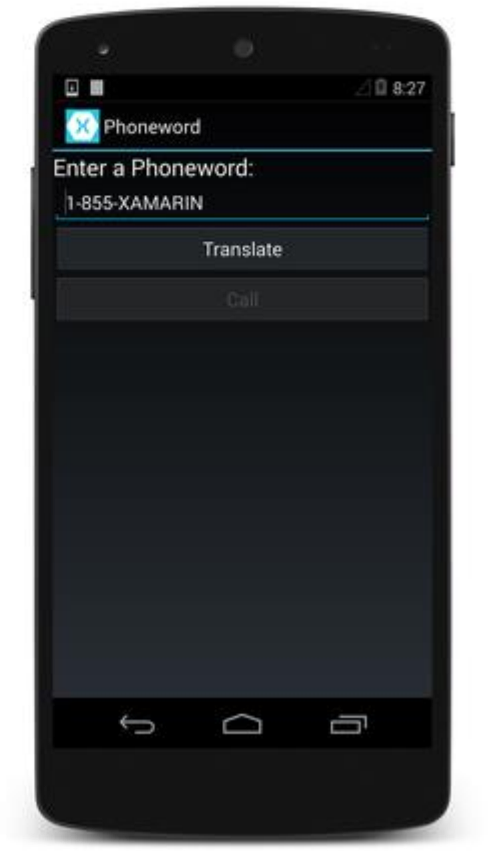
Hello, Android

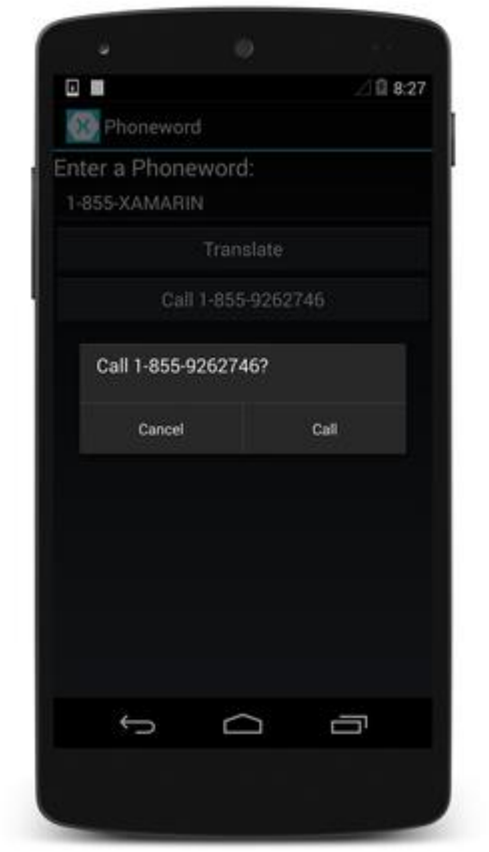
Introduction to Android Development with Xamarin

In this two-part guide, we build our 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 we'll introduce the tools, concepts, and steps required to build and deploy a Xamarin.Android application.

Hello, Android Quickstart

In this walkthrough, we 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:





Let's get started!

Requirements

Xamarin.Android works with any of the following setups:

- Latest version of Xamarin Studio on OS X Yosemite and above.
- Latest version of Xamarin Studio on Windows 7 and above.
- Windows 7 and above with Visual Studio 2012 Professional or higher.

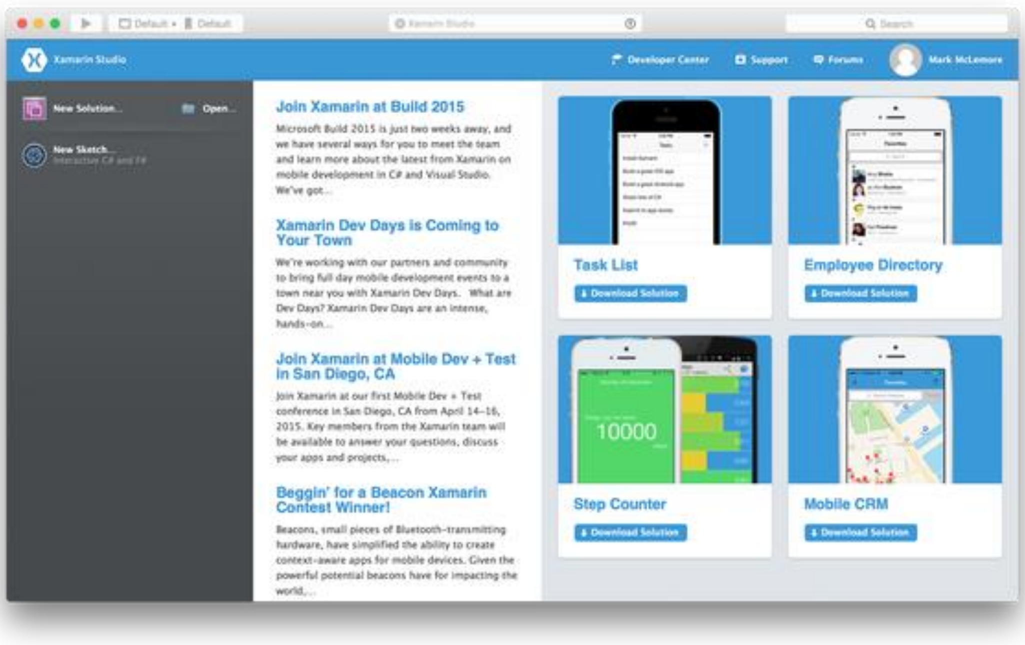
This walkthrough assumes that the latest version of Xamarin.Android is installed and running on your platform of choice. For a guide to installing Xamarin.Android, refer to the Xamarin.Android Installation guides. Before we get started, please download and unzip the Xamarin App Icons & Launch Screens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

Configuring Emulators

Android has several options for emulators. The standard Android emulator is the simplest to set up but runs slowly. Xamarin recommends that you use the high-performance Xamarin Android Player. If you are not using the Xamarin Android Player, you should configure your emulator to use hardware acceleration. Instructions for configuring hardware acceleration are available in the Accelerating Android Emulators guide.

Walkthrough

Let's launch Xamarin Studio from the *Applications* folder or from *Spotlight*. This opens the start page:



Join Xamarin at Build 2015

Microsoft Build 2015 is just two weeks away, and we have several ways for you to meet the team and learn more about the latest from Xamarin on mobile development in C# and Visual Studio. We've got...

Xamarin Dev Days is Coming to Your Town

We're working with our partners and community to bring full day mobile development events to a town near you with Xamarin Dev Days. What are Dev Days? Xamarin Dev Days are an intense, hands-on...

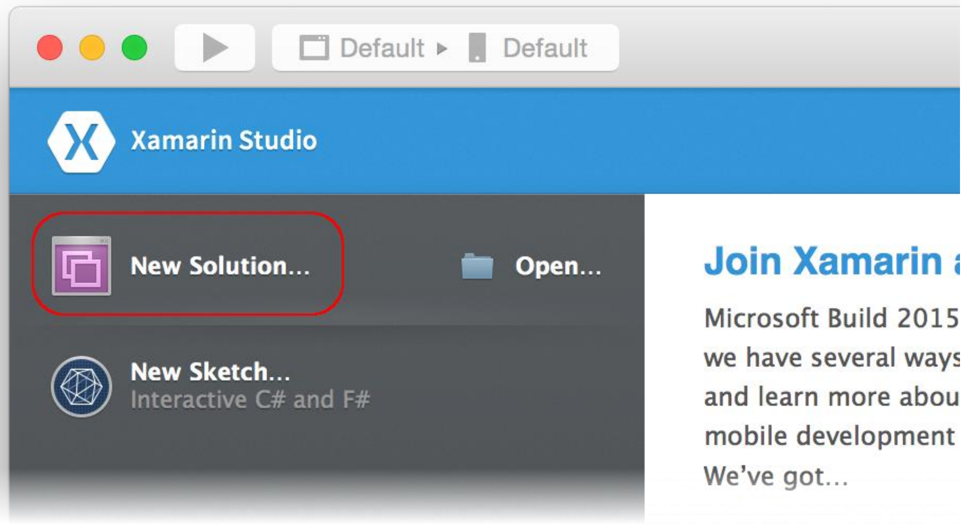
Join Xamarin at Mobile Dev + Test in San Diego, CA

Join Xamarin at our first Mobile Dev + Test conference in San Diego, CA from April 14–16, 2015. Key members from the Xamarin team will be available to answer your questions, discuss your apps and projects,...

Beggin' for a Beacon Xamarin Contest Winner!

Beacons, small pieces of Bluetooth-transmitting hardware, have simplified the ability to create context-aware apps for mobile devices. Given the powerful potential beacons have for impacting the world,...

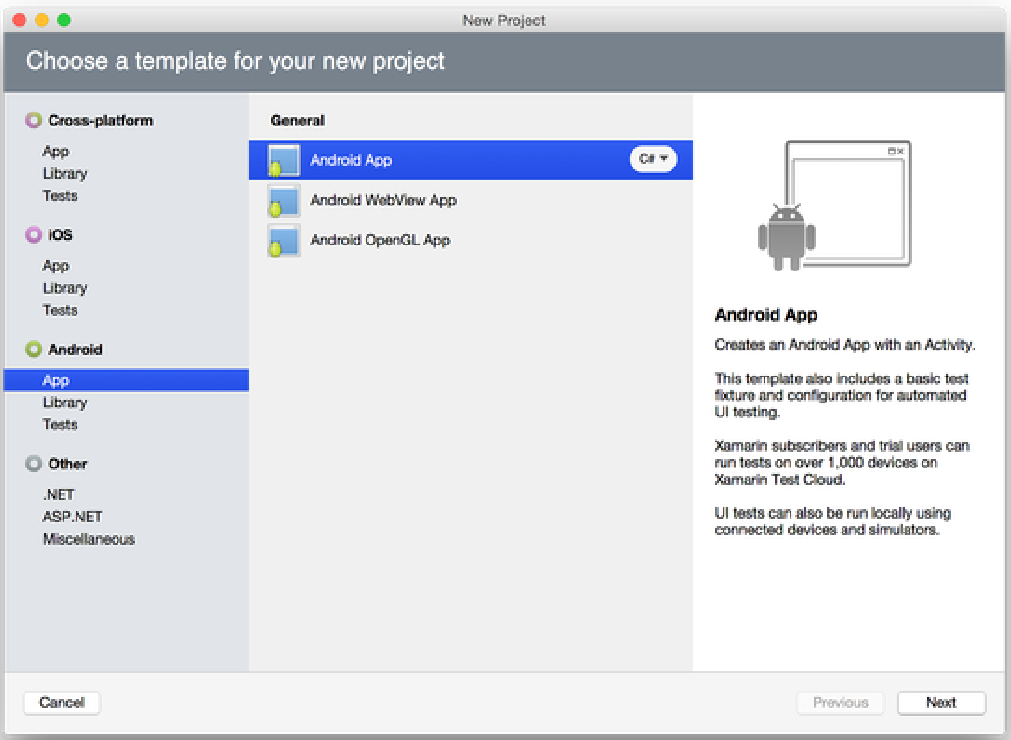
2. Click *New Solution...* to create a new project:



Join Xamarin a

Microsoft Build 2015
we have several ways
and learn more abou
mobile development
We've got...

3. In the *Choose a template for your new project* dialog, let's click *Android* > *App* and select the *Android App* template. Click *Next*.



Android App

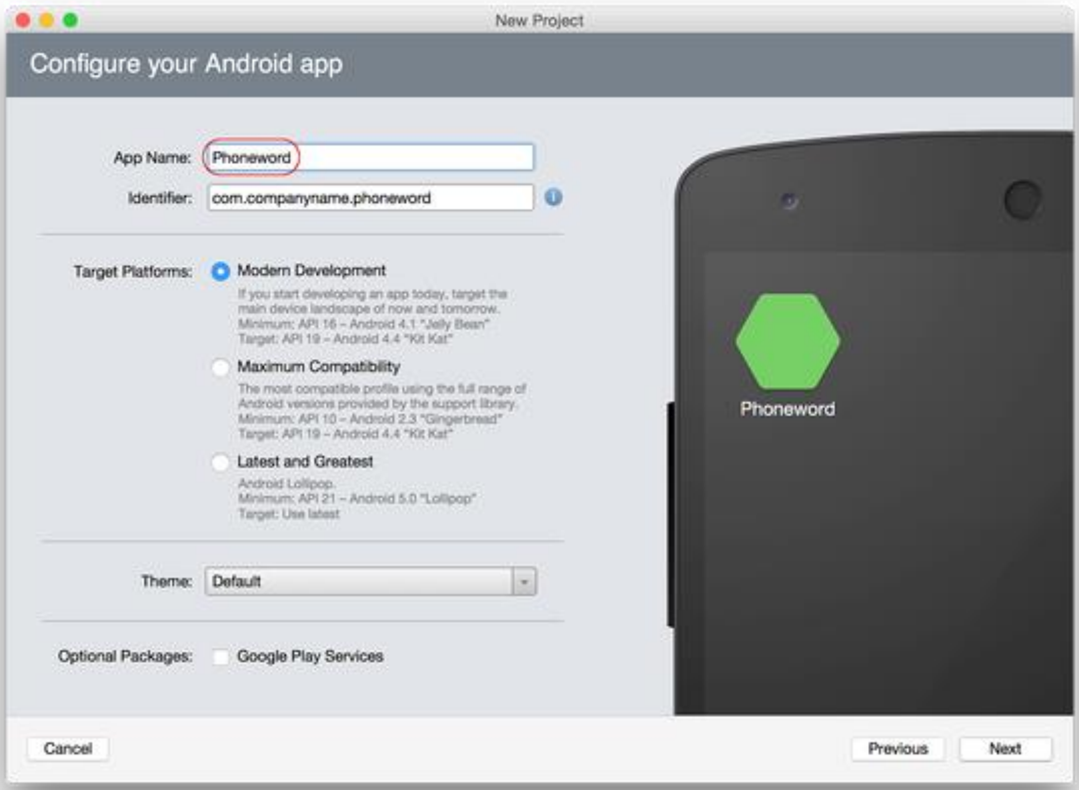
Creates an Android App with an Activity.

This template also includes a basic test fixture and configuration for automated UI testing.

Xamarin subscribers and trial users can run tests on over 1,000 devices on Xamarin Test Cloud.

UI tests can also be run locally using connected devices and simulators.

4. In the *Configure your Android app* dialog, we'll name the new app `Phoneword` and click *Next*:



App Name: Phoneword

Identifier: com.companyname.phoneword

Target Platforms: [✓ Modern Development]

If you start developing an app today, target the main device landscape of now and tomorrow.
Minimum: API 16 – Android 4.1 "Jelly Bean"
Target: API 19 – Android 4.4 "Kit Kat"

Maximum Compatibility

The most compatible profile using the full range of Android versions provided by the support library.
Minimum: API 10 – Android 2.3 "Gingerbread"
Target: API 19 – Android 4.4 "Kit Kat"

Latest and Greatest

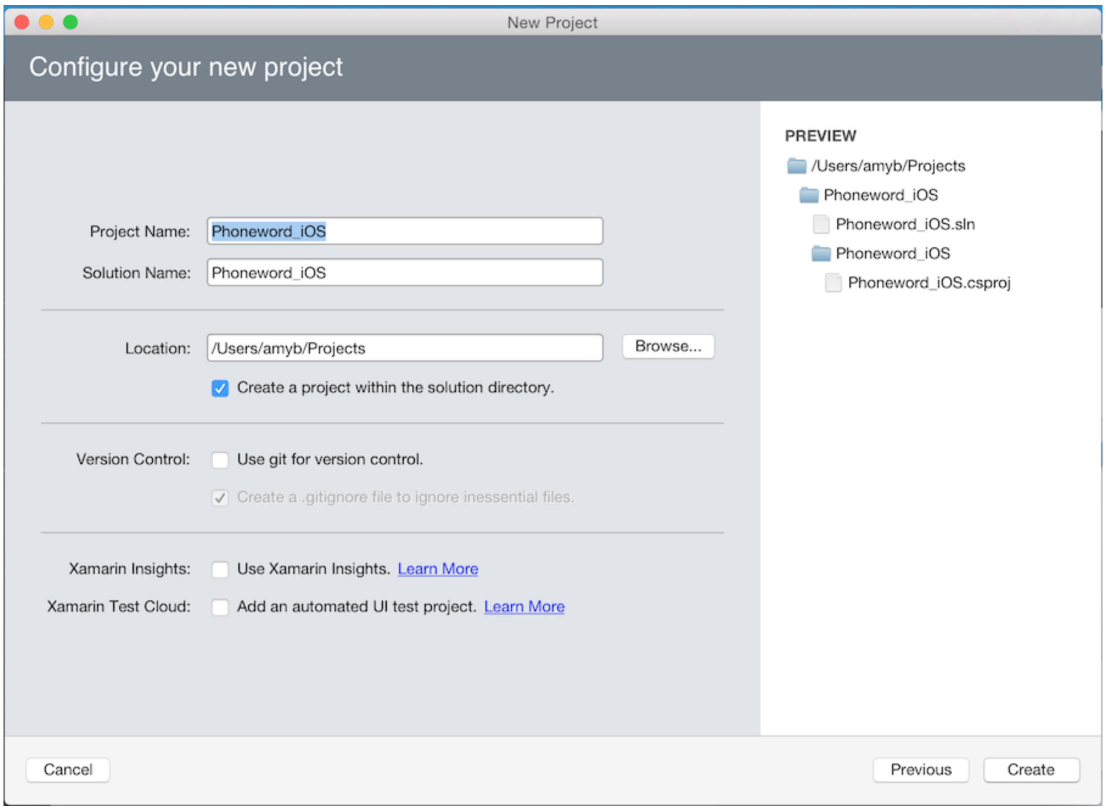
Android Lollipop.
Minimum: API 21 – Android 5.0 "Lollipop"
Target: Use latest

Theme: Default

Optional Packages: ☐ Google Play Services

5. In the *Configure your new project* dialog, we'll leave the Solution and Project names set to `Phoneword` and click *Create* to create the project.

   Make sure to uncheck the checkboxes for Xamarin Insights and Xamarin Test Cloud:


New Project

Configure your new project

Project Name: Phoneword_iOS

Solution Name: Phoneword_iOS

Location: /Users/amyb/Projects

☑ Create a project within the solution directory.

Version Control: ☐ Use git for version control.

☑ Create a .gitignore file to ignore inessential files.

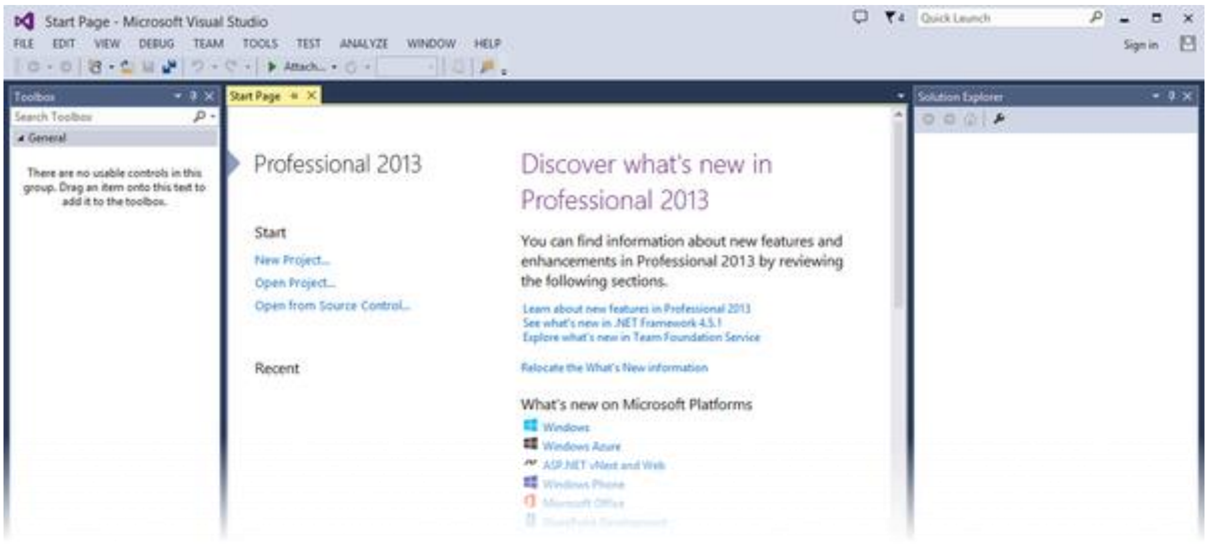
Xamarin Insights: ☐ Use Xamarin Insights. Learn More

Xamarin Test Cloud: ☐ Add an automated UI test project. Learn More

PREVIEW

- /Users/amyb/Projects
  - Phoneword_iOS
    - Phoneword_iOS.sln
    - Phoneword_iOS
      - Phoneword_iOS.csproj

6. Let's start Visual Studio:


Start Page - Microsoft Visual Studio
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Quick Launch
Sign in
Attach...
Toolbox
Search Toolbox
General
Start Page
Solution Explorer
Professional 2013
Start
New Project...
Open Project...
Open from Source Control...
Recent
Learn about new features in Professional 2013
See what's new in .NET Framework 4.5.1
Explore what's new in Team Foundation Service
Relocate the What's New information
Windows
Windows Azure
ASP.NET vNext and Web
Windows Phone
Microsoft Office

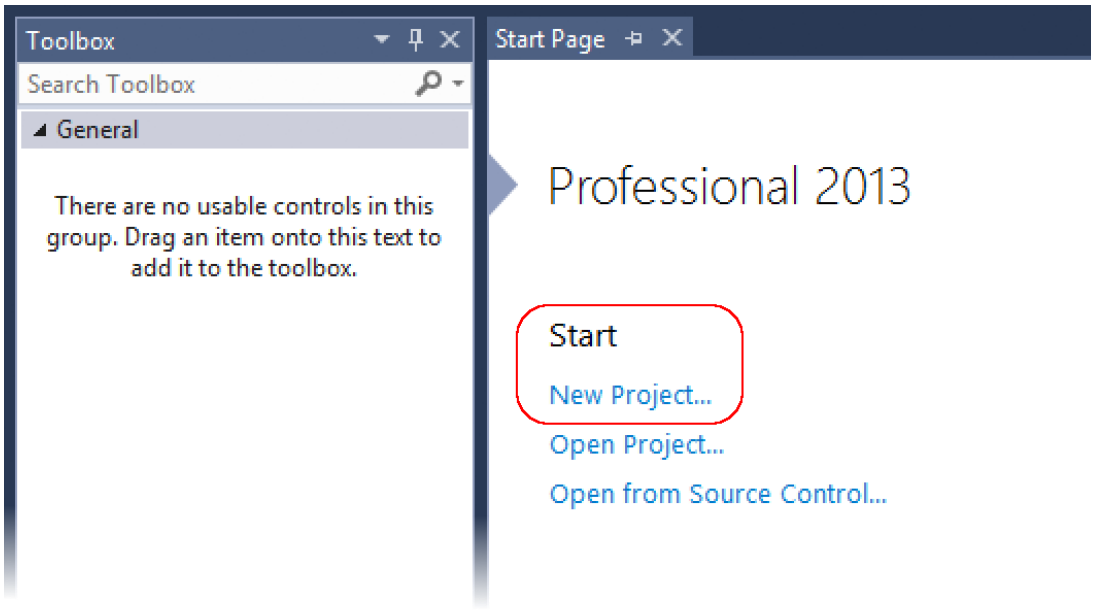
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Discover what's new in Professional 2013

You can find information about new features and enhancements in Professional 2013 by reviewing the following sections.

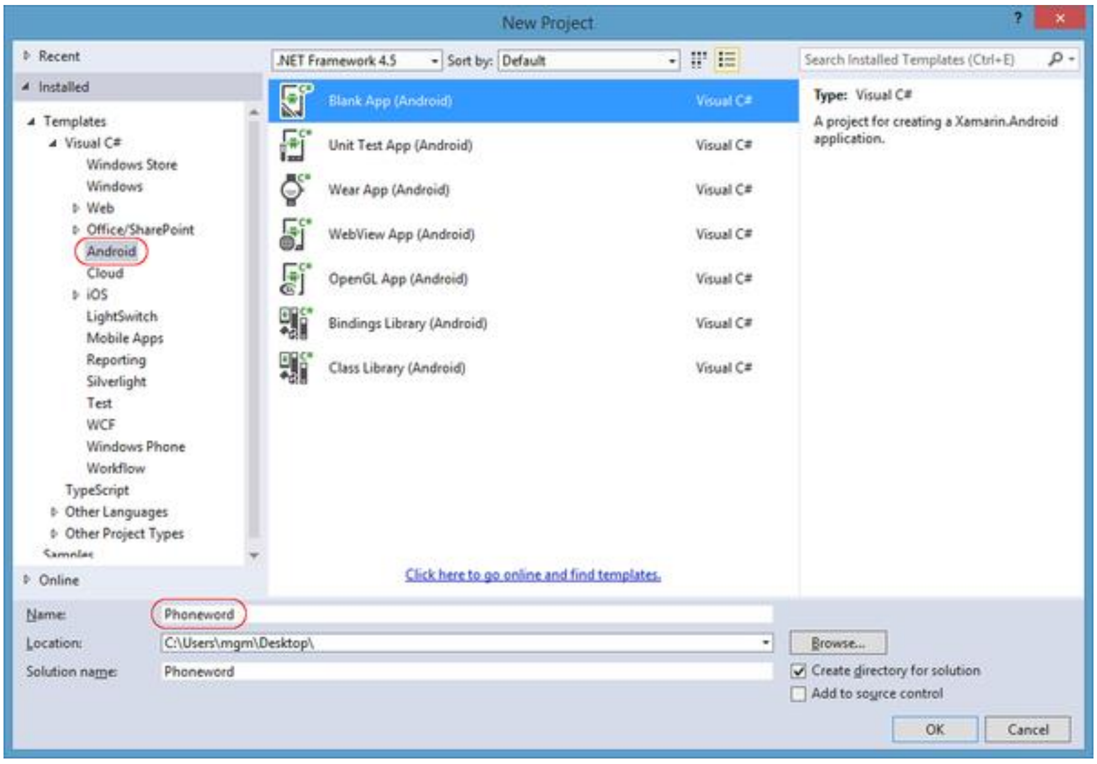
What's new on Microsoft Platforms

7. Click *Start* > *New Project* to create a new project:



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

8. In the *New Project* dialog, click *Visual C# > Android* and select the *Blank App (Android)* template. We'll name the new project `Phoneword`. Click *OK* to create the new project:


New Project
Recent
Installed
Templates
Visual C#
Windows Store
Windows
Web
Office/SharePoint
Android
Cloud
iOS
LightSwitch
Mobile Apps
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
TypeScript
Other Languages
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Android) Visual C#
Unit Test App (Android) Visual C#
Wear App (Android) Visual C#
WebView App (Android) Visual C#
OpenGL App (Android) Visual C#
Bindings Library (Android) Visual C#
Class Library (Android) Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#

A project for creating a Xamarin.Android application.

Name: Phoneword

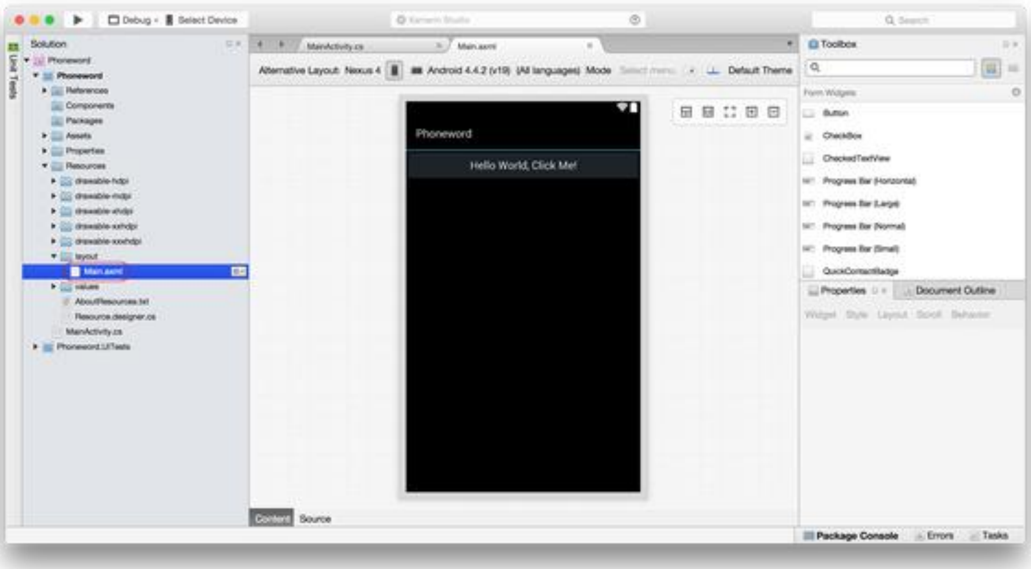
Location: C:\Users\mgm\Desktop\

Solution name: Phoneword

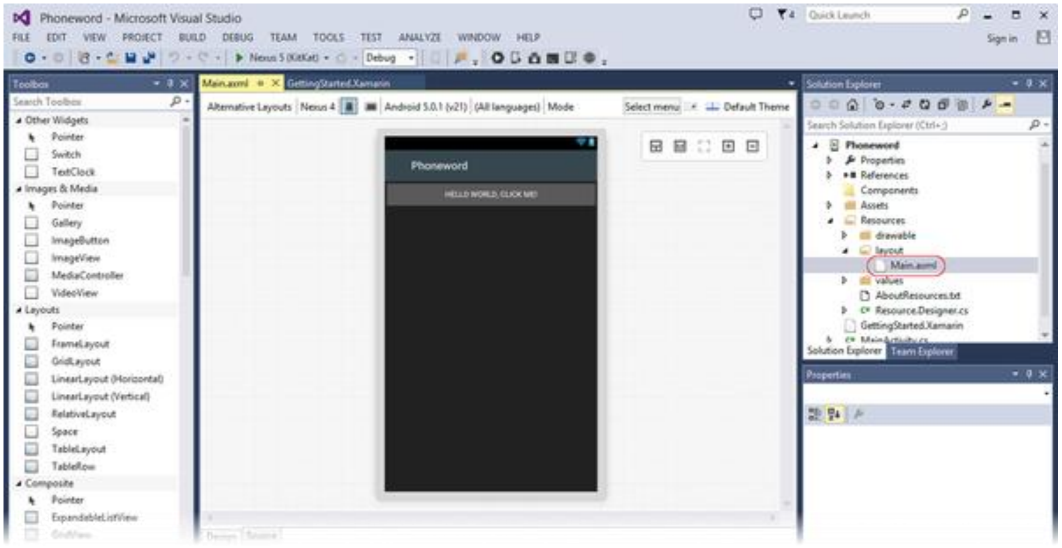
☑ Create directory for solution

☐ Add to source control

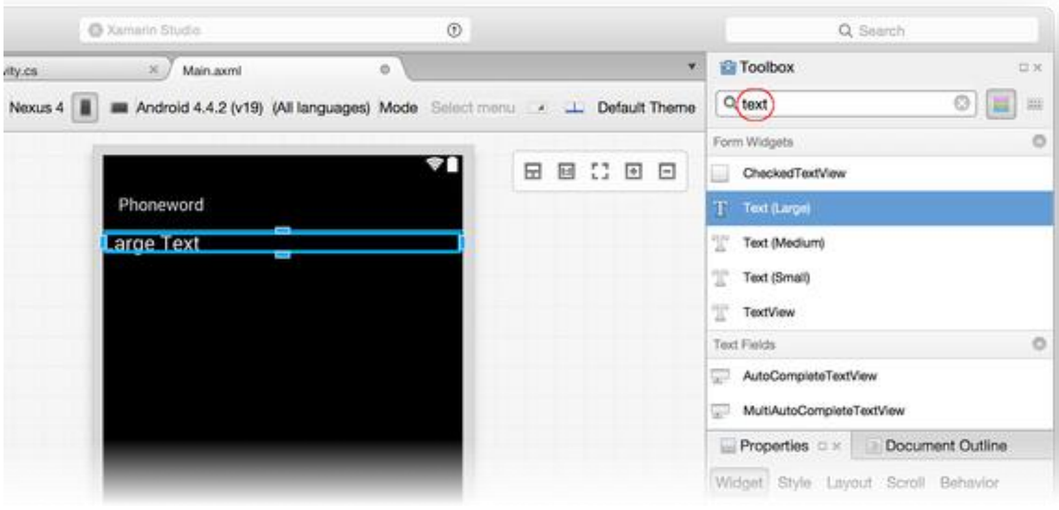
9. After the new project is created, let's expand the *Resources* folder and then the *layout* folder in the *Solution* pad. Double-click **Main.axml** to open it in the Android Designer. This is the layout file for our screen:



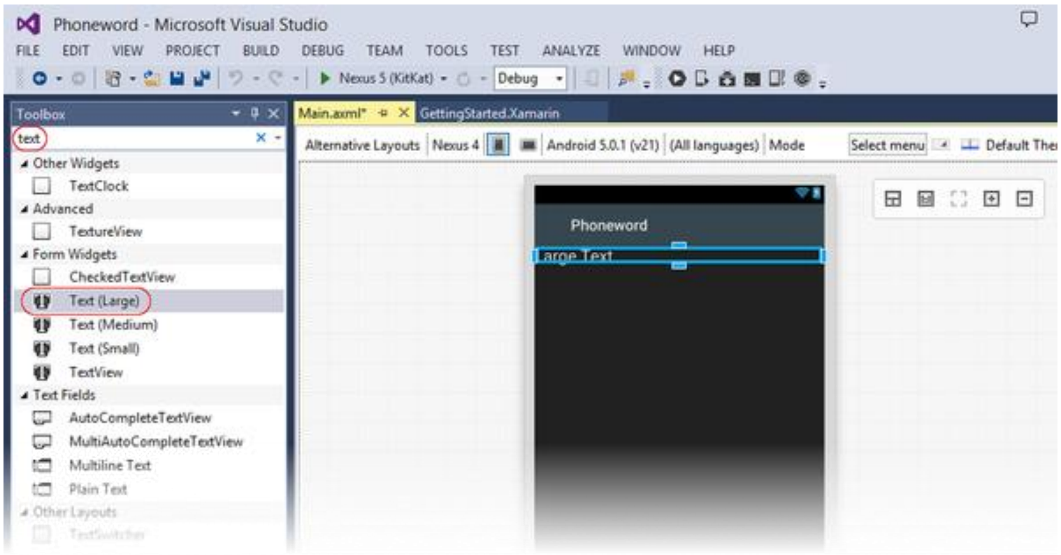
10. After the new project is created, let‘s expand the *Resources* folder and then the *layout* folder in the *Solution Explorer*. Double-click **Main.axml** to open it in the Android Designer. This is the layout file for our screen:



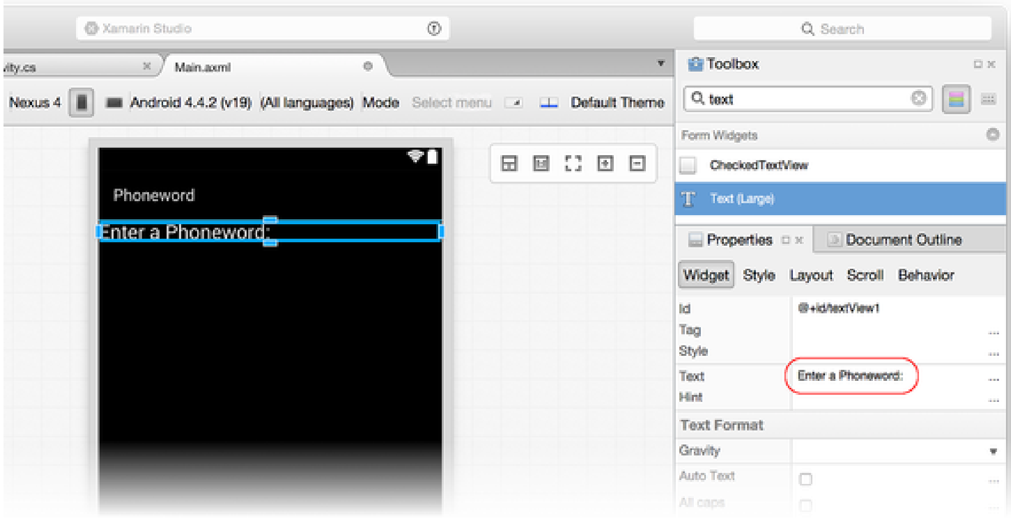
11. Let‘s select the **Hello World, Click Me!** *Button* on the design surface and press the `Delete` key to remove it. From the *Toolbox* (the area on the right), enter `text` into the search field and drag a *Text (Large)* widget onto the design surface (the area in the center):



12. Let's select the **Hello World, Click Me!** *Button* on the design surface and press the `Delete` key to remove it. From the *Toolbox* (the area on the left), enter `text` into the search field and drag a *Text (Large)* control onto the design surface (the area in the center):

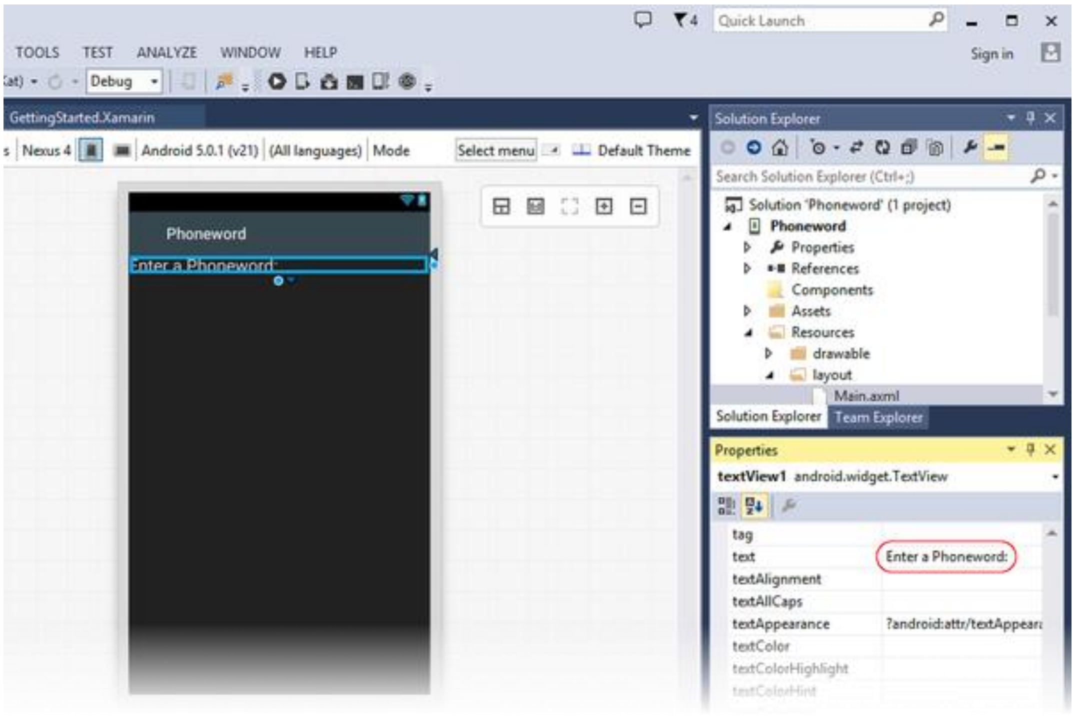


13. With the *Text (Large)* widget selected on the design surface, we can use the *Properties* pad to change the `Text` property of the *Text (Large)* widget to `Enter a Phoneword:` as seen below:

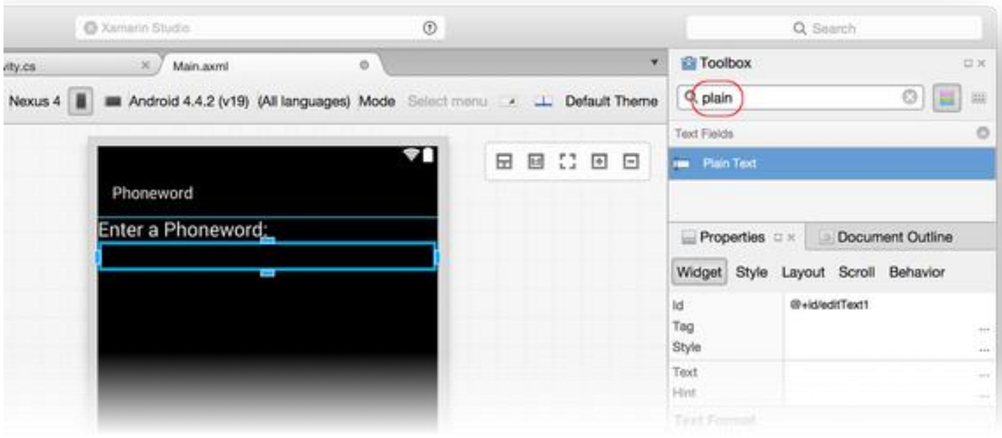


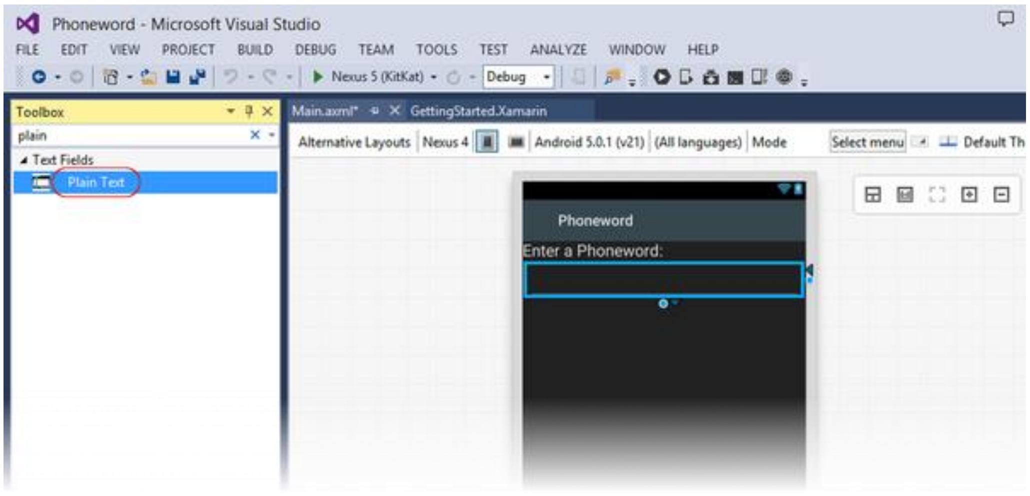
Note: You can bring up the *Properties* pad or *Toolbox* at any time by navigating to *View* > *Pads*.

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

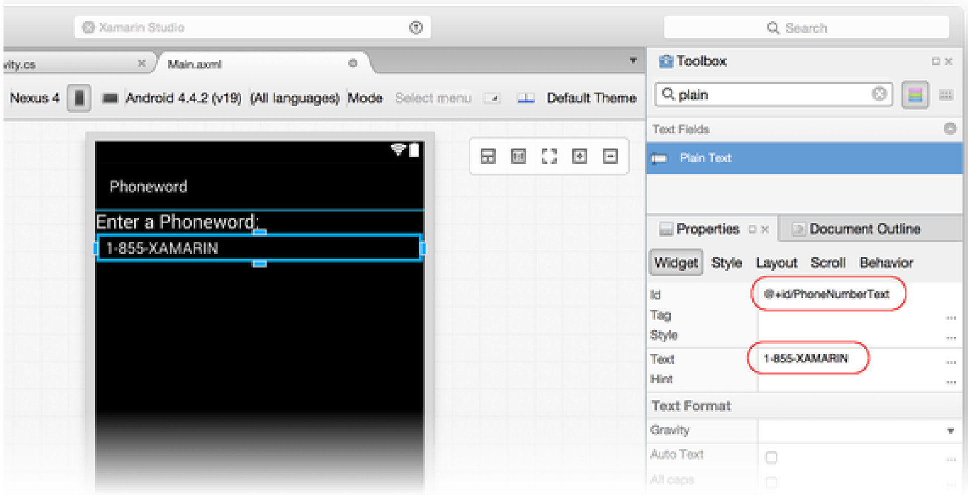


15. Next, let‘s drag a *Plain Text* widget from the *Toolbox* to the design surface and place it underneath the *Text (Large)* widget. Notice that we can use the search field to help locate widgets by name:

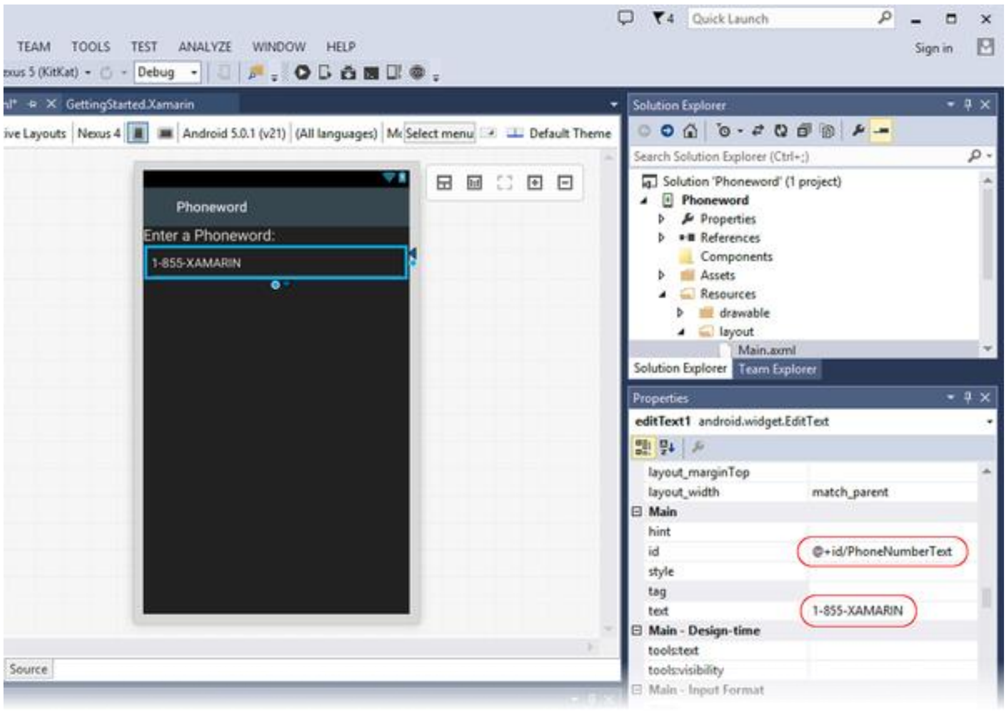




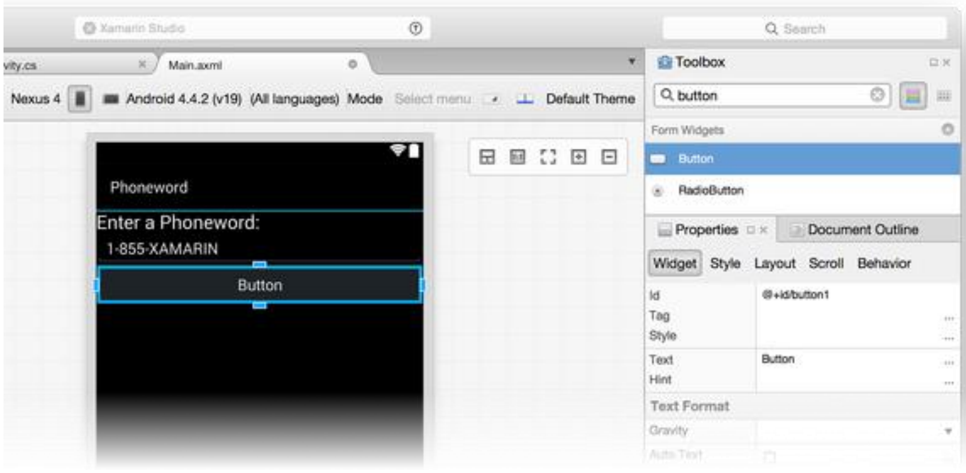
16. With the *Plain Text* widget selected on the design surface, we can use the *Properties* pad to change the `Id` property of the *Plain Text* widget to `@+id/PhoneNumberText` and change the `Text` property to `1-855-XAMARIN`:

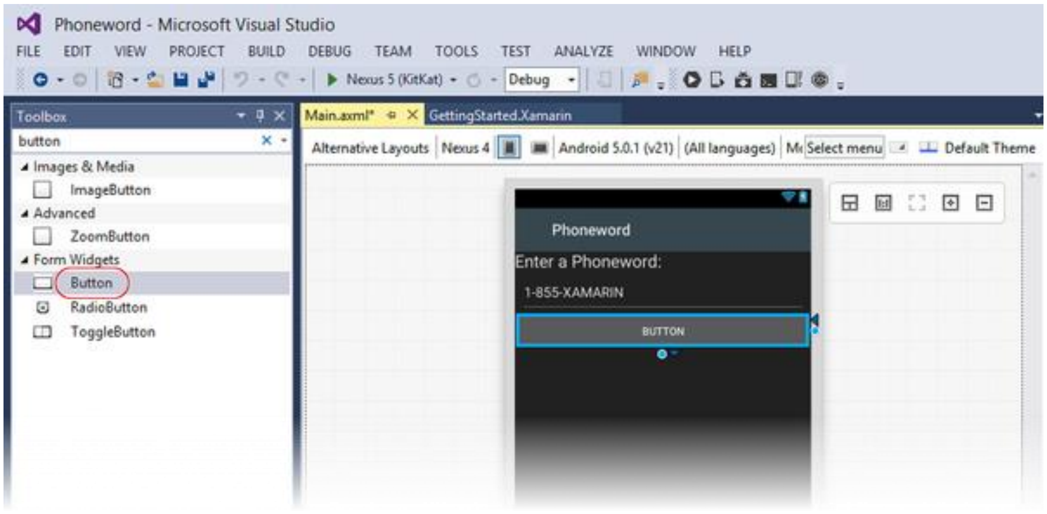


17. With the *Plain Text* control selected on the design surface, we can use the *Properties* pane to change the `id` property of the *Plain Text* control to `@+id/PhoneNumberText` and change the `text` property to `1-855-XAMARIN`:

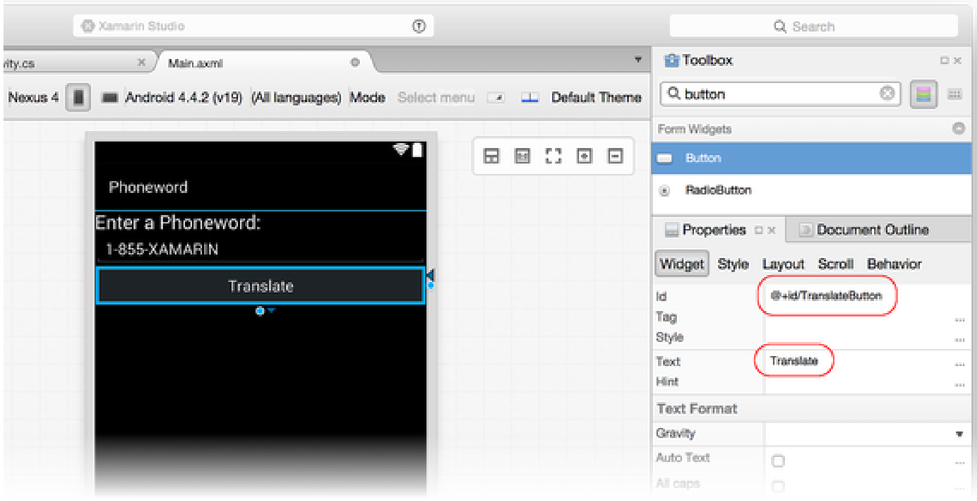


18. Let‘s drag a *Button* from the *Toolbox* to the design surface and place it underneath the *Plain Text* widget:



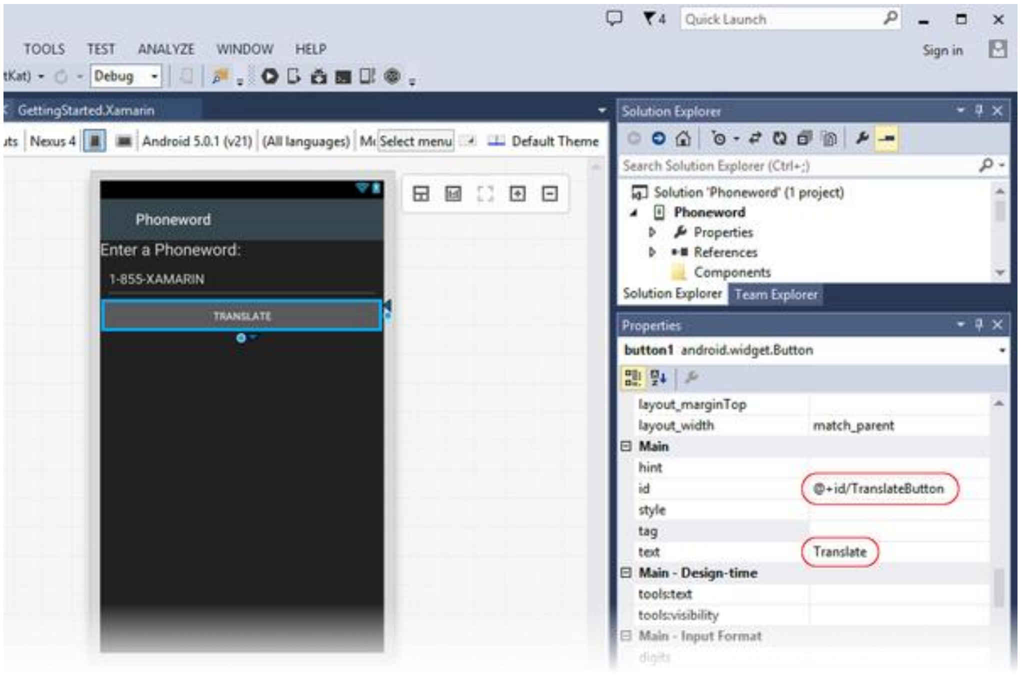


19. With the *Button* selected on the design surface, we can use the *Properties* pad to change the `Id` property of the *Button* to `@+id/TranslateButton` and change the `Text` property to `Translate`:



Id	@+id/TranslateButton
Tag	
Style	
Text	Translate
Hint	

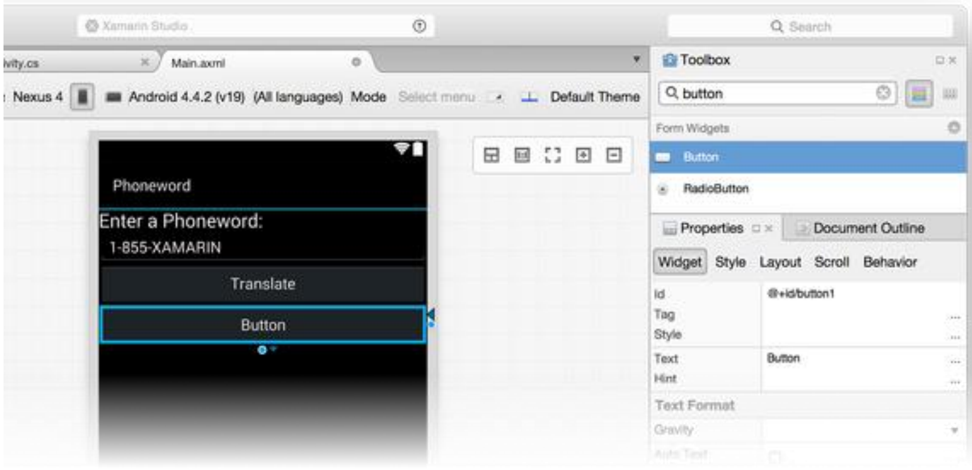
20. With the *Button* selected on the design surface, we can use the *Properties* pane to change the `id` property of the *Button* to `@+id/TranslateButton` and change the `text` property to `Translate`:



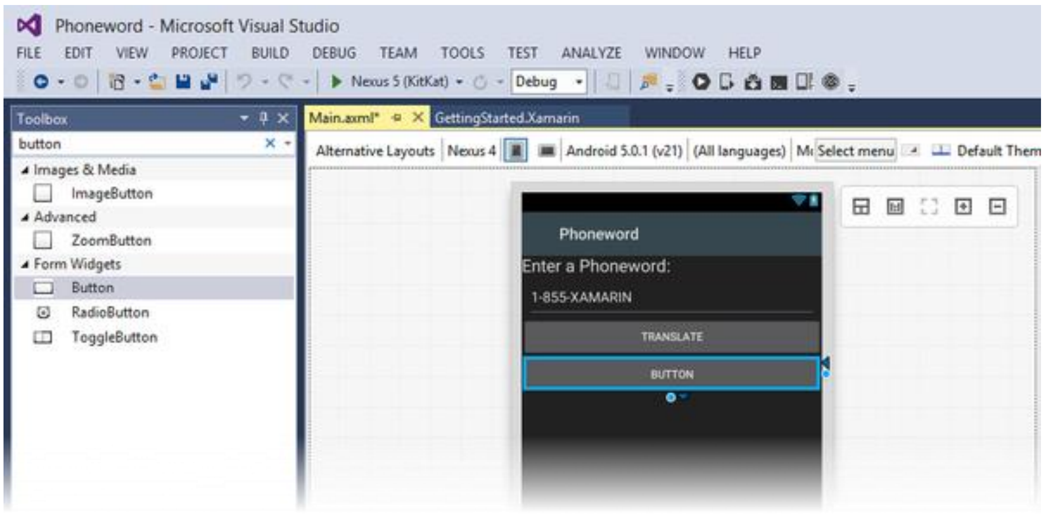
button1 android.widget.Button

layout_marginTop	
layout_width	match_parent
Main	
hint	
id	@+id/TranslateButton
style	
tag	
text	Translate
Main - Design-time	
tools:text	
tools:visibility	
Main - Input Format	
digits	

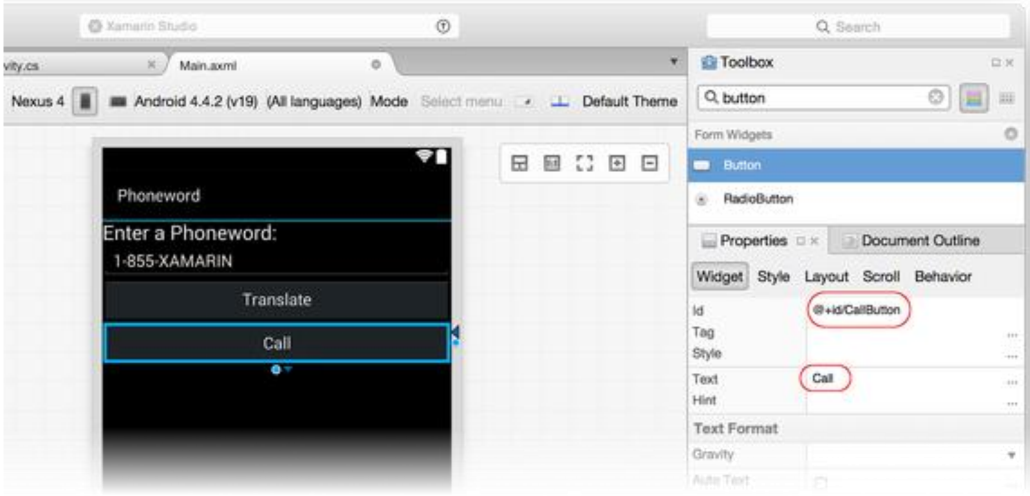
21. Next, let's drag a second *Button* from the *Toolbox* to the design surface and place it underneath the *Translate* button:



Id	@+id/button1
Tag	
Style	
Text	Button
Hint	
Text Format	
Gravity	
Auto Text	

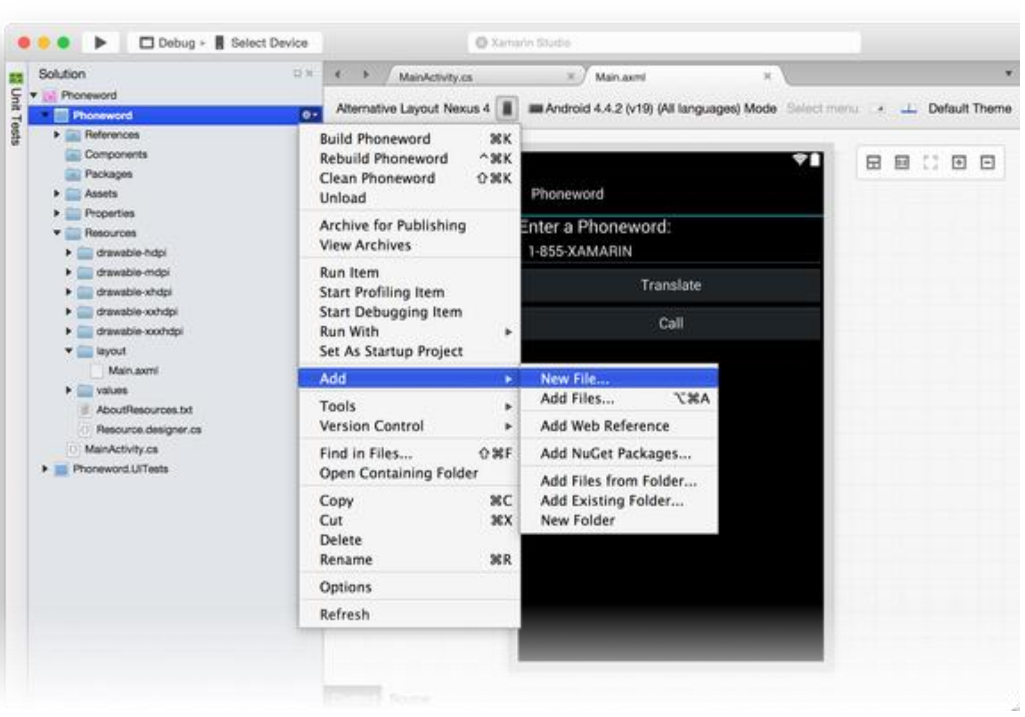


22. With the *Button* selected on the design surface, we can use the *Properties* pad to change the `Id` property of the *Button* to `@+id/CallButton` and change the `Text` property to `Call`:

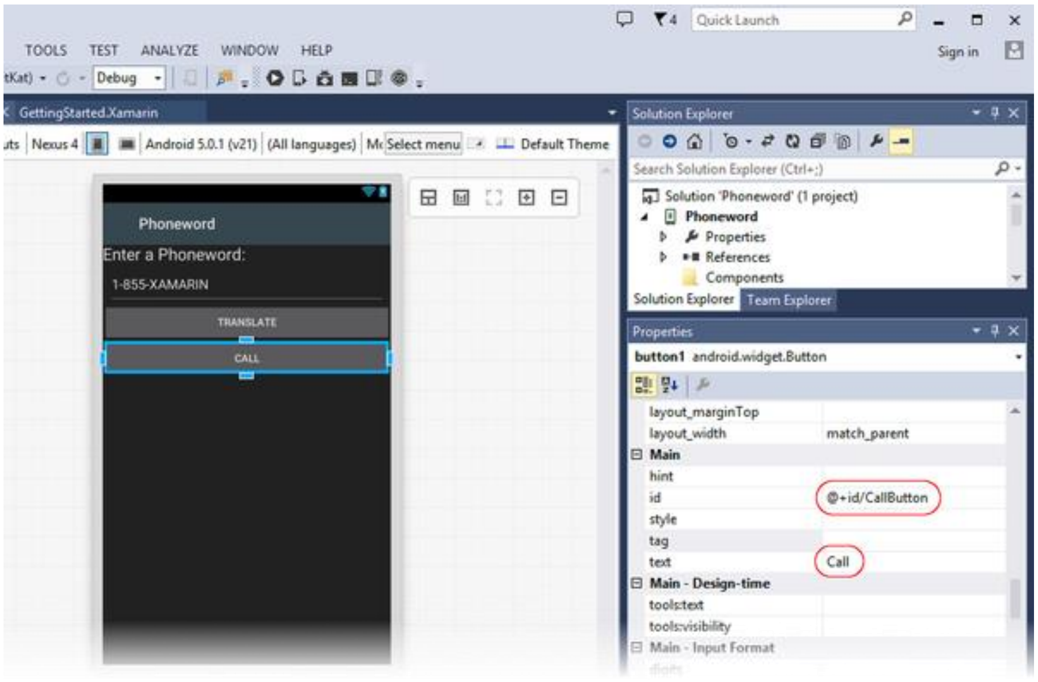


Let's save our work by pressing `⌘ + S`.

23. Now, let‘s add some code to translate phone numbers from alphanumeric to numeric. We‘ll add a new file to the project by clicking on the gear icon next to the *Phoneword* project in the *Solution* pad and choosing *Add > New File...*:

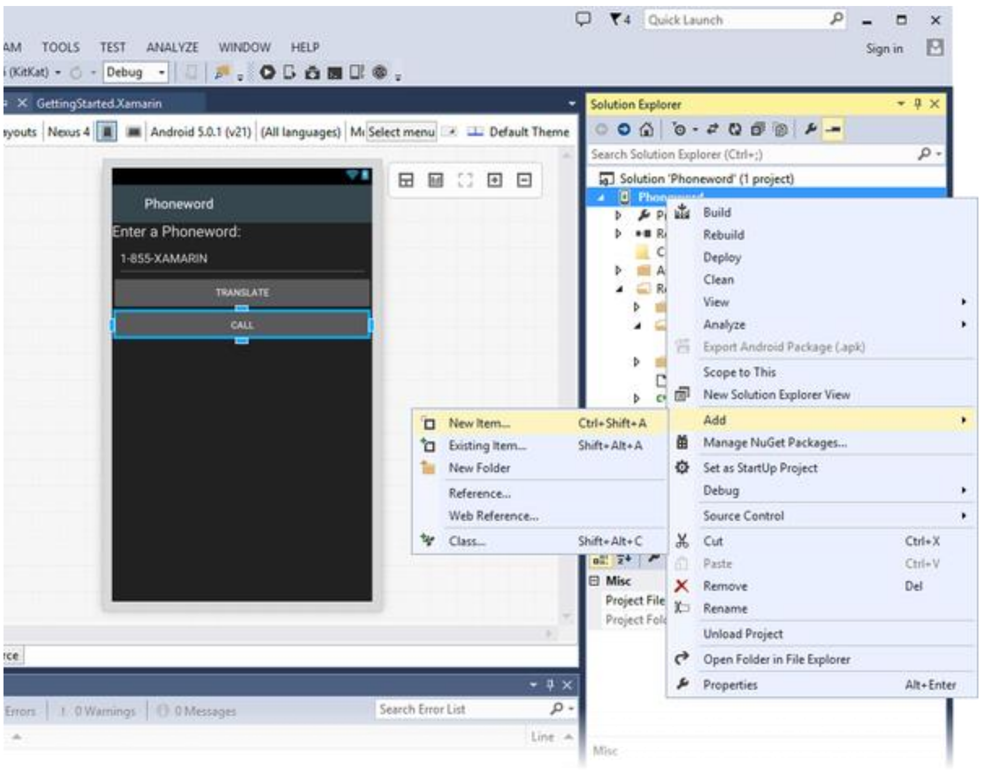


24. With the new *Button* control selected on the design surface, we can use the *Properties* pane to change the `id` property of the *Button* to `@+id/CallButton` and change the `text` property to `Call`:

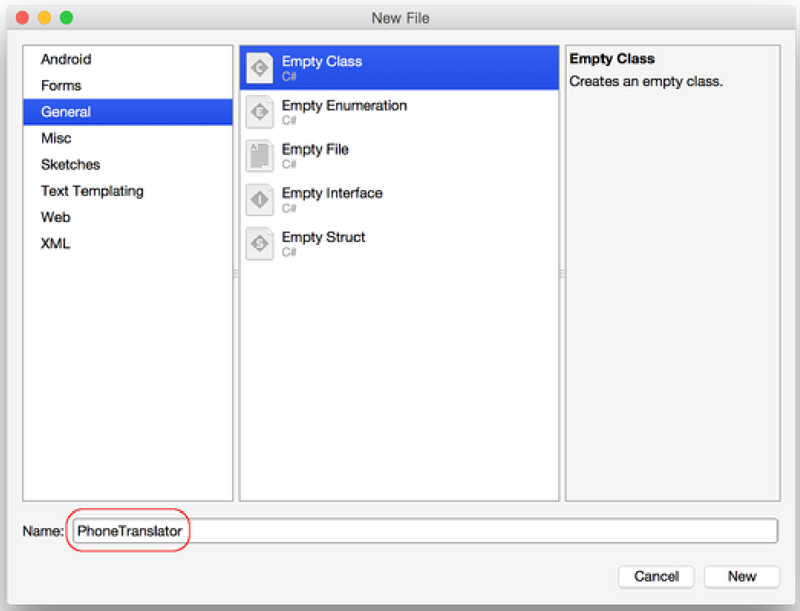


Let's save our work by pressing `CTRL+S`.

25. Now, let‘s add some code to translate phone numbers from alphanumeric to numeric. We‘ll add a new file to the project by right-clicking the *Phoneword* project in the *Solution Explorer* pane and choosing *Add > New Item...*:

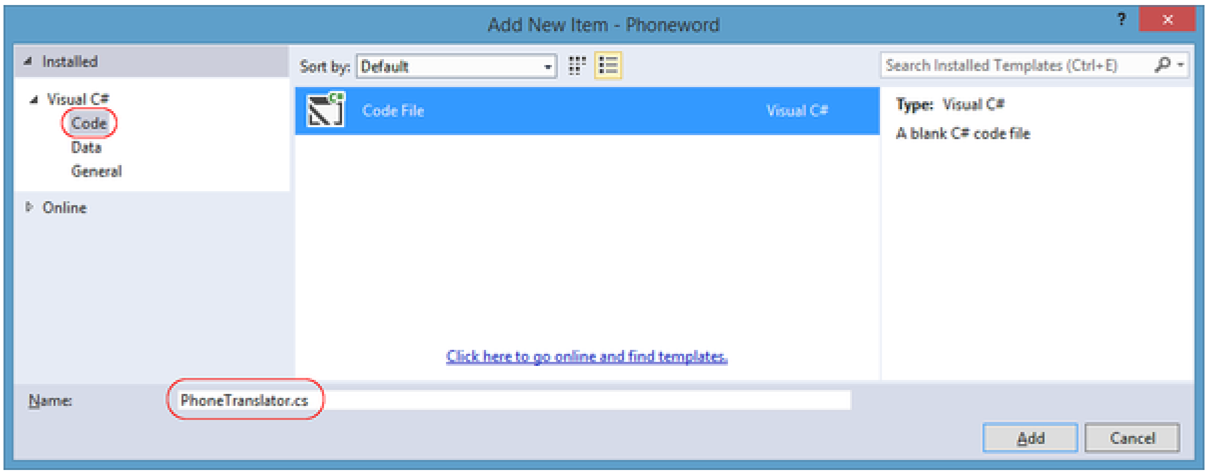


26. In the *New File* dialog, let's select *General > Empty Class*, name the new file **PhoneTranslator**, and click *New*:



Name: PhoneTranslator

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



Type: Visual C#

A blank C# code file

Name: PhoneTranslator.cs

28. This creates a new empty C# class for us. Let's remove all of the template code and replace it with the following code:

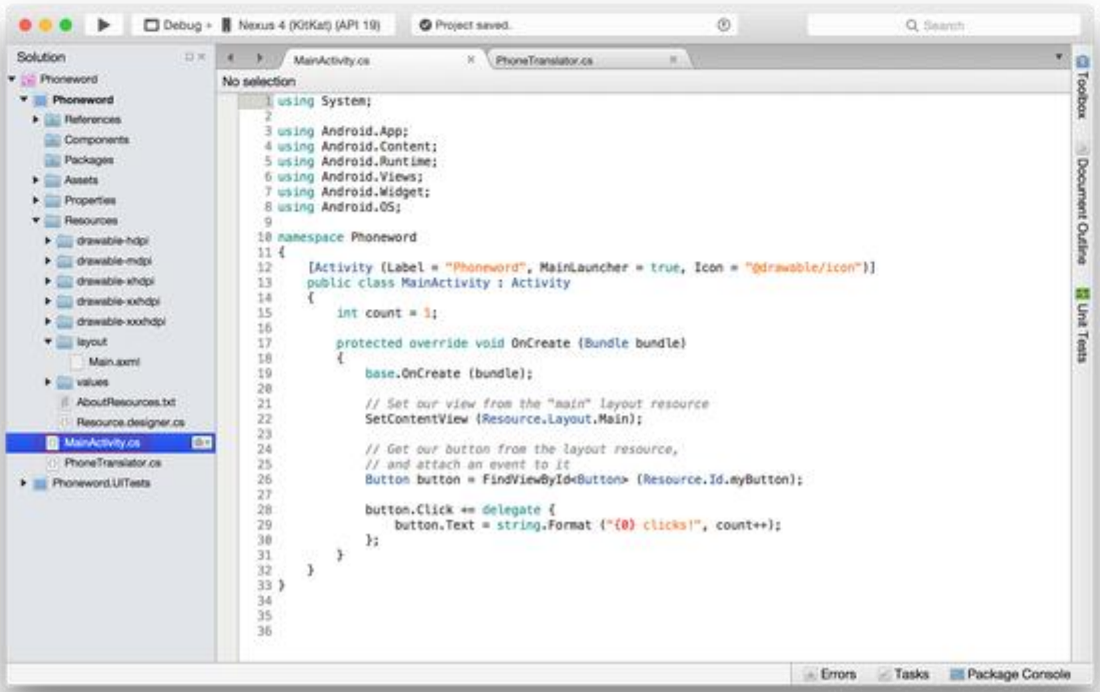
```
using System.Text;

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;
        }
    }
```

```
}
```

Let's save the changes to the **PhoneTranslator.cs** file by choosing *File > Save* (or by pressing `⌘ + S`), then close the file.

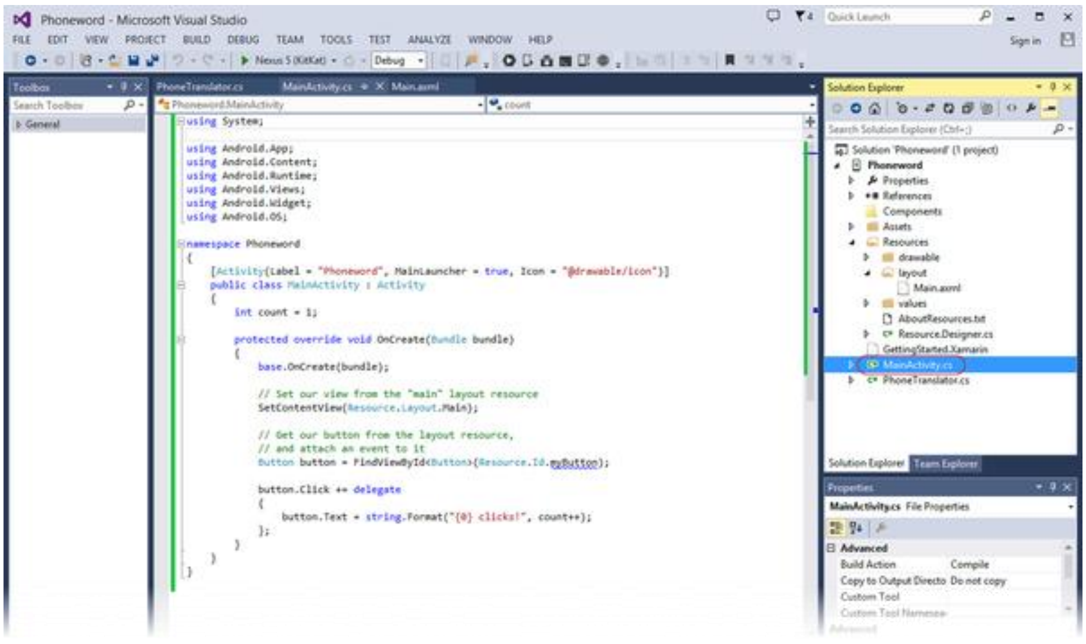
Let's save the changes to the **PhoneTranslator.cs** file by clicking *File > Save* (or by pressing `CTRL+S`), then close the file.

29. Next we‘re going to add code to wire up the user interface. Let‘s add the backing code into the `MainActivity` class. Double-click **MainActivity.cs** in the *Solution Pad* to open it:



```
using System;

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

namespace Phoneword
{
    [Activity (Label = "Phoneword", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        int count = 1;

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

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

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button> (Resource.Id.myButton);

            button.Click += delegate {
                button.Text = string.Format ("{0} clicks!", count++);
            };
        }
    }
}
```

30. Next we‘re going to add code to wire up the user interface. Let‘s add the backing code into the `MainActivity` class. Double-click **MainActivity.cs** in the *Solution Explorer* to open it:



```
using System;

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

namespace Phoneword
{
    [Activity(Label = "Phoneword", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        int count = 1;

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

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

            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button>(Resource.Id.myButton);

            button.Click += delegate
            {
                button.Text = string.Format("{0} clicks!", count++);
            };
        }
    }
}
```

31. We begin by wiring up the *Translate* button. In the `MainActivity` class, find the `OnCreate` method. We'll add our button code inside `OnCreate`, below the `base.OnCreate(bundle)` and `SetContentView (Resource.Layout.Main)` calls. Remove the template button handling code so that the `OnCreate` method resembles the following:

```
using System;

using Android.App;

using Android.Content;

using Android.Runtime;

using Android.Views;

using Android.Widget;

using Android.OS;


namespace Phoneword

{

    [Activity (Label = "Phoneword", MainLauncher = true)]

    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);

            // Our code will go here
        }
    }
}
```

32. Next, we need to get a reference to the controls that we created in the layout file with the Android Designer. Let's add the following code inside the `OnCreate` method, after the call to `SetContentView`:

```
// Get our 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);
```

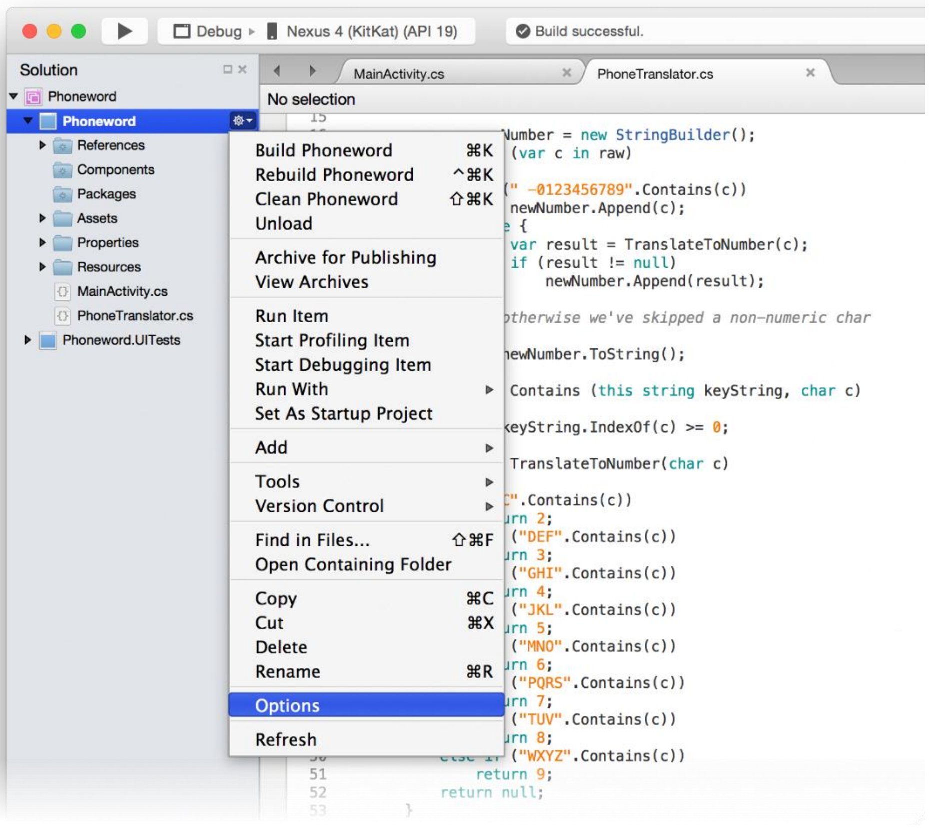
33. Now let's add code that responds to user presses of the *Translate* button. Add the following code to the `OnCreate` method (after the lines we added in the last step):

```
// 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;

    }

};
```

34. Next let‘s add code that responds to user presses of the *Call* button. We‘ll place the following code below the code for the *Translate* button:

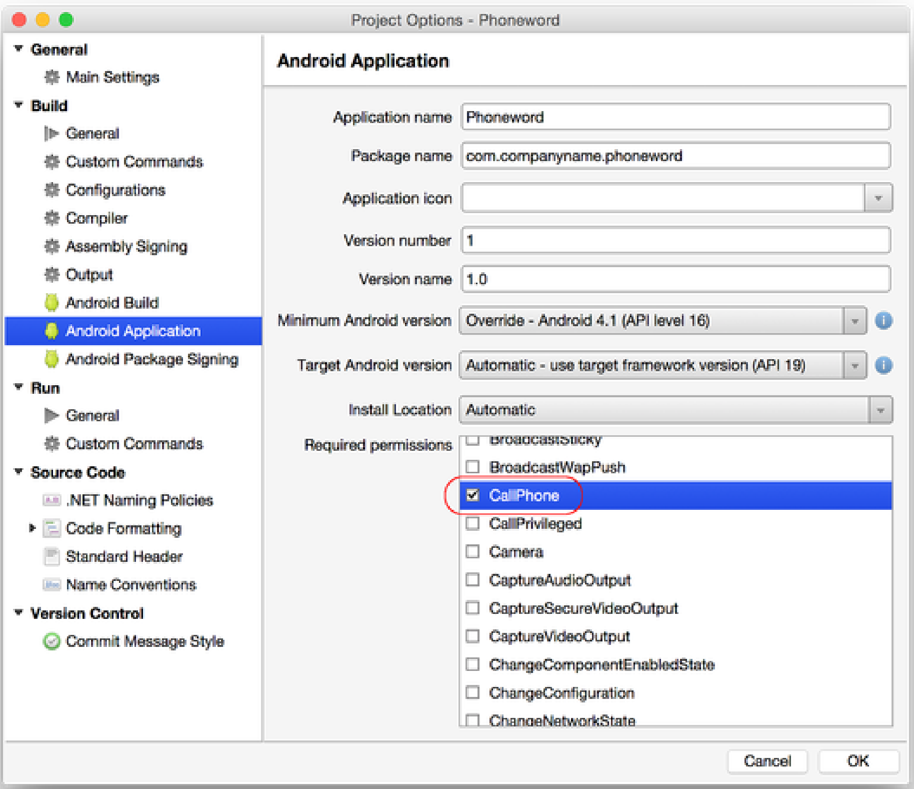
```
callButton.Click += (object sender, EventArgs e) =>

{

    // 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();
};
```

35. Finally, let‘s give our application permission to place a phone call. Open the project options by right-clicking *Phoneword* in the *Solution* pad and selecting *Options*:



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

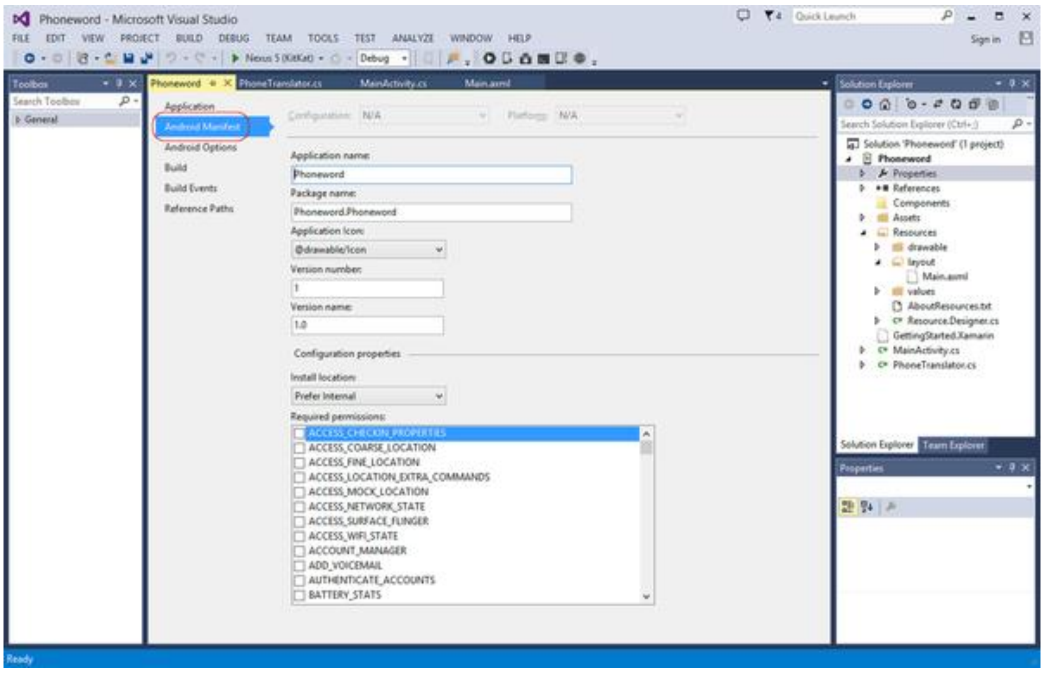
In the *Project Options* dialog, select *Build > Android Application*. In the *Required Permissions* section, enable the *CallPhone* permission:



Android Application

Application name: Phoneword
Package name: com.companyname.phoneword
Application icon:
Version number: 1
Version name: 1.0
Minimum Android version: Override - Android 4.1 (API level 16)
Target Android version: Automatic - use target framework version (API 19)
Install Location: Automatic
Required permissions:
- [ ] BroadcastSticky
- [ ] BroadcastWapPush
- [x] CallPhone
- [ ] CallPrivileged
- [ ] Camera
- [ ] CaptureAudioOutput
- [ ] CaptureSecureVideoOutput
- [ ] CaptureVideoOutput
- [ ] ChangeComponentEnabledState
- [ ] ChangeConfiguration
- [ ] ChangeNetworkState

36. Finally, let‘s give our application permission to place a phone call. We can edit app permissions in the Android Manifest. Open the Android Manifest by double-clicking *Properties* under *Phoneword* in the *Solution Explorer*, then select the *Android Manifest* page:



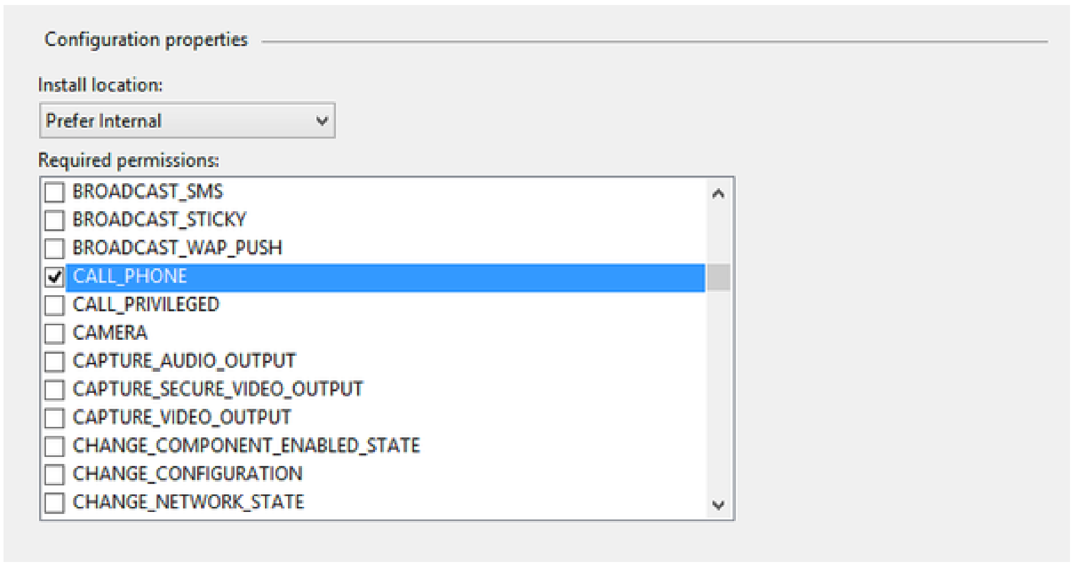
Configuration: N/A Platform: N/A

Application name: Phoneword
Package name: Phoneword.Phoneword
Application icon: @drawable/Icon
Version number: 1
Version name: 1.0

Configuration properties

Install location: Prefer Internal
Required permissions:
- [ ] ACCESS_CHECKIN_PROPERTIES
- [ ] ACCESS_COARSE_LOCATION
- [ ] ACCESS_FINE_LOCATION
- [ ] ACCESS_LOCATION_EXTRA_COMMANDS
- [ ] ACCESS_MOCK_LOCATION
- [ ] ACCESS_NETWORK_STATE
- [ ] ACCESS_SURFACE_FLINGER
- [ ] ACCESS_WIFI_STATE
- [ ] ACCOUNT_MANAGER
- [ ] ADD_VOICEMAIL
- [ ] AUTHENTICATE_ACCOUNTS
- [ ] BATTERY_STATS

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


Configuration properties

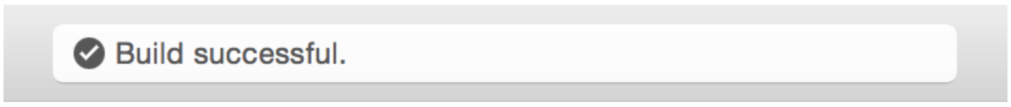
Install location:

Prefer Internal

Required permissions:

- [ ] BROADCAST_SMS
- [ ] BROADCAST_STICKY
- [ ] BROADCAST_WAP_PUSH
- [x] CALL_PHONE
- [ ] CALL_PRIVILEGED
- [ ] CAMERA
- [ ] CAPTURE_AUDIO_OUTPUT
- [ ] CAPTURE_SECURE_VIDEO_OUTPUT
- [ ] CAPTURE_VIDEO_OUTPUT
- [ ] CHANGE_COMPONENT_ENABLED_STATE
- [ ] CHANGE_CONFIGURATION
- [ ] CHANGE_NETWORK_STATE

37. Let's save our work and build the application by selecting *Build* > *Build All* (or by pressing `⌘ + B`). If our application compiles, we will get a success message at the top of Xamarin Studio:


Build successful.

If there are errors, we can 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 and Xamarin Studio updates.

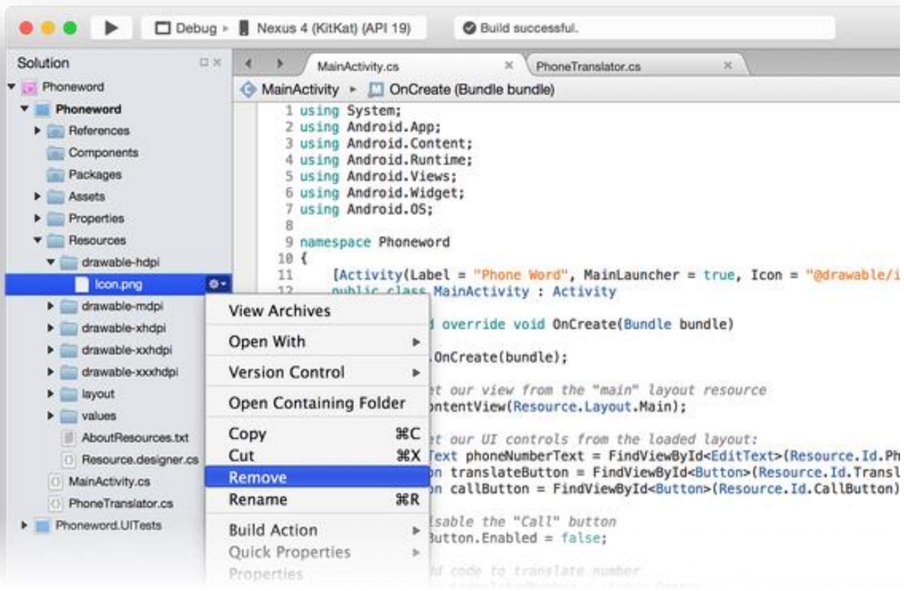
38. Let's save our work by selecting *File* > *Save All* (or by pressing `CTRL-SHIFT-S`) and build the application by selecting *Build* > *Rebuild Solution* (or by pressing `CTRL-SHIFT-B`). If our application compiles, we will get a success message in the bottom left corner of Visual Studio:

If there are errors, we can 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.

39. We now have a working application – it‘s time to add the finishing touches! Let‘s start by editing the `Label` for our `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 seen here:

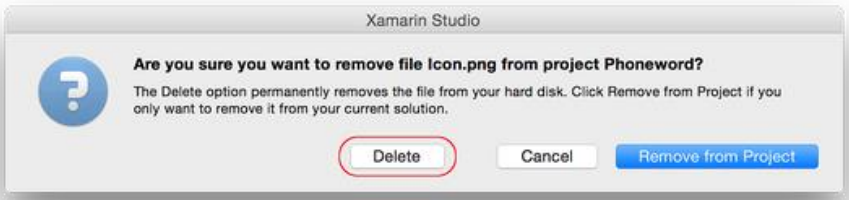
```
namespace Phoneword

{

    [Activity (Label = "Phone Word", MainLauncher = true)]

    public class MainActivity : Activity

    {

        ...

    }

}
```

40. Next, let's set the application icon. First, let's open the downloaded and unzipped Xamarin App Icons set. Next, let's expand the *drawable-hdpi* folder under *Resources* and remove the existing **Icon.png** by right-clicking it and selecting *Remove*:



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

namespace Phoneword
{
    [Activity(Label = "Phone Word", MainLauncher = true, Icon = "@drawable/i
    public class MainActivity : Activity
     override void OnCreate(Bundle bundle)
     OnCreate(bundle);
     t our view from the "main" layout resource
     ontentView(Resource.Layout.Main);
     t our UI controls from the loaded layout:
     Text phoneNumberText = FindViewById<EditText>(Resource.Id.Ph
     n translateButton = FindViewById<Button>(Resource.Id.Transl
     n callButton = FindViewById<Button>(Resource.Id.CallButton)
     isable the "Call" button
     Button.Enabled = false;
     dd code to translate number
```

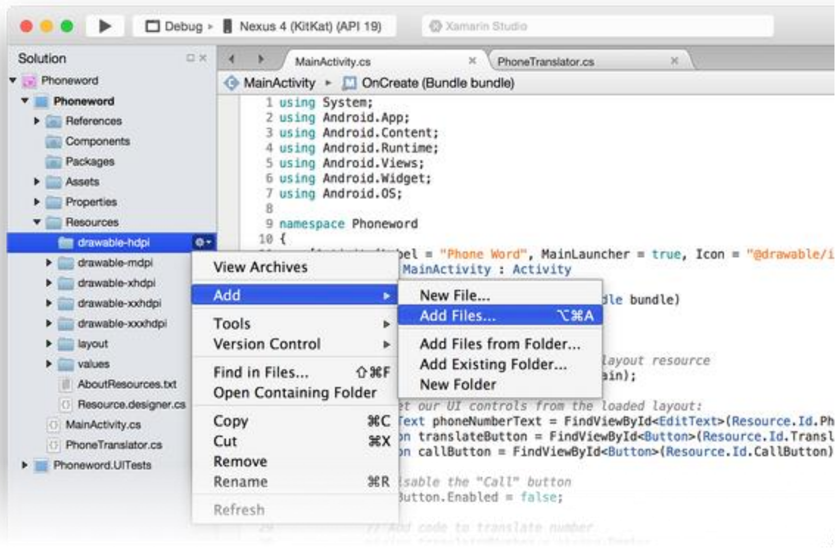
When the following dialog box is displayed, select *Delete*:



Are you sure you want to remove file Icon.png from project Phoneword?

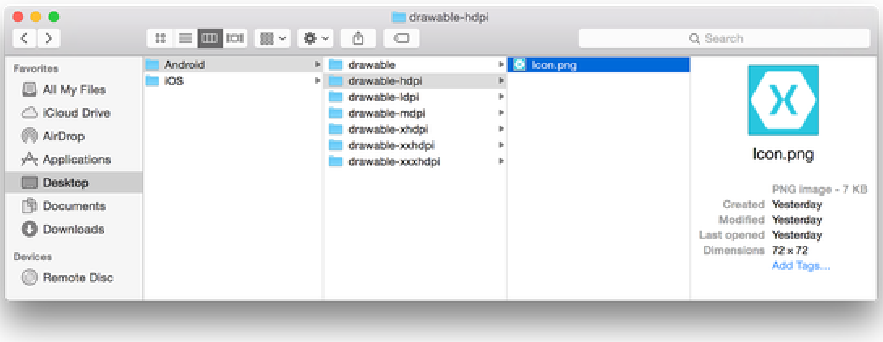
The Delete option permanently removes the file from your hard disk. Click Remove from Project if you only want to remove it from your current solution.

41. Next, let's right-click the *drawable-hdpi* folder and select *Add* > *Add Files*:

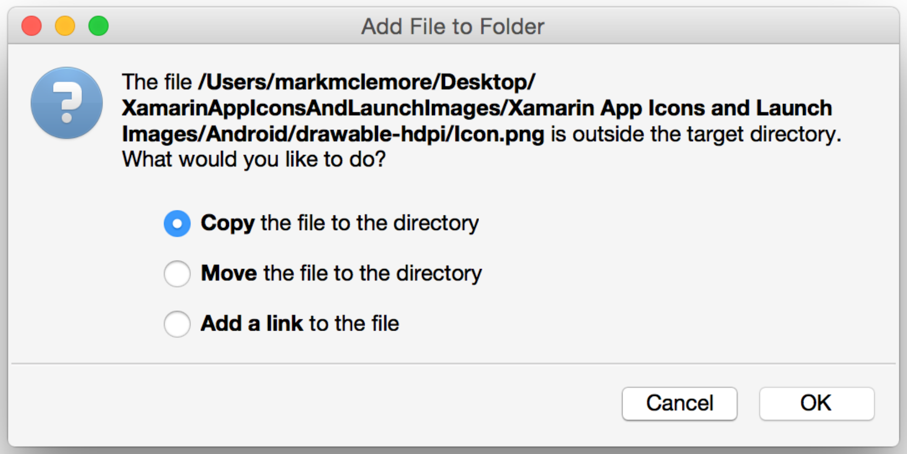


```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace Phoneword
{
     bel = "Phone Word", MainLauncher = true, Icon = "@drawable/i
     MainActivity : Activity
     dle bundle)
     layout resource
     ain);
     t our UI controls from the loaded layout:
     Text phoneNumberText = FindViewById<EditText>(Resource.Id.Ph
     n translateButton = FindViewById<Button>(Resource.Id.Transl
     n callButton = FindViewById<Button>(Resource.Id.CallButton)
     isable the "Call" button
     Button.Enabled = false;
     // Add code to translate number
```

42. From the selection dialog, let's navigate to the unzipped Xamarin App Icons directory and open the *drawable-hdpi* folder. Select **Icon.png**:



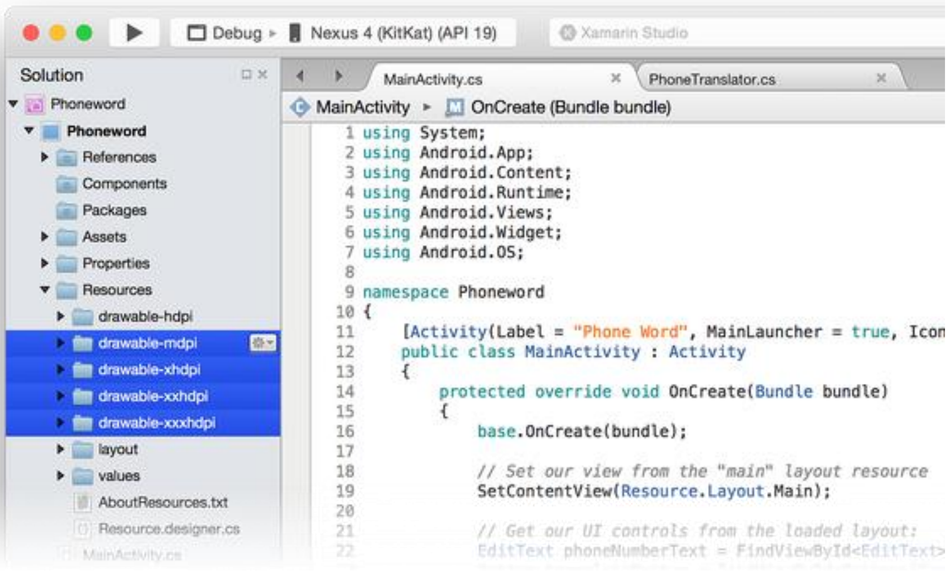
43. In the *Add File to Folder* dialog box, select *Copy the file into the directory* and click *OK*:


Add File to Folder

The file **/Users/markmclemore/Desktop/XamarinAppIconsAndLaunchImages/Xamarin App Icons and Launch Images/Android/drawable-hdpi/Icon.png** is outside the target directory. What would you like to do?

(●) **Copy** the file to the directory
( ) **Move** the file to the directory
( ) **Add a link** to the file

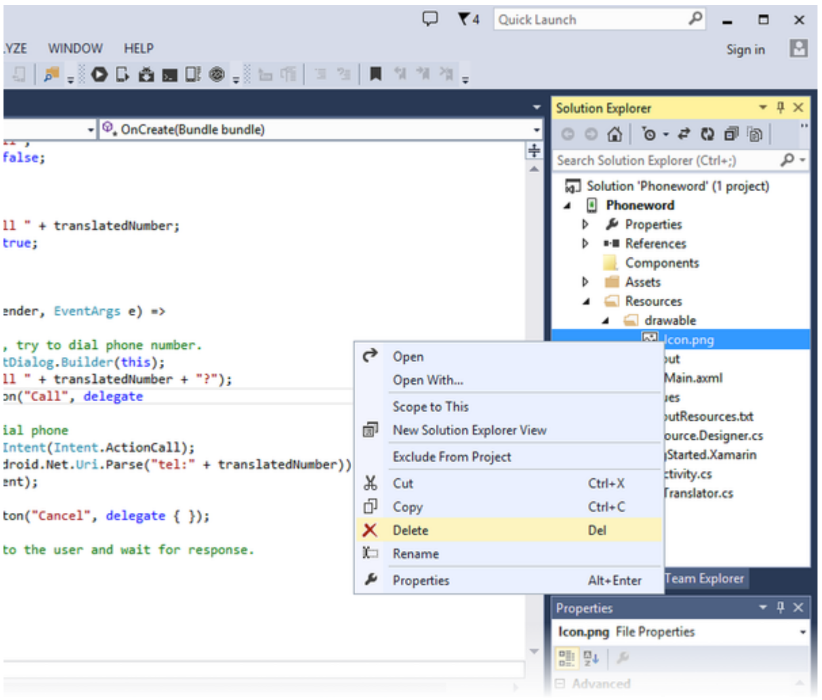
44. Repeat these steps for each of the *drawable-** folders until the contents of the *drawable-** Xamarin App Icons folders are copied to their counterpart *drawable-** folders in the **Phoneword** project:



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

namespace Phoneword
{
    [Activity(Label = "Phone Word", MainLauncher = true, 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);

            // Get our UI controls from the loaded layout:
            EditText phoneNumberText = FindViewById<EditText>
```

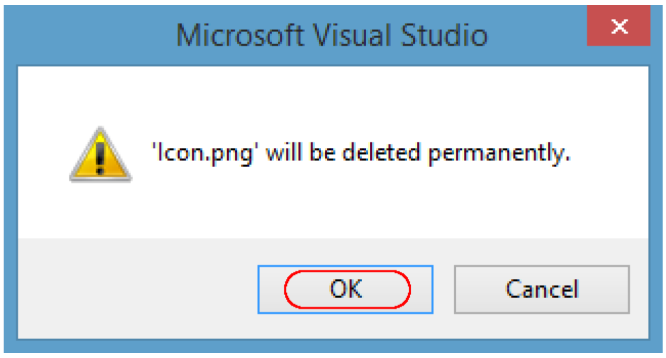
These folders provide different resolutions of the icon so that it renders correctly on different devices with different screen densities.

45. Next, let's 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*:



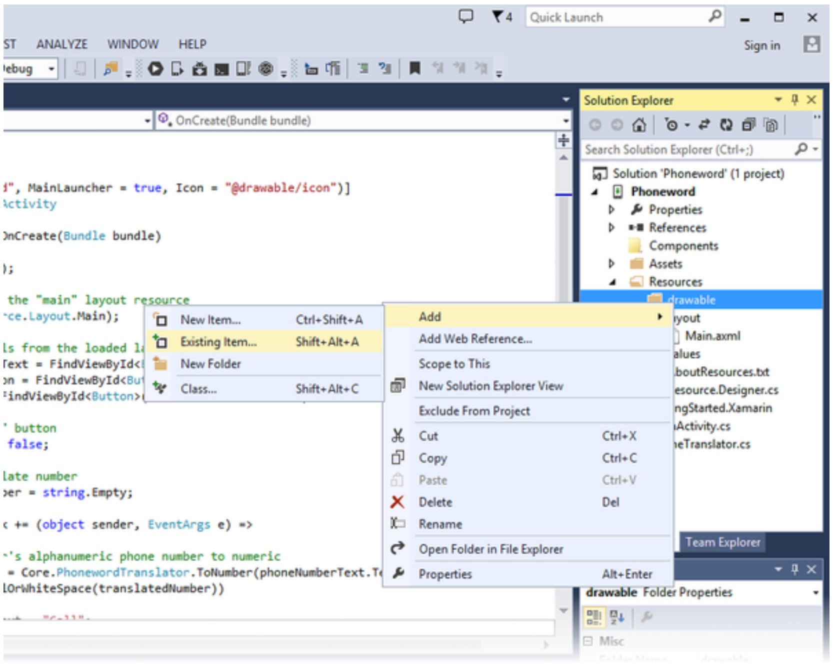
```
false;


ll " + translatedNumber;
true;



ender, EventArgs e) =>

, try to dial phone number.
tDialog.Builder(this);
ll " + translatedNumber + "?");
on("Call", delegate

ial phone
Intent(Intent.ActionCall);
droid.Net.Uri.Parse("tel:" + translatedNumber))
ent);

ton("Cancel", delegate { });

to the user and wait for response.
```

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

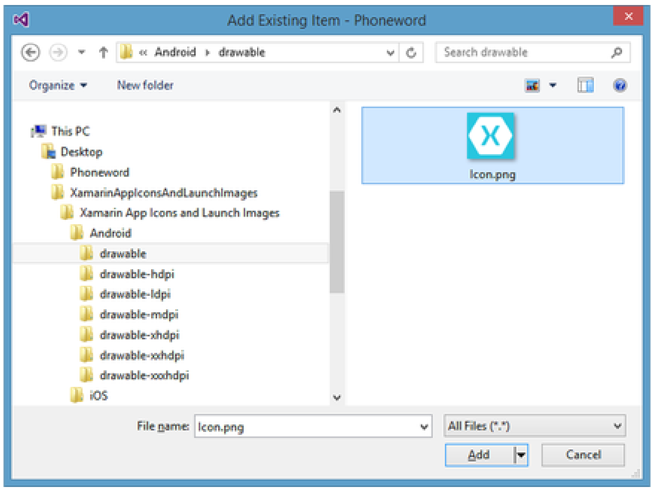

Microsoft Visual Studio

'Icon.png' will be deleted permanently.

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


```
d", MainLauncher = true, Icon = "@drawable/icon")]
Activity

OnCreate(Bundle bundle)

);

 the "main" layout resource
rce.Layout.Main);

ls from the loaded la
Text = FindViewById<
on = FindViewById<Bu
FindViewById<Button>(

' button
 false;

late number
ber = string.Empty;

k += (object sender, EventArgs e) =>

-'s alphanumeric phone number to numeric
 = Core.PhonewordTranslator.ToNumber(phoneNumberText.Te
lOrWhiteSpace(translatedNumber))
```

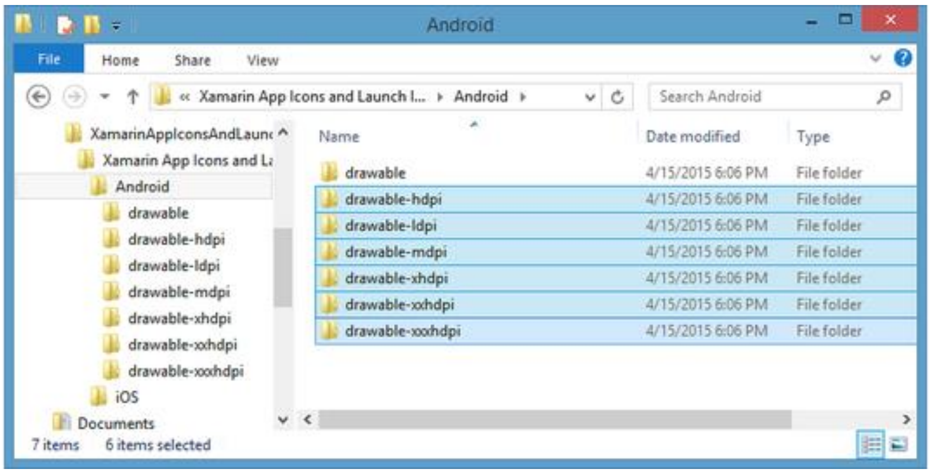
47. From the selection dialog, let's navigate to the unzipped Xamarin App Icons directory and open the *drawable* folder. Select **Icon.png** and click *Add*:


Add Existing Item - Phoneword

File name: Icon.png

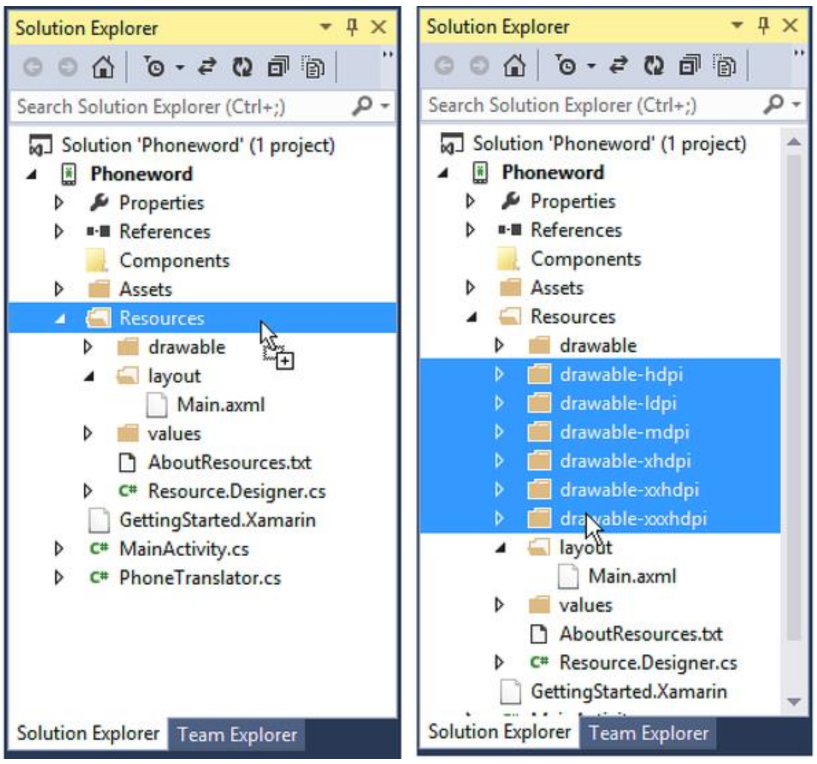
All Files (*.*)

48. Next, let's 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:

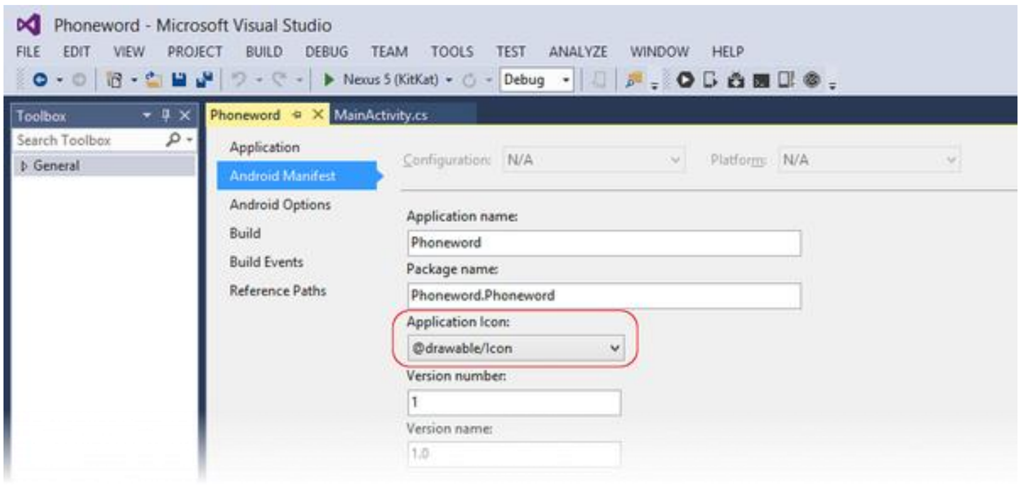


Name	Date modified	Type
drawable	4/15/2015 6:06 PM	File folder
drawable-hdpi	4/15/2015 6:06 PM	File folder
drawable-ldpi	4/15/2015 6:06 PM	File folder
drawable-mdpi	4/15/2015 6:06 PM	File folder
drawable-xhdpi	4/15/2015 6:06 PM	File folder
drawable-xxhdpi	4/15/2015 6:06 PM	File folder
drawable-xxxhdpi	4/15/2015 6:06 PM	File folder

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



49. Next, let's specify the icon in the Android Manifest by choosing `@drawable/Icon` from the *Application Icon* drop-down menu:


Application name: Phoneword

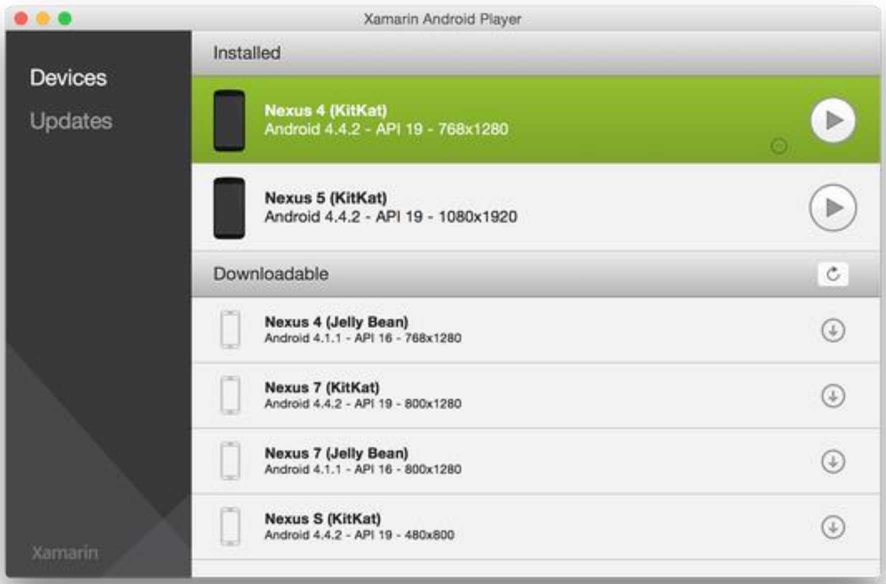
Package name: Phoneword.Phoneword

Application Icon: @drawable/Icon

Version number: 1

Version name: 1.0

50. Finally, we can test our application by deploying it to an Android emulator. If you have not yet configured your emulator, please see [Xamarin Android Player] for setup instructions. In this example, we have installed the *Nexus 4 (KitKat)* (Android 4.4.2, API Level 19) virtual device and we have started it from the Xamarin Android Player *Device Manager* console:


Installed

Nexus 4 (KitKat) Android 4.4.2 - API 19 - 768x1280

Nexus 5 (KitKat) Android 4.4.2 - API 19 - 1080x1920

Downloadable

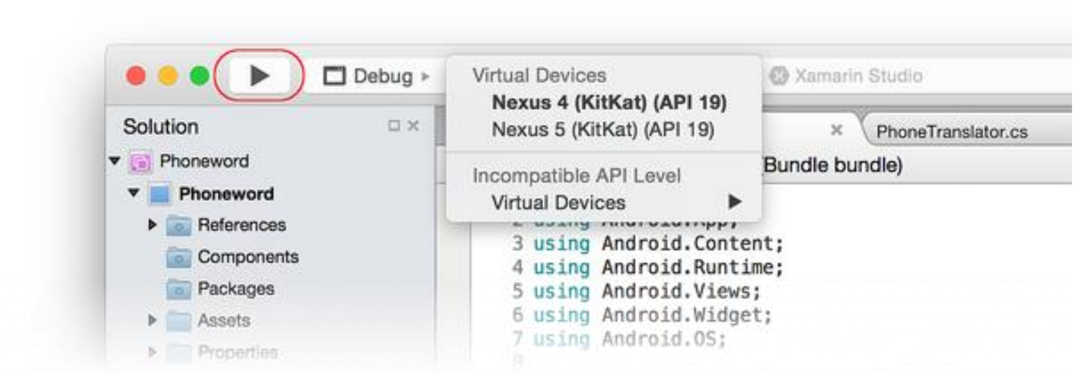
Nexus 4 (Jelly Bean) Android 4.1.1 - API 16 - 768x1280

Nexus 7 (KitKat) Android 4.4.2 - API 19 - 800x1280

Nexus 7 (Jelly Bean) Android 4.1.1 - API 16 - 800x1280

Nexus S (KitKat) Android 4.4.2 - API 19 - 480x800

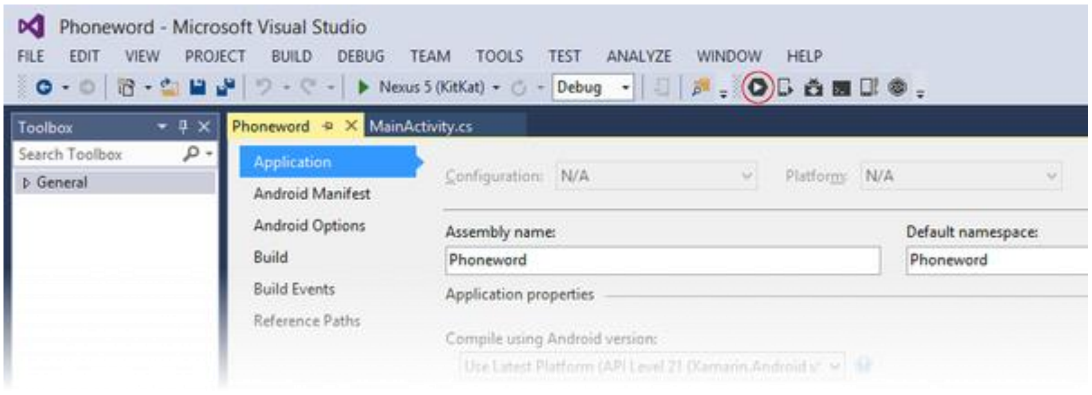
In Xamarin Studio, select this virtual device (under *Virtual Devices*) and click the play button in the upper left corner:



```
Bundle bundle)
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
```

As shown in this screenshot, we have selected the *Nexus 4 (KitKat) (API 19)* virtual device that is running in the Xamarin Android Player.

51. Finally, we can test our application by deploying it to an emulator. If you have not yet configured your emulator, please see [Xamarin Android Player]() for setup instructions. If you are using a Google Android emulator, please see [Accelerating Android Emulators]() for tips on improving emulation performance. Let's open the Xamarin Android Player *Device Manager* by clicking the *Manage virtual devices (XAP)* icon in the toolbar:



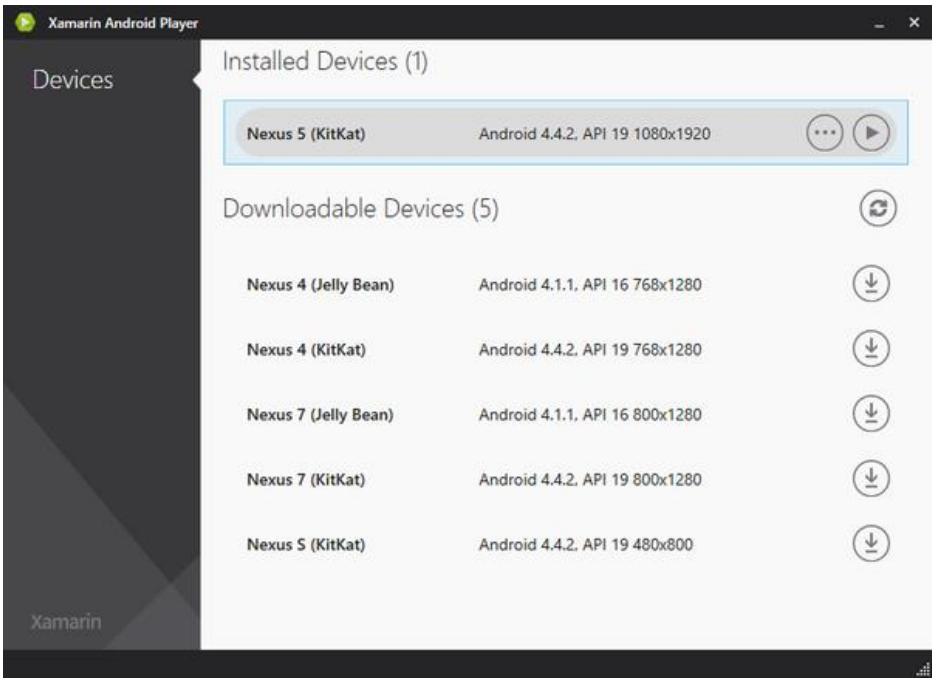
Configuration: N/A Platform: N/A

Assembly name: Phoneword

Default namespace: Phoneword

Compile using Android version: Use Latest Platform (API Level 21 (Xamarin.Android v

In the Xamarin Android Player *Device Manager*, let's download, install, and start the *Nexus 5 (KitKat)* (Android 4.4.2, API Level 19) virtual device:


Xamarin Android Player

Devices

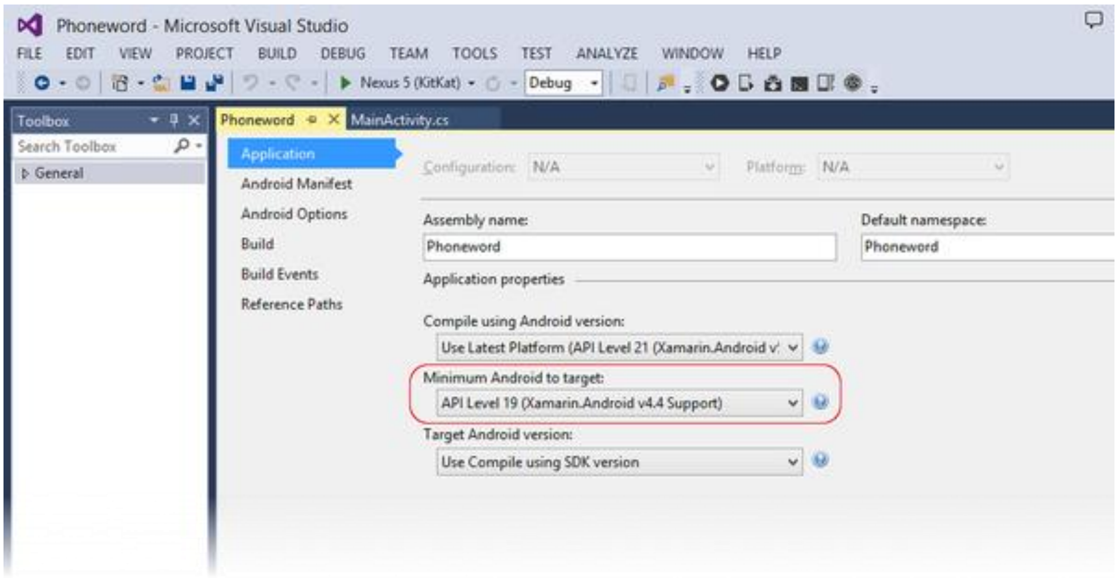
Installed Devices (1)

Nexus 5 (KitKat)	Android 4.4.2, API 19 1080x1920

Downloadable Devices (5)

Nexus 4 (Jelly Bean)	Android 4.1.1, API 16 768x1280
Nexus 4 (KitKat)	Android 4.4.2, API 19 768x1280
Nexus 7 (Jelly Bean)	Android 4.1.1, API 16 800x1280
Nexus 7 (KitKat)	Android 4.4.2, API 19 800x1280
Nexus S (KitKat)	Android 4.4.2, API 19 480x800

Before we send our app to this emulator, let's configure the app's minimum Android version so that it will run on our chosen virtual device. In Visual Studio, open the *Application* page of *Properties*. Under *Minimum Android to target:*, select API Level 19 (for more information about Android API levels, see Understanding Android API Levels).


Phoneword - Microsoft Visual Studio

Configuration: N/A
Platform: N/A

Assembly name: Phoneword
Default namespace: Phoneword

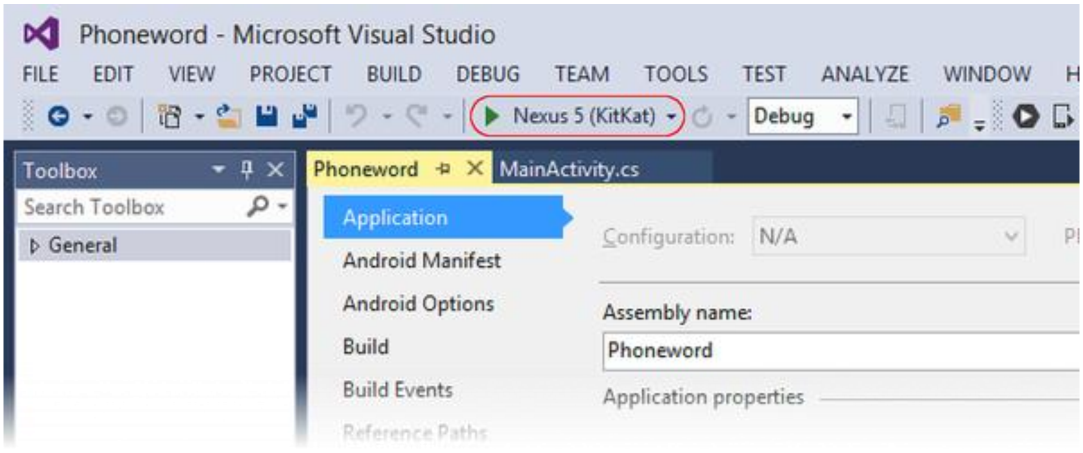
Application properties

Compile using Android version: Use Latest Platform (API Level 21 (Xamarin.Android v

Minimum Android to target: API Level 19 (Xamarin.Android v4.4 Support)

Target Android version: Use Compile using SDK version

Next, let's deploy our app to the *Nexus 5 (KitKat)* device from the drop-down menu on the toolbar as shown here:



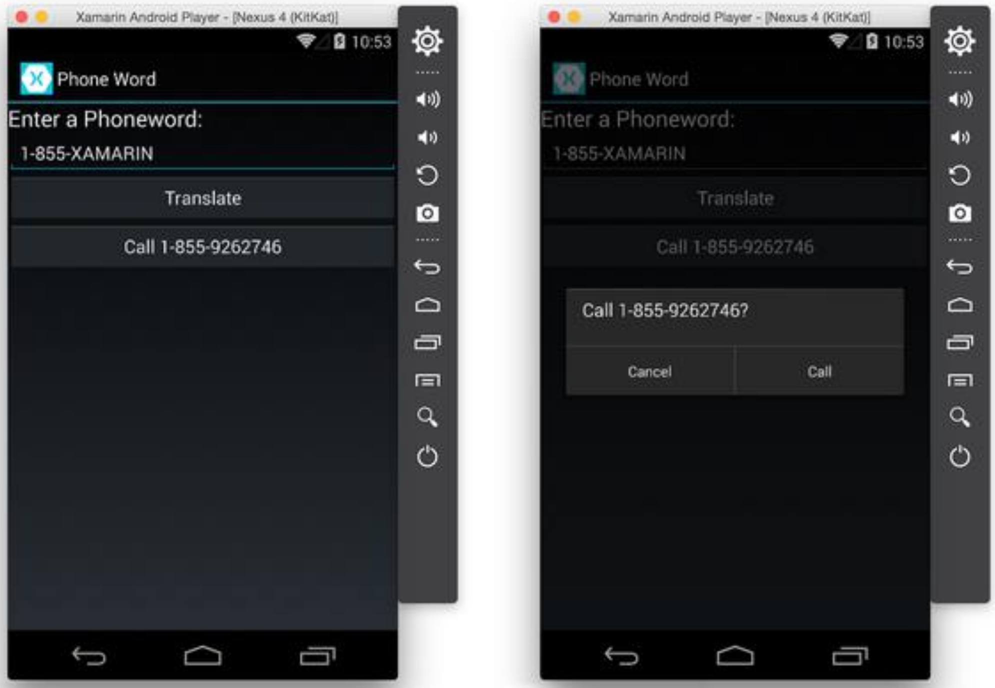
Configuration: N/A

Assembly name: Phoneword

Application properties

Visual Studio will copy files to this emulator before installing and launching our app.

52. After Xamarin Studio loads the application into the virtual device, the **Phoneword** app is automatically started. The screenshots below illustrate the **Phoneword** application running in the Xamarin Android Player. The icons that we installed are displayed next to the *Phone Word* label that we configured in `MainActivity`. 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:



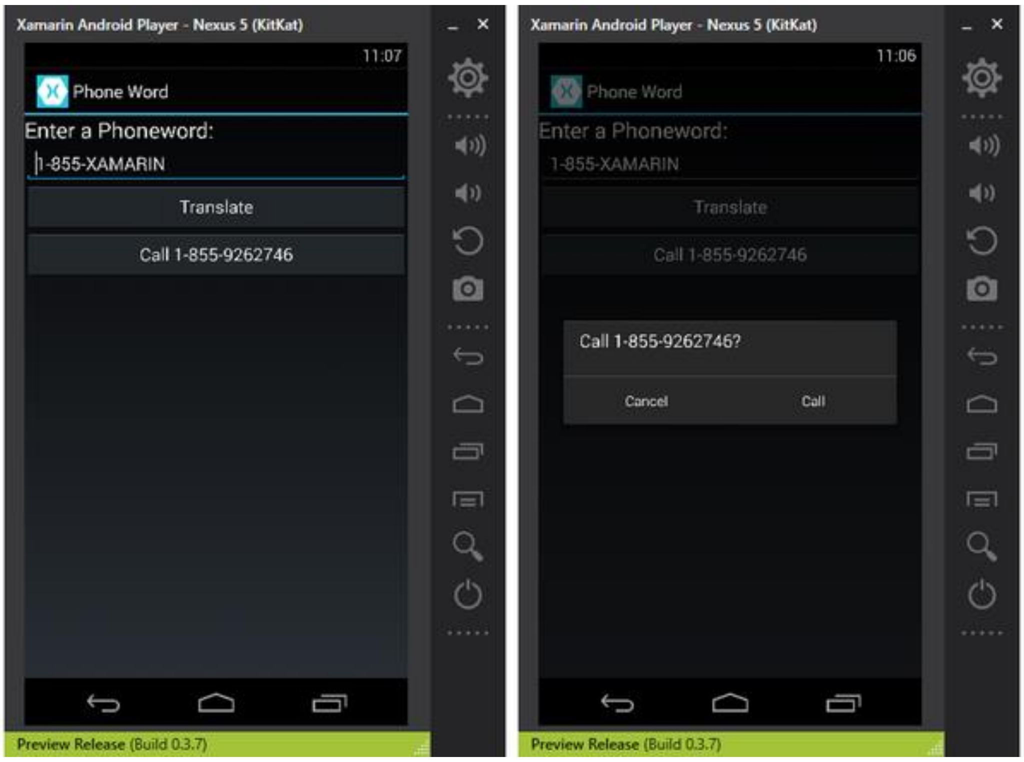
Enter a Phoneword:

1-855-XAMARIN

Call 1-855-9262746

Call 1-855-9262746?

53. The screenshots below illustrate the **Phoneword** application running in Xamarin Android Player. The icons that we installed are displayed next to the *Phone Word* label that we configured in `MainActivity`. 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:



Congratulations on completing your first Xamarin.Android application! Now it's time to dissect the tools and skills we just learned in the Hello, Android Deep Dive.