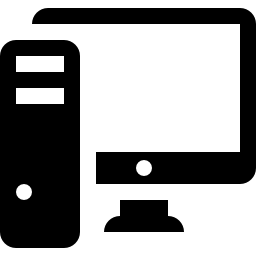
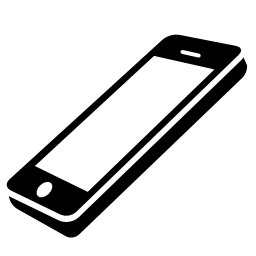
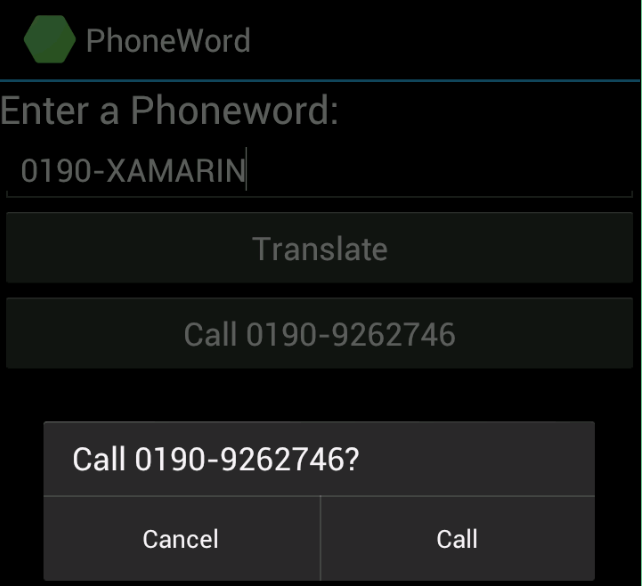
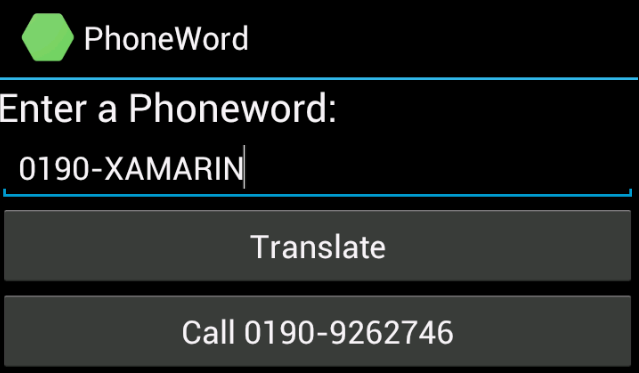
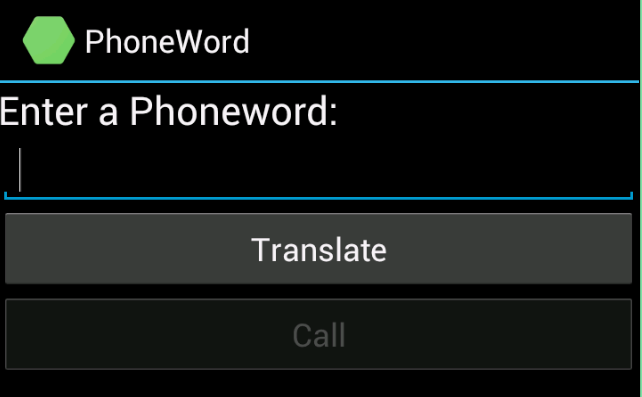
Developing Xamarin Applications



# Phoneword App

You can make yourself more comfortable in building apps with Xamarin and its associated tools and techniques with the Phoneword exercise from Xamarin. The app has the following requirements:

* Translates alphanumeric phone numbers into numeric numbers (0190-XAMARIN 🡪 0190-9262746).
* You can perform a phone call to the translated number.



The following guides for the Phoneword exercise will guide you step-by-step through the different processes how to build an app for Android and iOS:

* Android: [Introduction to Android Development with Xamarin](http://developer.xamarin.com/guides/android/getting_started/hello,android/)
* iOS: [Introduction to iOS Development with Xamarin](http://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/)

You don’t need to install any Xamarin components if you don’t want to. You can go through the guides and try to understand the implementation steps without implementing the app.

If you are encouraged to implement the app: please don’t copy / paste the PhonewordTranslator code form the iOS or Android guide. Try to implement the feature with Test Drive Development (see section 1.1).

|  |  |
| --- | --- |
| warning icon | The both applications will not share any code between the platforms and therefore you will have no gain from code-reusability between the platforms. It just gives you a basic overview of the concepts and steps involved. |

## Test Driven Development

Because we are professional developers, we will first write a test before implementing our features.  
If you haven’t installed the required components please refer first to section 3.

Our architect provided us with the highly sophisticated interface IPhonewordTranslator, which defines the behavior hot the translation should occur.

/// <summary>

/// Phoneword translator, which provides functions to translate phone numbers.

/// </summary>

public interface IPhonewordTranslator

{

/// <summary>

/// Translate a alphanumeric number to a numeric number.

/// </summary>

/// <returns>The numeric number.</returns>

/// <param name="alphanumericNumber">Alphanumeric number to be translated.</param>

string ToNumericNumber(string alphanumericNumber);

}

Copy the interface definition into your Core project. Also create a first dummy implementation of the interface which always returns an empty string.

### Write your unit test

For that we will create a NUnit test project, where we will put our tests.

Now write your first unit test, which will do nothing more than just check if your dummy method returns an empty string.

If you are not familiar in writing unit tests with NUnit, check the following [Quick Start guide](http://www.nunit.org/index.php?p=quickStart&r=2.2.10).  
One common pattern in writing unit tests is the Arrange Act Asser (AAA) pattern. A quick explanation can be found [here](https://store.xamarin.com/account/my/subscription?product=Xamarin.Androidhttp://c2.com/cgi/wiki?ArrangeActAssert).

When you have written your test run all the tests in Xamarin Studio with “Ctrl + T” or in Visual Studio.

Our implementation is very unproductive at the moment. Rewrite or add additional tests so that you can test the proper behavior.

When you have a closer look to the specification / user story for this feature you will find out that the conversion should happen as follows:

|  |  |
| --- | --- |
| Letter | Converted Number |
| a, b, c | 2 |
| d, e, f | 3 |
| g, h, i | 4 |
| j, k, l | 5 |
| m, n, o | 6 |
| p, q, r, s | 7 |
| t, u, v | 8 |
| w, x, y, z | 9 |

You should also consider the following cases:

* Lower and uppercase letters should be handed similarly.
* Spaces and dashes should be allowed (e.g. 0190 555-222-1 is valid).
* Other or Special Characters should be skipped.

As a last hint, have a look at the [Row Test feature](http://www.nunit.org/index.php?p=testCase&r=2.5) in NUnit. It might help you running the same test with different input.

Of course your tests will fail. Now you need to implement the desired behavior in your interface implementation. When you implemented the functionality, run your tests again to see that everything works as expected.

# Phoneword App (Portable Class Library + Xamarin.Forms)

Feel free to go further and extend the Phoneword app from the previous exercise with the following features:

* Make the business logic reusable (as a Portable Class Library) for the different target devices
* Adding Model-View-ViewModel (MVVM) architecture and make use of Xamarin.Forms
* Add a history page, where we can see all the dialed numbers
* The translated number should be narrated from the phone’s speech engine

The source code for this exercise is already available on [GitHub](https://github.com/saxos1983/Xamarin-Talks/tree/exercise/Phoneword.Forms). All the required implementation steps are marked with TODO comments. To see all TODO’s press Alt-Shift-T in Xamarin Studio or Ctrl-Alt-D in Visual Studio. If you haven’t installed the required components please refer first to section 3.

# Installation of required components

For a hassle-free and efficient development experience of Cross-Plattform Mobile Apps with Xamarin, the following components are required:

* Xamarin Subscription (Xamarin Trial Account or higher)
  + <https://store.xamarin.com/account/register>
* Xamarin Components with the Xamarin Installer
  + [Download the Xamarin Platform](http://xamarin.com/download)
* Xamarin apps can be built with Visual Studio or Xamarin Studio. For the Visual Studio Integration at least Visual Studio 2010 and a business license is required.
* Depending of your desired targets you will need additional software packages or hardware requirements. Here you will find more detailed installation manuals for your target and development platform.
  + Android: [Installing Xamarin.Android](http://developer.xamarin.com/guides/android/getting_started/installation/mac/)
    - Note: Android Emulators are usually very slow when no Hardware Acceleration is installed and enabled ([How-To](https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-windows))
  + iOS: [Installing Xamarin.iOS](http://developer.xamarin.com/guides/ios/getting_started/installation/)
  + Windows Phone: You will need Visual Studio and the [Windows Phone SDK](http://dev.windows.com/en-us/develop/download-phone-sdk)