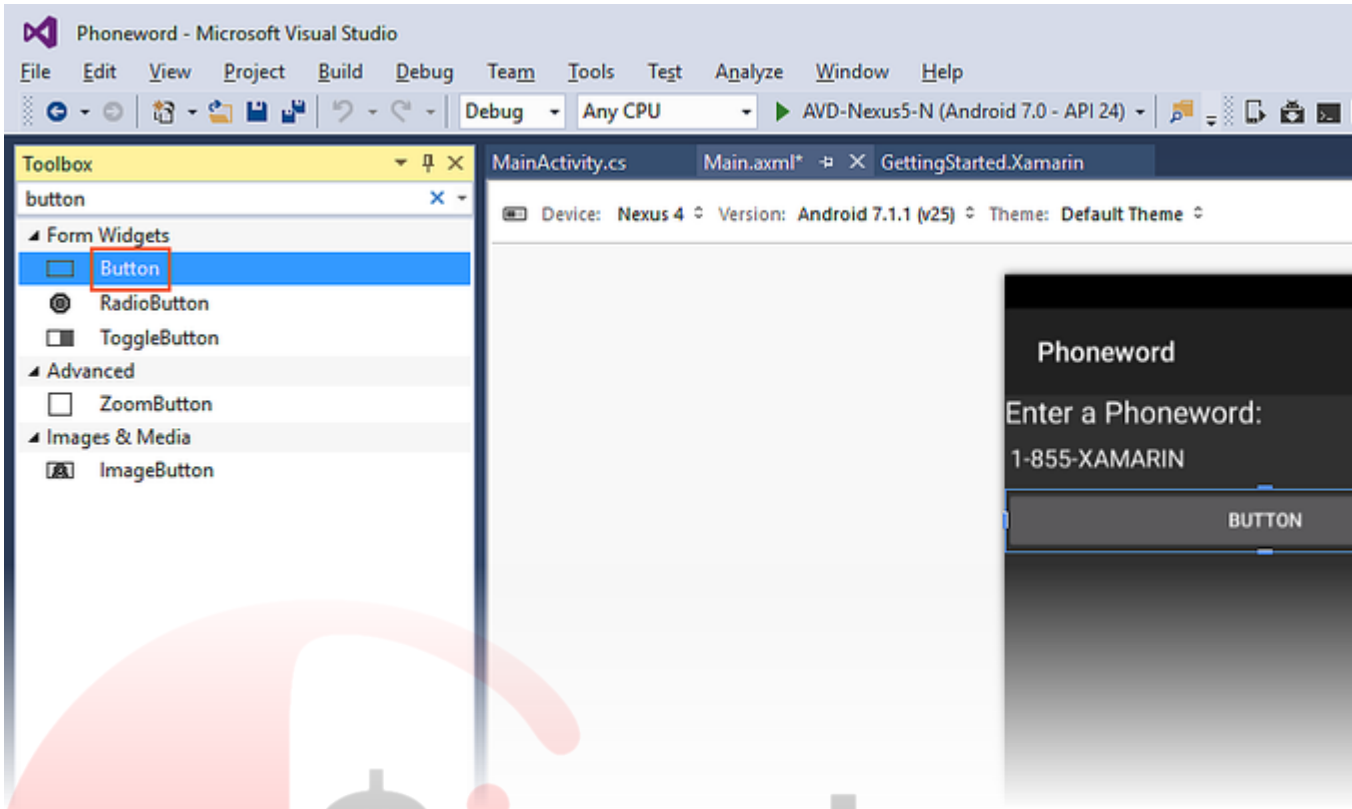
Drag a **Plain Text** widget from the **Toolbox** to the design surface and place it underneath the **Text (Large)** widget:



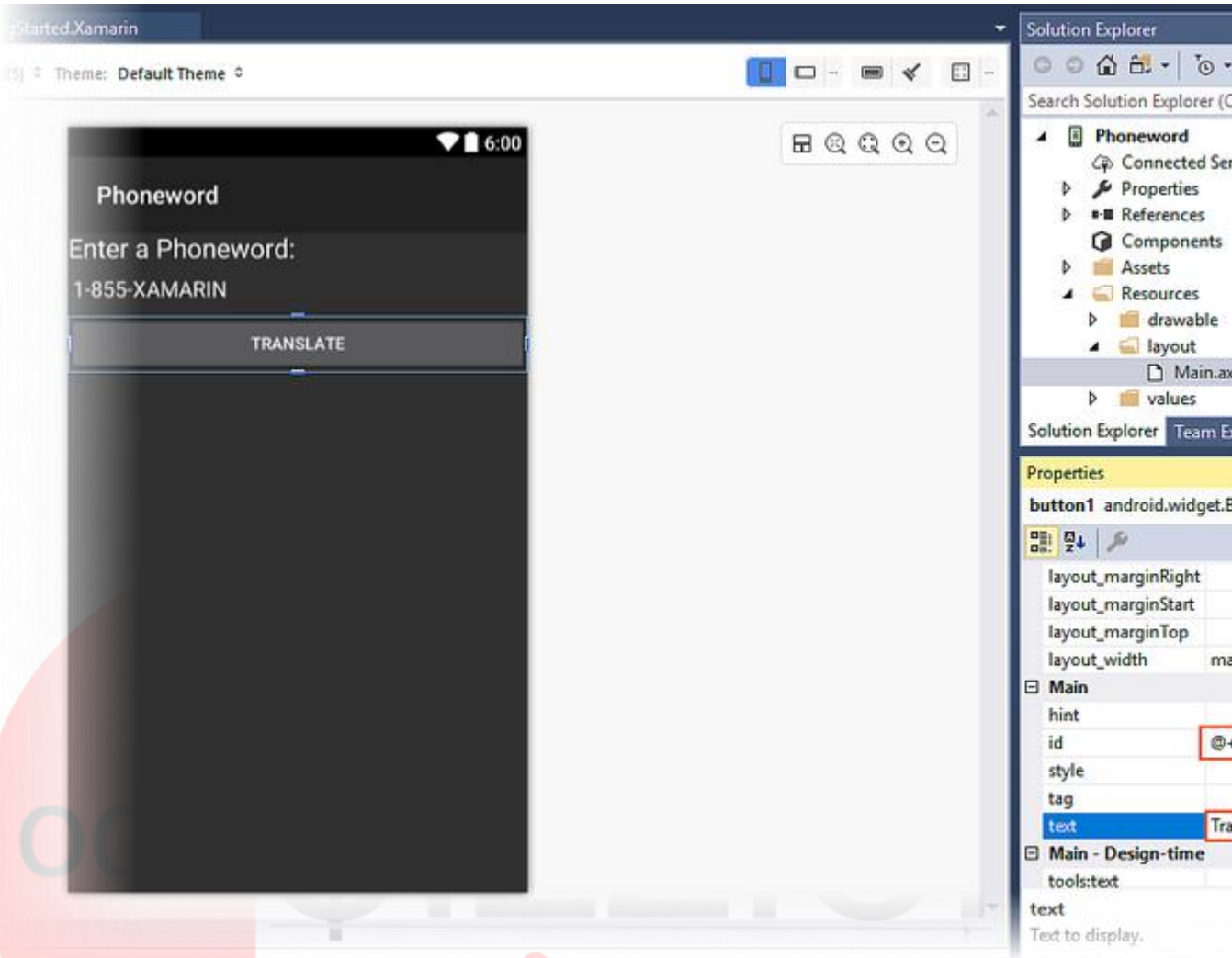
With the **Plain Text** widget selected on the design surface, use the **Properties** pane to change the `id` property of the**Plain Text** widget to `@+id/PhoneNumberText` and change the `text` property to `1-855-XAMARIN`:



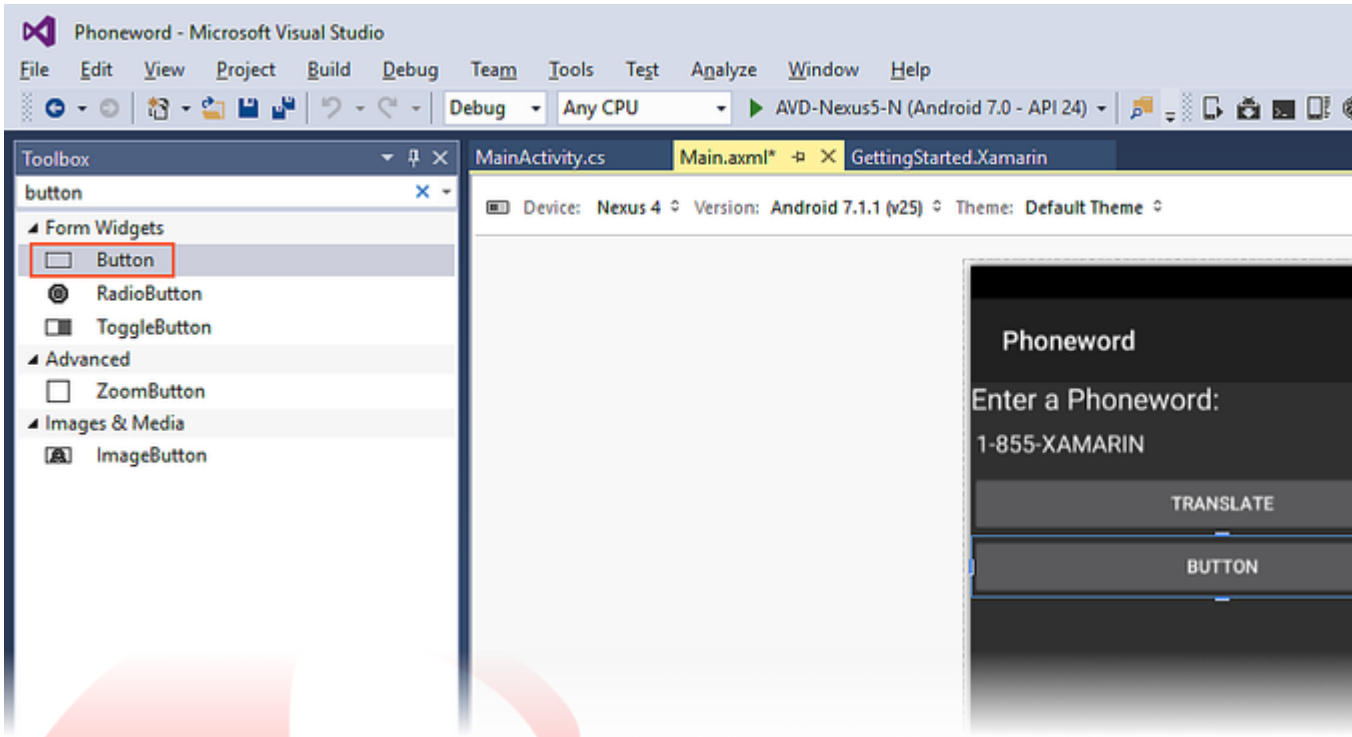
Drag a **Button** from the **Toolbox** to the design surface and place it underneath the **Plain Text** widget:



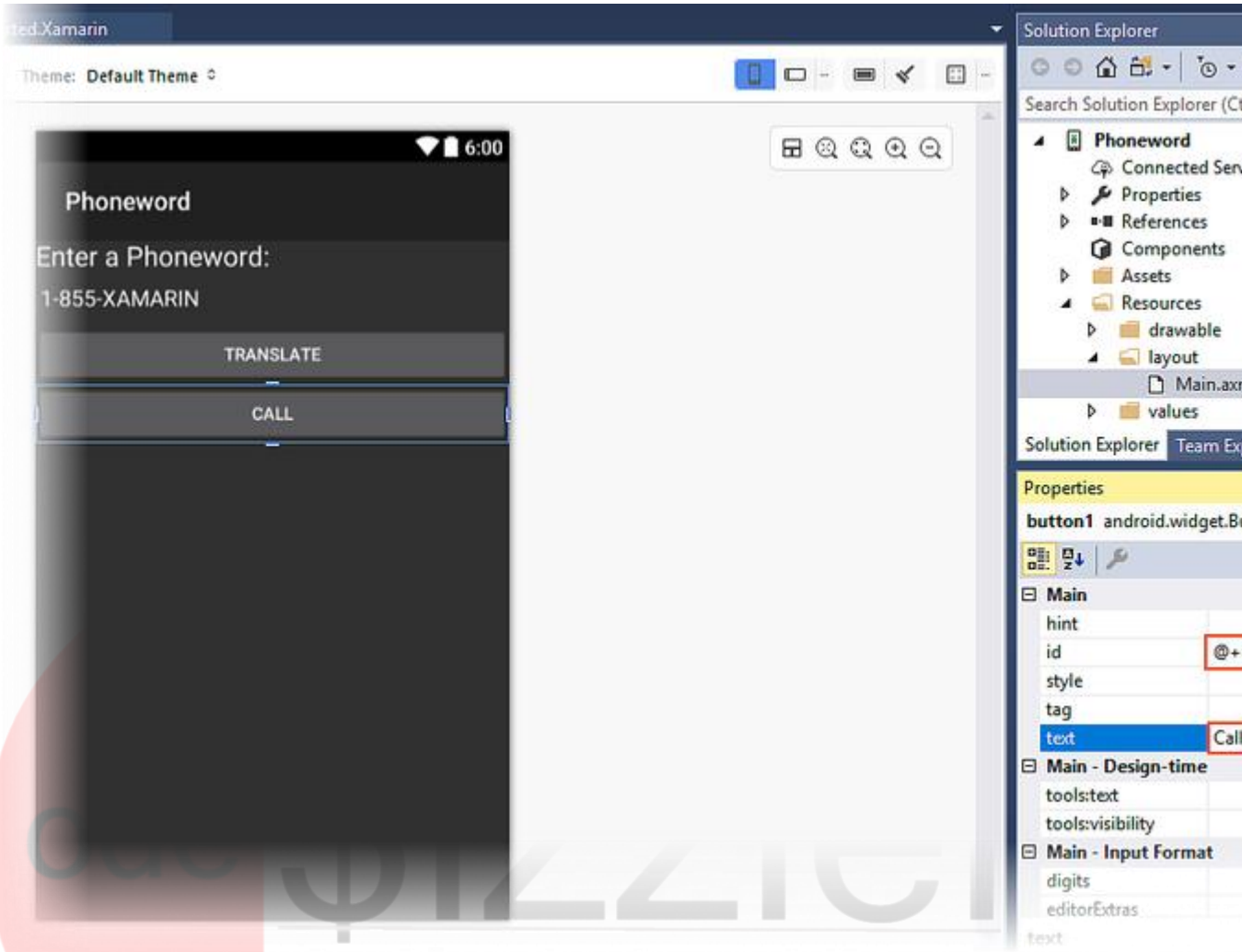
With the **Button** selected on the design surface, use the **Properties** pane to change the `id` property of the **Button** to `@+id/TranslateButton` and change the `text` property to `Translate`:



Drag a second **Button** from the **Toolbox** to the design surface and place it underneath the **Translate** button:

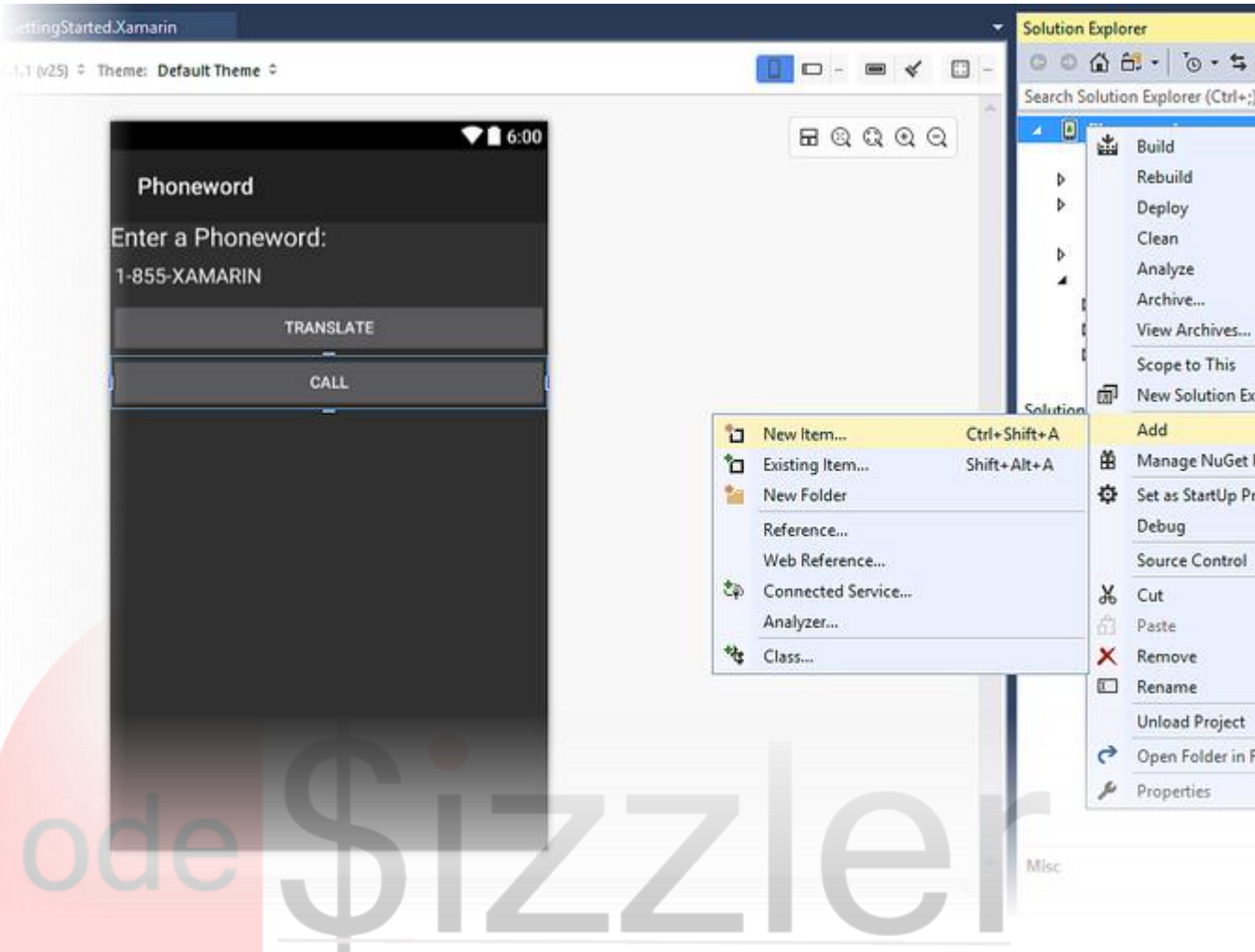


With the new **Button** control selected on the design surface, use the **Properties** pane to change the `id` property of the**Button** to `@+id/CallButton` and change the `text` property to `Call`:

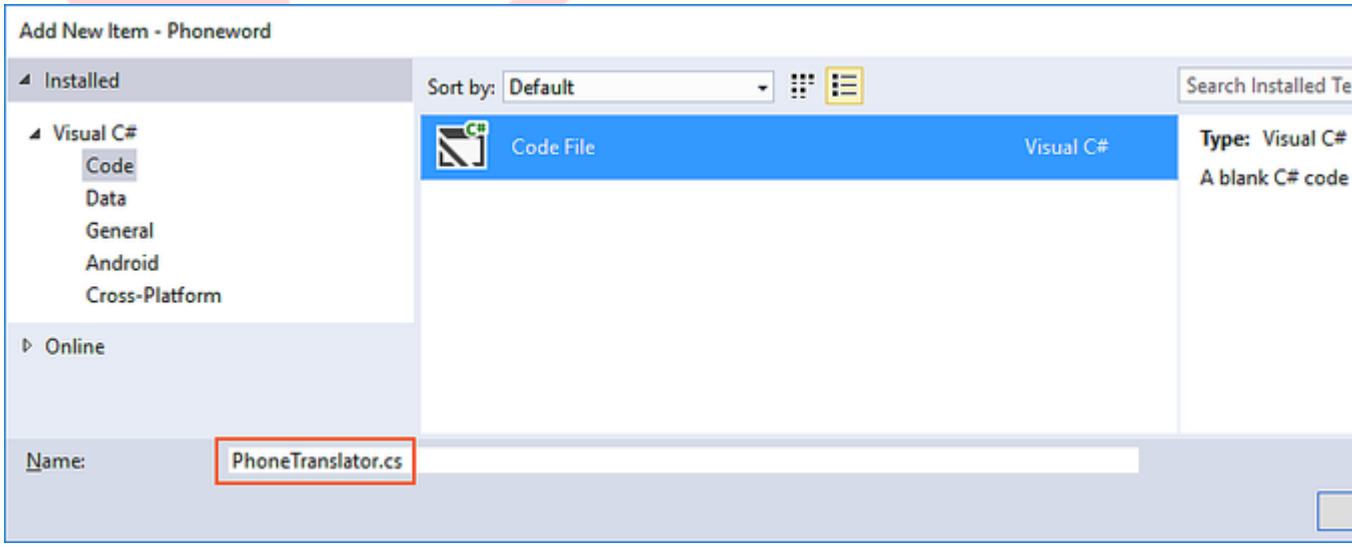


Save your work by pressing **CTRL+S**.

The next step is to add some code to translate phone numbers from alphanumeric to numeric. Add a new file to the project by right-clicking the **Phoneword** project in the **Solution Explorer** pane and choosing **Add > New Item...** as shown below:



In the **Add New Item** dialog, select **Visual C# > Code** and name the new code file **PhoneTranslator.cs**:



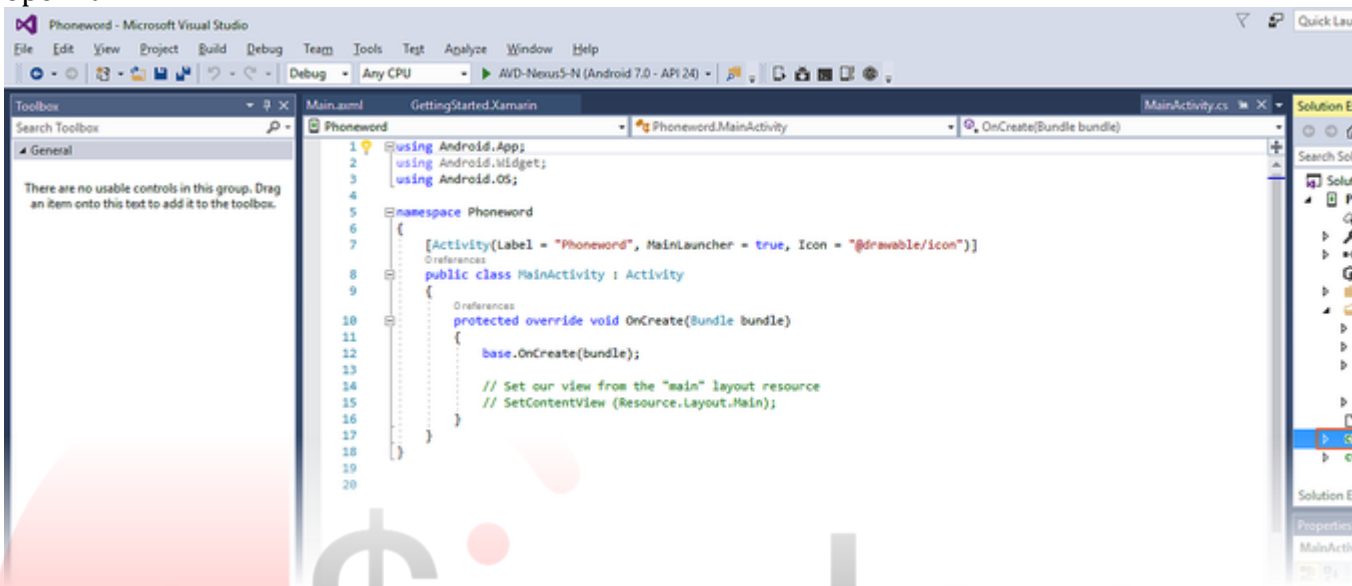
This creates a new empty C# class. Insert the following code into this file:

```
using System.Text;
using System;
namespace Core
{
    public static class PhonewordTranslator
    {
        public static string ToNumber(string raw)
        {
            if (string.IsNullOrWhiteSpace(raw))
                return "";
            else
                raw = raw.ToUpperInvariant();

            var newNumber = new StringBuilder();
            foreach (var c in raw)
            {
                if (" -0123456789".Contains(c))
                    newNumber.Append(c);
                else {
                    var result = TranslateToNumber(c);
                    if (result != null)
                        newNumber.Append(result);
                }
                // otherwise we've skipped a non-numeric char
            }
            return newNumber.ToString();
        }
        static bool Contains (this string keyString, char c)
        {
            return keyString.IndexOf(c) >= 0;
        }
        static int? TranslateToNumber(char c)
        {
            if ("ABC".Contains(c))
                return 2;
            else if ("DEF".Contains(c))
                return 3;
            else if ("GHI".Contains(c))
                return 4;
            else if ("JKL".Contains(c))
                return 5;
            else if ("MNO".Contains(c))
                return 6;
            else if ("PQRS".Contains(c))
                return 7;
            else if ("TUV".Contains(c))
                return 8;
            else if ("WXYZ".Contains(c))
                return 9;
            return null;
        }
    }
}
```

Save the changes to the **PhoneTranslator.cs** file by clicking **File > Save** (or by pressing **CTRL+S**), then close the file.

The next step is to add code to wire up the user interface by inserting backing code into the `MainActivity` class. In the **Solution Explorer**, find **MainActivity.cs** and open it:



Begin by wiring up the **Translate** button. In the `MainActivity` class, find the `OnCreate` method. The next step is to add the button code inside `OnCreate`, below the `base.OnCreate(bundle)` and `SetContentView (Resource.Layout.Main)` calls. First, modify the template code so that the `OnCreate` method resembles the following:

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

namespace Phoneword
{
    [Activity (Label = "Phoneword", MainLauncher = true, Icon =
"@drawable/icon")]
    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);

            // New code will go here
        }
    }
```

```
}
```

Get a reference to the controls that were created in the layout file via the Android Designer. Add the following code inside the `OnCreate` method, after the call to `SetContentView`:

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

Add code that responds to user presses of the **Translate** button. Add the following code to the `OnCreate` method (after the lines added in the previous step):

```
// Disable the "Call" button
callButton.Enabled = false;

// Add code to translate number
string translatedNumber = string.Empty;

translateButton.Click += (object sender, EventArgs e) =>
{
    // Translate user's alphanumeric phone number to numeric
    translatedNumber                                                      =
Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (String.IsNullOrWhiteSpace(translatedNumber))
    {
        callButton.Text = "Call";
        callButton.Enabled = false;
    }
    else
    {
        callButton.Text = "Call " + translatedNumber;
        callButton.Enabled = true;
    }
};
```

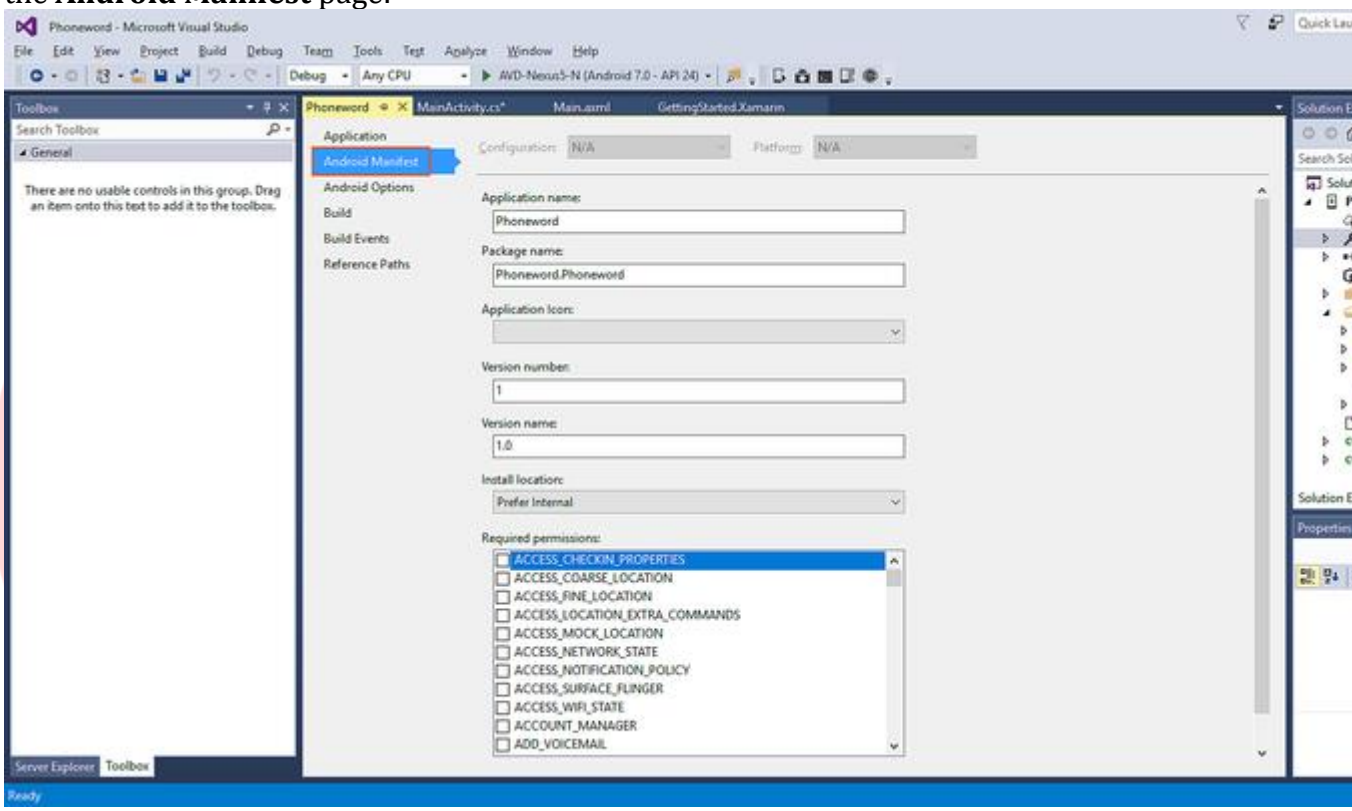
Add code that responds to user presses of the **Call** button. Place the following code below the code for the **Translate**button:

```
callButton.Click += (object sender, EventArgs 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));
```

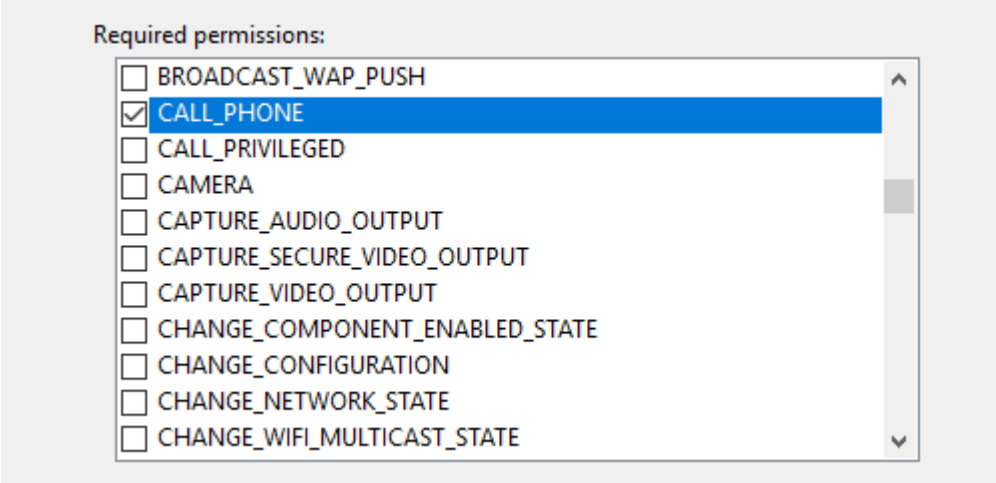
```
            StartActivity(callIntent);
        });
    callDialog.SetNegativeButton("Cancel", delegate { });

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

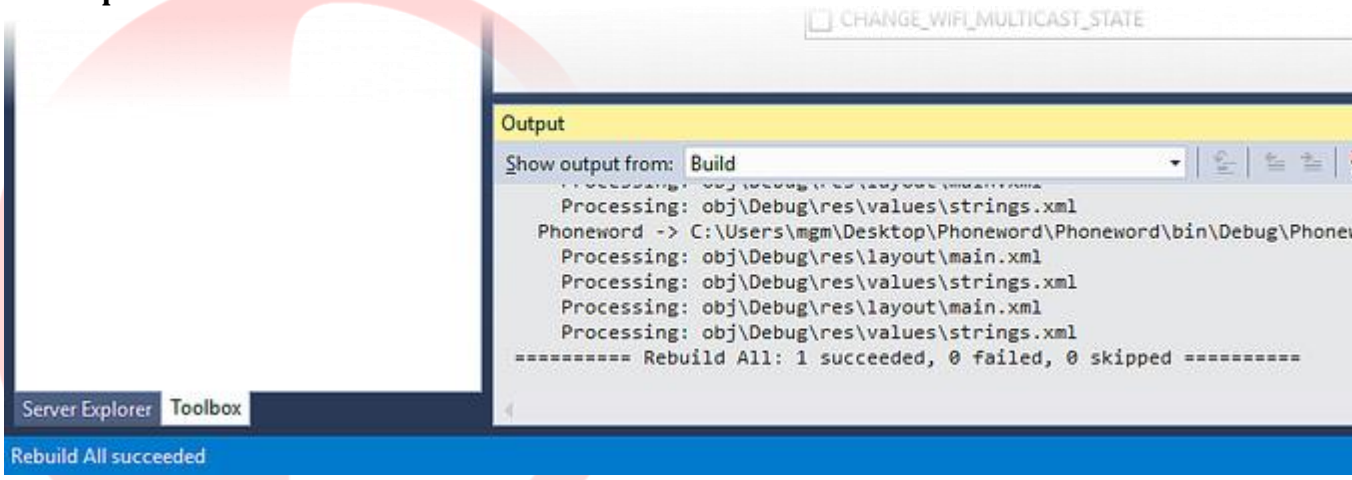
Finally, it's time give the application permission to place a phone call. App permissions can be edited in the Android Manifest. In the **Solution Explorer**, open the Android Manifest by double-clicking **Properties** under **Phoneword**, then select the **Android Manifest** page:



Under **Required Permissions**, enable the **CALL_PHONE** permission:



Save your work by selecting **File > Save All** (or by pressing **CTRL-SHIFT-S**) and build the application by selecting **Build > Rebuild Solution** (or by pressing **CTRL-SHIFT-B**). If the application compiles, you will get a success message in the**Output** window and in the bottom left corner of Visual Studio:



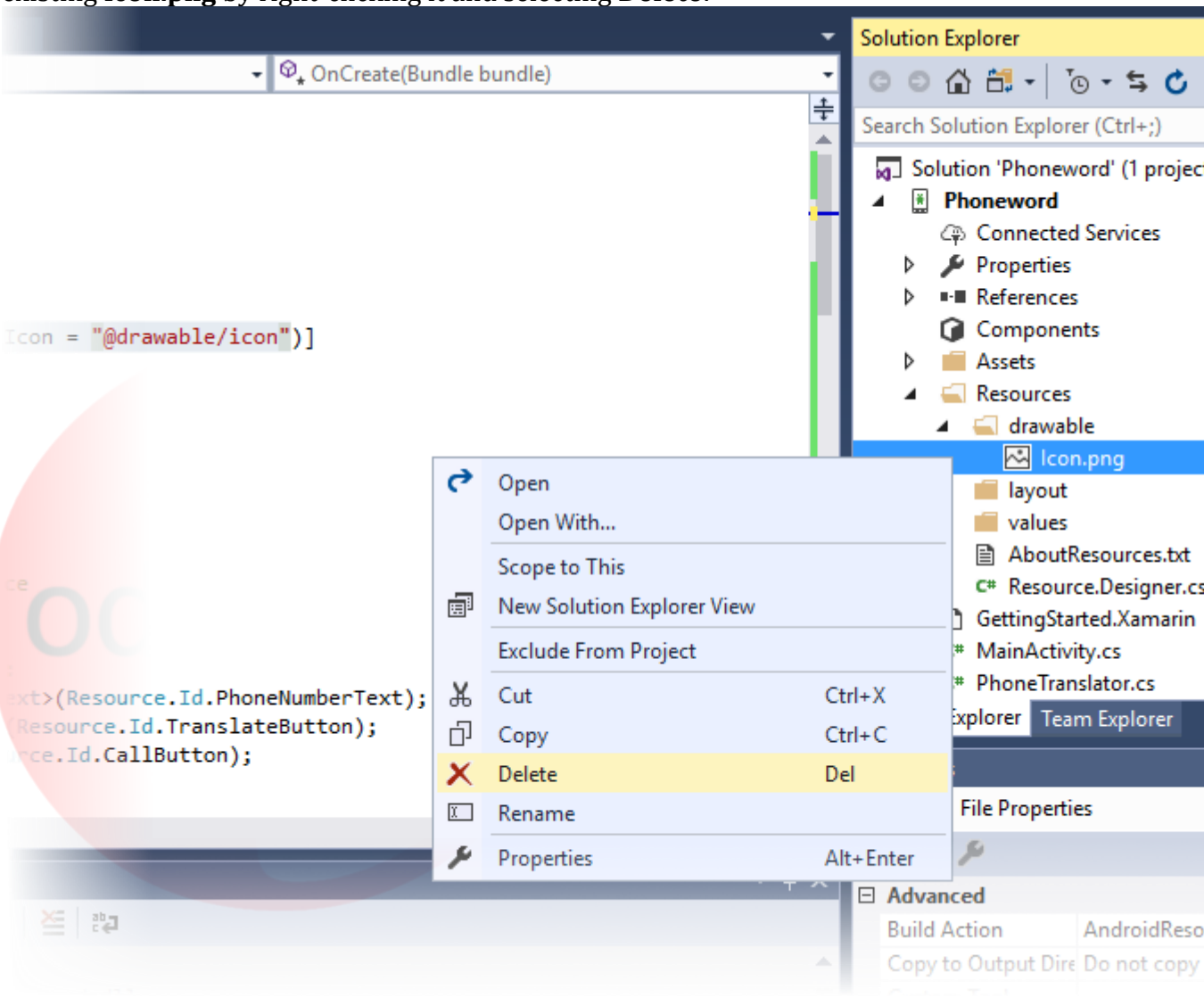
If there are errors, go through the previous steps and correct any mistakes until the application builds successfully. If you get a build error such as, *Resource does not exist in the current context*, verify that the namespace name in **MainActivity.cs** matches the project name (`Phoneword`) and then completely rebuild the solution. If you still get build errors, verify that you have installed the latest Xamarin.Android updates.

You should now have a working application – it's time to add the finishing touches! In **MainActivity.cs**, edit the `Label`for the `MainActivity`. The `Label` is what Android displays at the top of the screen to let users know where they are in the application. At the top of the `MainActivity` class, change the `Label` to `Phone Word` as shown here:

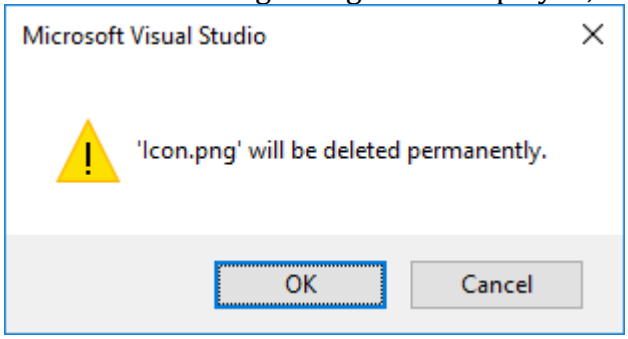
```
namespace Phoneword
{
    [Activity (Label = "Phone Word", MainLauncher = true, , Icon =
"@drawable/icon")]
    public class MainActivity : Activity
    {
        ...
    }
```

```
}
```

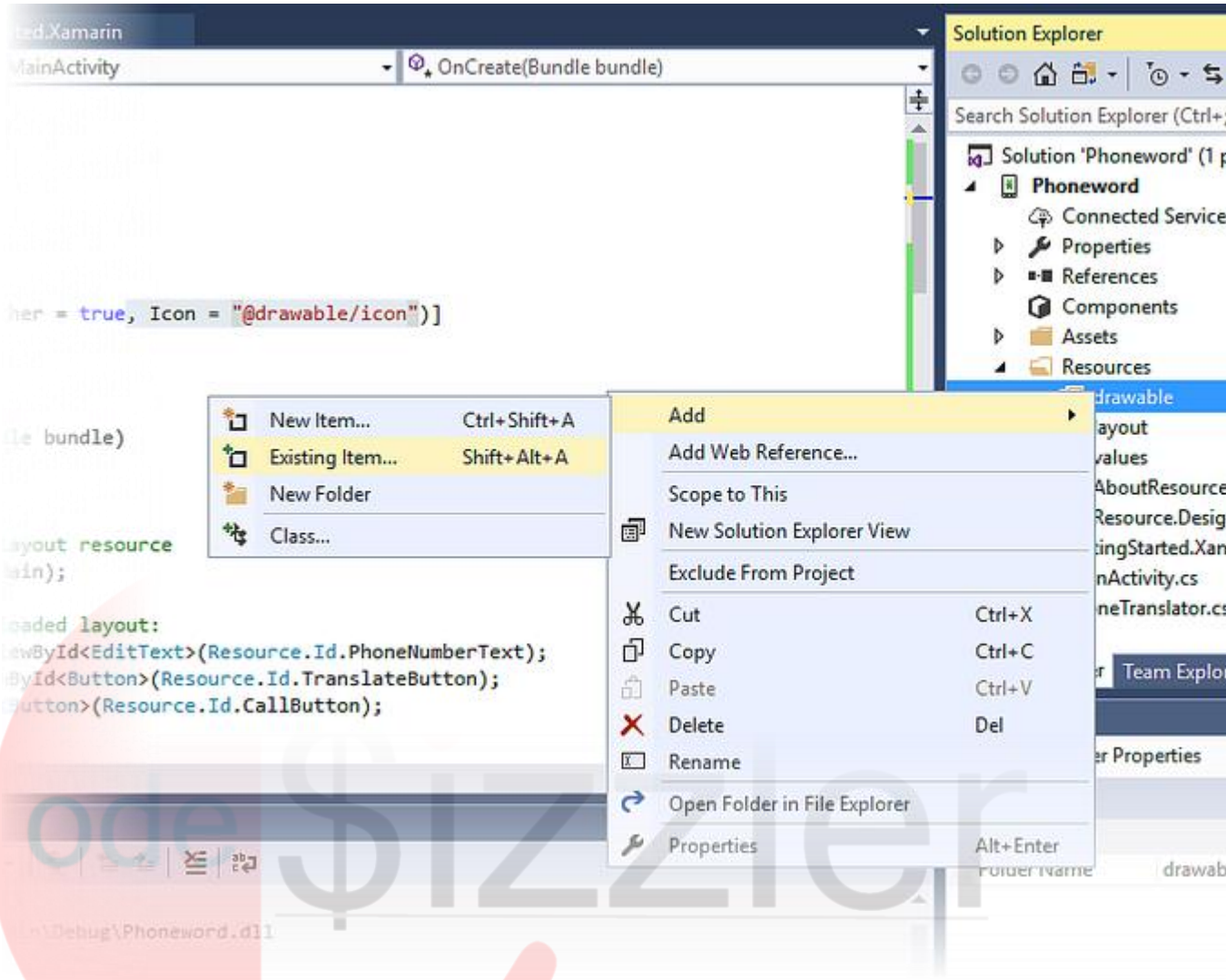
Next, set the application icon. First, download and unzip the Xamarin App Icons set. Next, expand the **drawable** folder under **Resources** and remove the existing **Icon.png** by right-clicking it and selecting **Delete**:



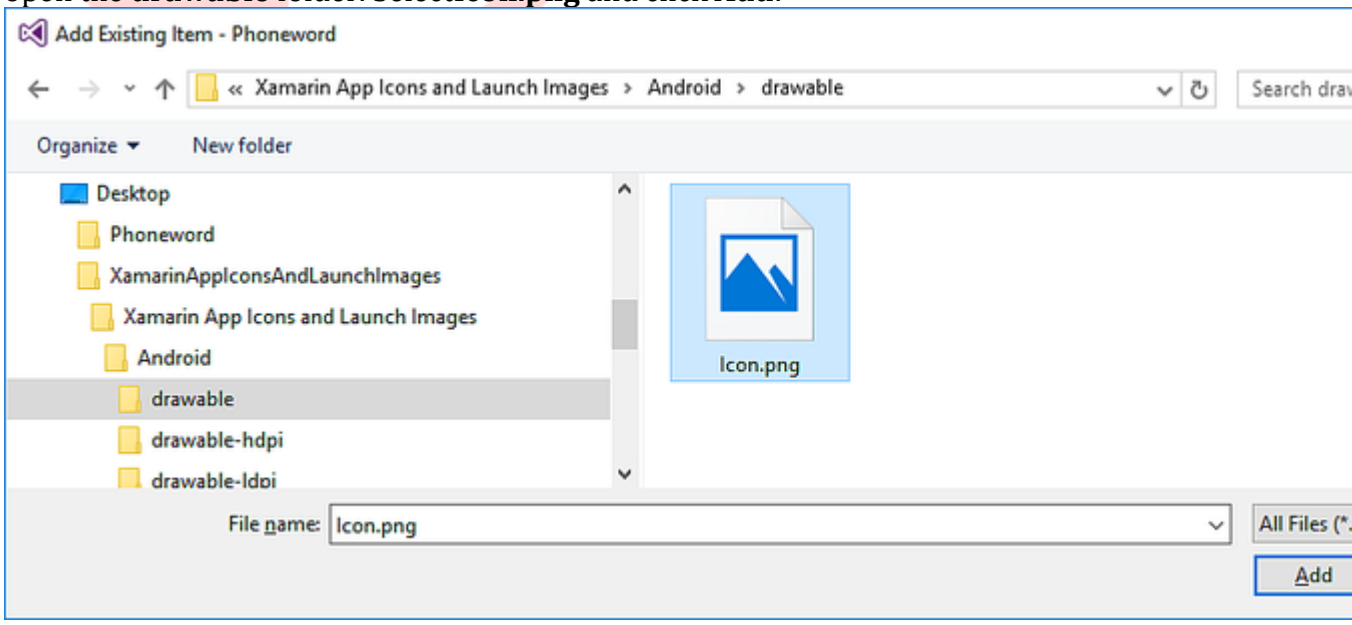
When the following dialog box is displayed, click **OK**:



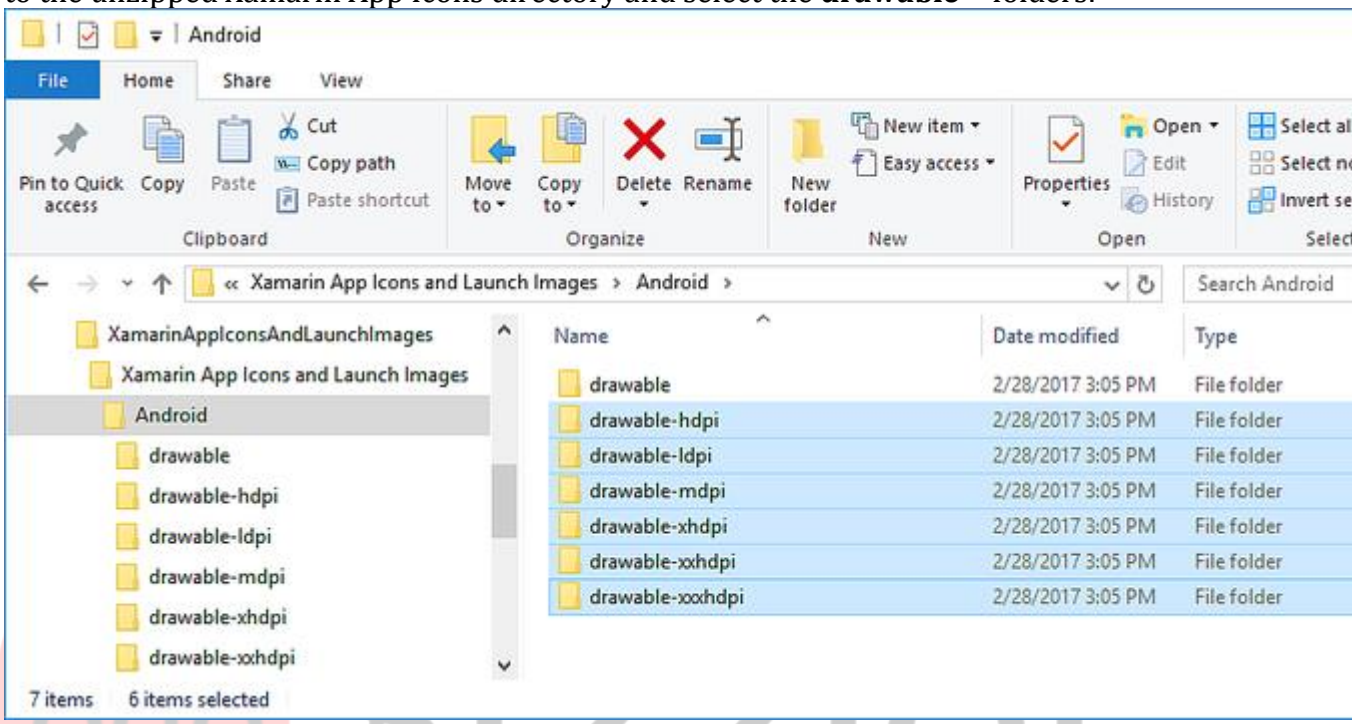
Next, right-click the **drawable** folder and select **Add > Existing Item...**:



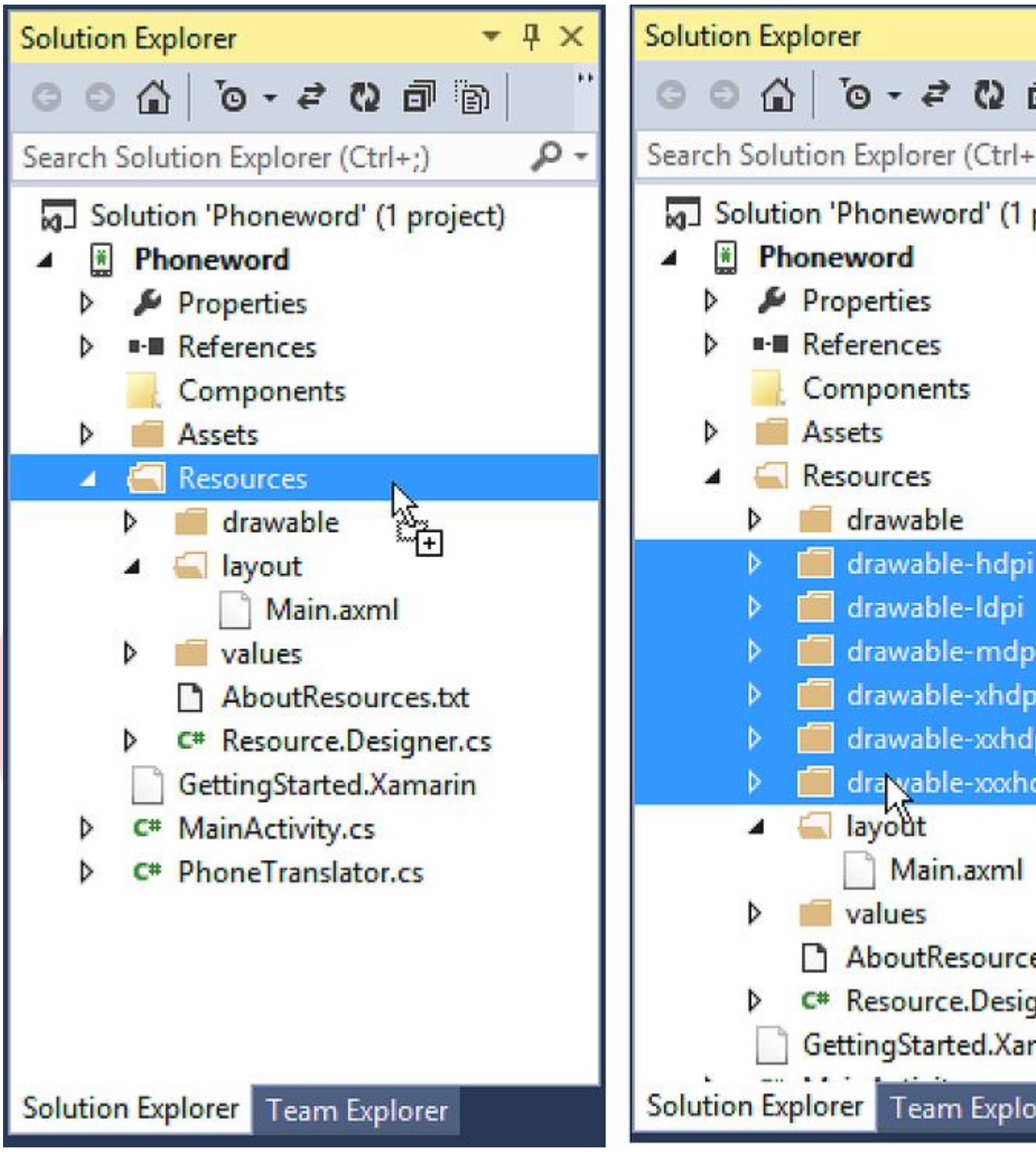
From the selection dialog, navigate to the unzipped Xamarin App Icons directory and open the **drawable** folder. Select**Icon.png** and click **Add**:



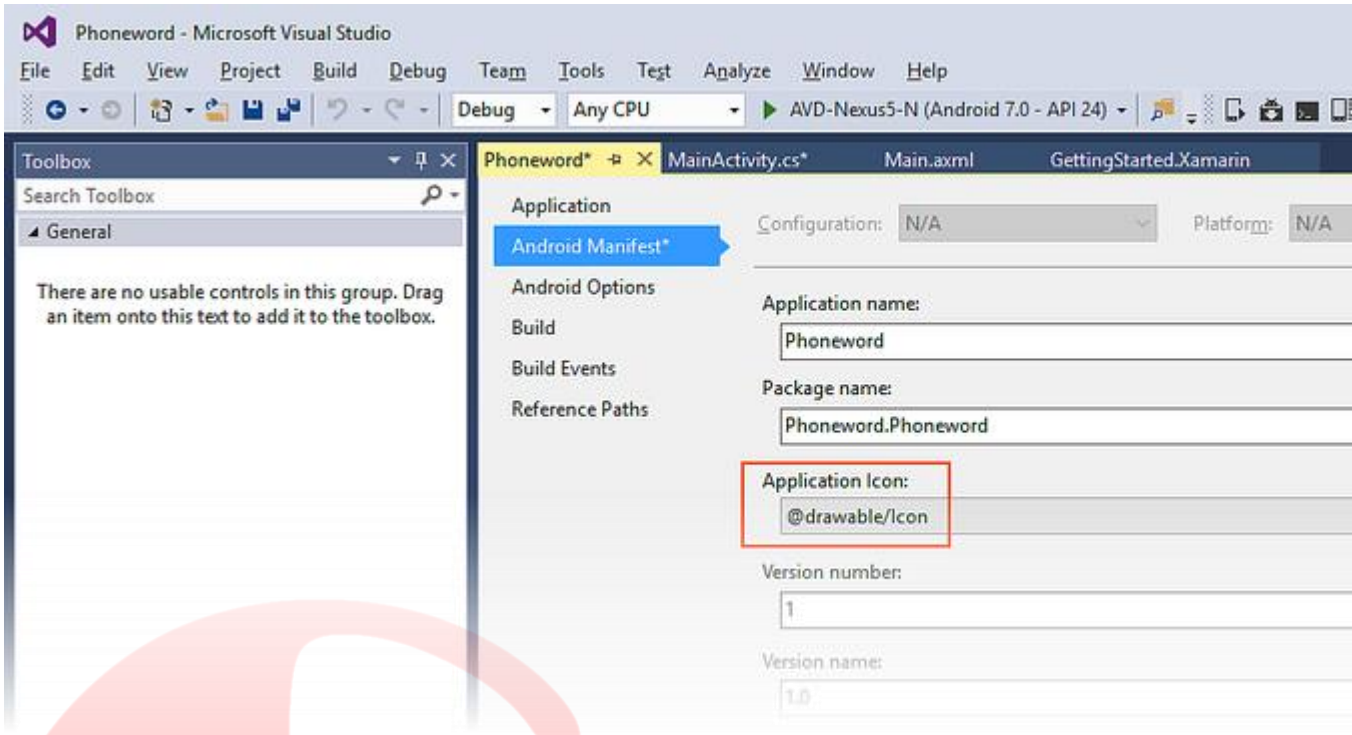
Next, add the rest of the Xamarin App Icons **drawable-** folders to the project. These folders provide different resolutions of the icon so that it renders correctly on different devices with different screen densities. In a File Explorer window, navigate to the unzipped Xamarin App Icons directory and select the **drawable-*** folders:



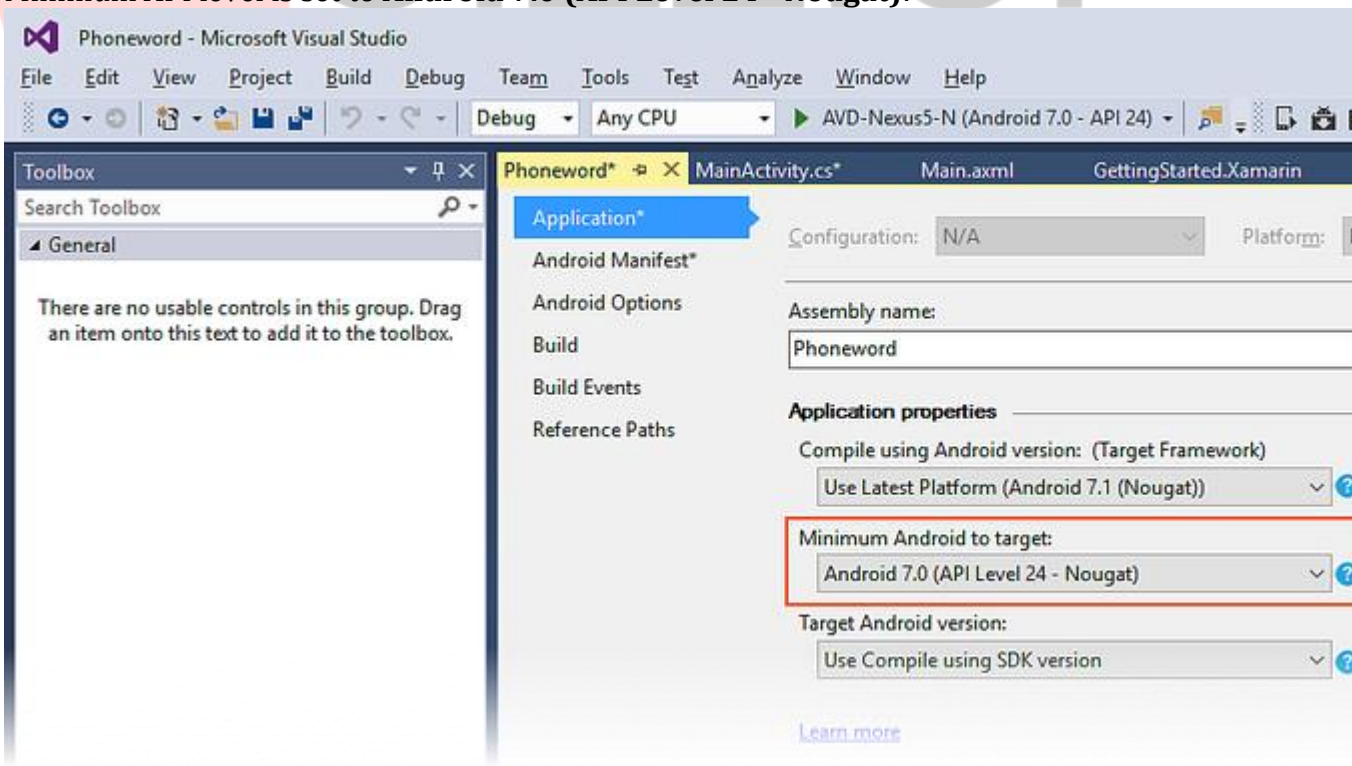
Drag these folders onto the **Resources** folder in the Visual Studio **Solution Explorer** pane. These folders are now part of the project as shown in **Solution Explorer** on the right:



Specify the icon in the Android Manifest by choosing `@drawable/Icon` from the **Application Icon** drop-down menu:

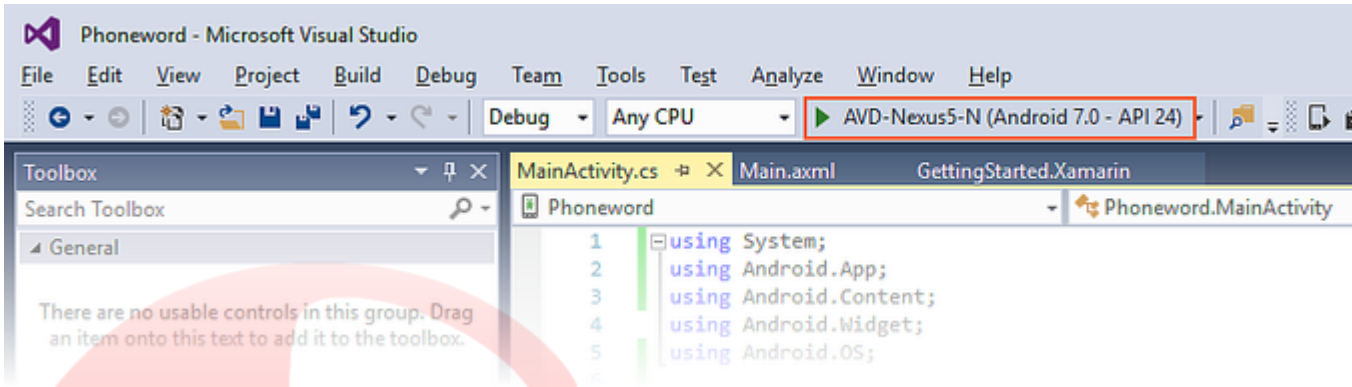


Finally, you can test your application by deploying it to an emulator. Before you send your app to an emulator or device, you must configure the app's minimum Android version so that it matches the Android version of your emulator or test device. In Visual Studio, open the **Application** page of **Properties** and set the API level in the **Minimum Android to target** drop-down menu. In the following, screen shot, the Minimum API level is set to **Android 7.0 (API Level 24 - Nougat)**:

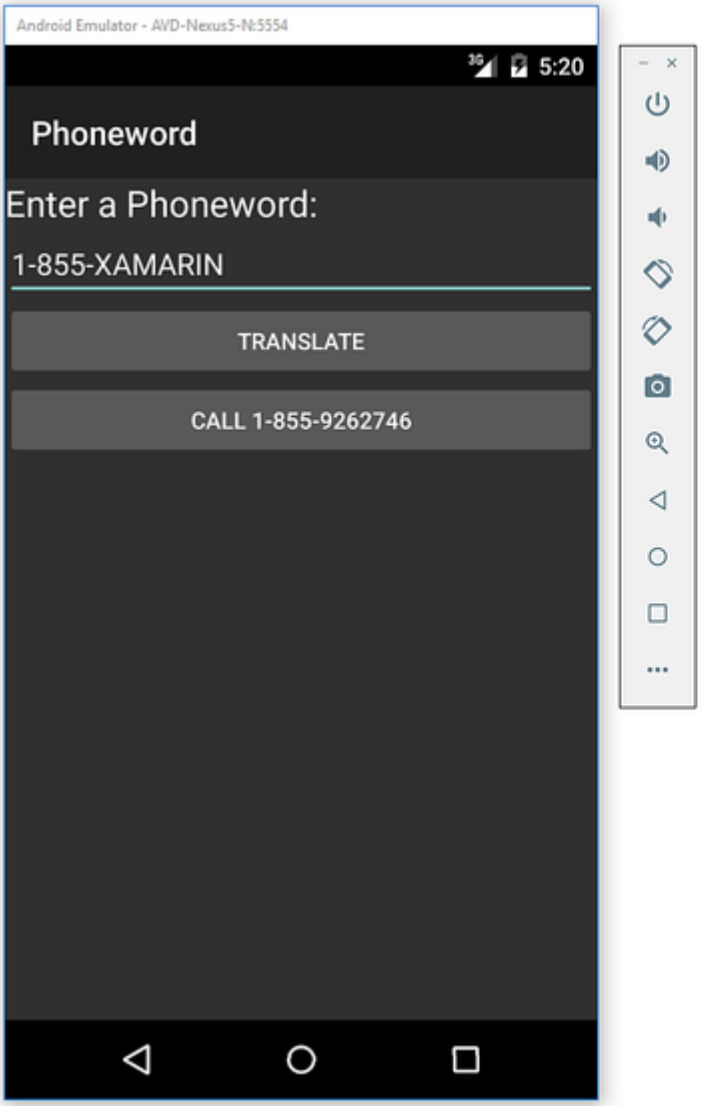


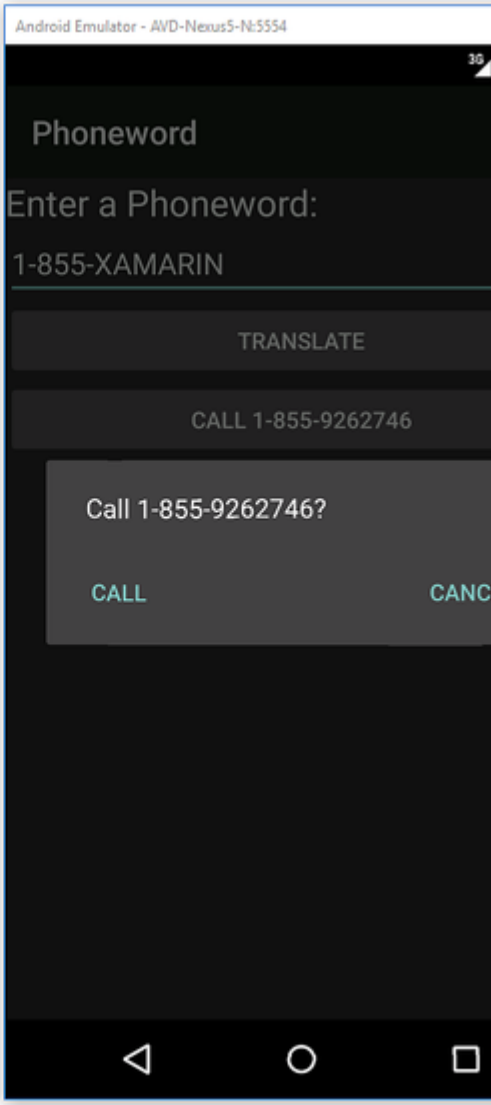
This setting is compatible with the virtual device that will be used in the next step.

In the next screenshot, a custom virtual device called **AVD-Nexus5-N (Android 7.0 - API 24)** is chosen from the device drop-down menu. If no virtual devices are available in the drop-down menu, you must create and configure a virtual device via the Android AVD Manager as explained in Android SDK Emulator. Deploy the app by choosing the device from the drop-down menu and then clicking the green "play" arrow as shown here:



Visual Studio will copy files to this emulator before installing and launching the app.
The screenshots below illustrate the **Phoneword** application running in the Android SDK Emulator. Clicking the **Translate** button updates the text of the **Call** button, and clicking the **Call** button causes the call dialog to appear as shown on the right:


Android Emulator - AVD-Nexus5-N:5554
5:20
Phoneword
Enter a Phoneword:
1-855-XAMARIN
TRANSLATE
CALL 1-855-9262746


Android Emulator - AVD-Nexus5-N:5554
Phoneword
Enter a Phoneword:
1-855-XAMARIN
TRANSLATE
CALL 1-855-9262746
Call 1-855-9262746?
CALL
CANC