Xamarin Fast Start

Lab: Introduction to Xamarin.Forms

Prepared by: Bruno.Oliveira@microsoft.com

Date: Monday 4 July 2016

Template version 1.1

Table of Contents

Exercise 01: Creating Xamarin.Forms Phoneword

Duration: 30 minutes

Goal

The goal will be to create a new Xamarin.Forms application which displays a UI to translate and dial a phone number. The UI will consist of a: Label, Entry, and two Buttons.

Our steps for this exercise be broken into two separate sections - first, creating the UI by:

1. Create the UI using a *StackLayout*, *Label*, *Entry* and two *Button* elements.

Next, we will add some behavior that will:

2. Respond to a Click event on the Translate Button.
3. Retrieve the text typed into the Entry.
4. Translate the alphanumeric text into a traditional phone number, where ABC ➞ 2, DEF ➞ 3, etc.
5. Update the text of the Call Button to include the translated numeric phone number; we will also update the enabled state so the user can tap the button, although we won't add any behavior until the next exercise.

Required Assets

The *Exercise 01* folder contains C# source files named ***PhoneTranslator.cs***, which you will add to your project.

Steps

Create the initial project

The first step is to create a new Xamarin.Forms project. You can use either Visual Studio (shown here) or Xamarin Studio.

1. Create a new solution and select the "Blank App (Xamarin.Forms Portable)" project type from the ***Cross-Platform*** category under (Visual) C#. ("Forms App" from the App category under Multiplatform in Xamarin Studio.)

2. Name the project "Phoneword" and place it into a known location.

    a. If you are using Xamarin Studio, when configuring your new project, you will need to deselect the "Use XAML for the user interface files" checkbox as you did in Exercise 1 to get your project to match.

Add PhoneTranslator to our project

We need to add the code to translate the alphanumeric phone number into a numeric phone number. The code has already been written for you and is in the ***Exercise 01/PhoneTranslator.cs*** source file.

1. Right-click the shared project node (this should be named ***Phoneword*** and be in the root of the solution) and select ***Add > Add Files...*** in Xamarin Studio, or ***Add > Add Existing Item...*** in Visual Studio.

    a. **Note**: this is a sharable C# file - make sure you place it into the shared project (PCL or shared library) which is used with all three applications.

2. Navigate to the Exercise 1 folder and select the *PhoneTranslator.cs* file to add it to the project.

    a. In Xamarin Studio, use the default Copy choice.

3. Open the file and examine the contents - it has a single public static method, which you will be using to translate the number from alphanumeric text into a straight numeric number.

Restructure the Boilerplate code

As a first step, let's make the code a little more maintainable by restructuring the shared code a little.

1. Open the ***App.cs*** file which was created as part of the template, this is the main file which creates the initial UI. It might have a different name (such as Phoneword.cs) depending on the environment you are using.

2. Take the code which creates the page and move it to a new class that derives from ContentPage - name the class ***MainPage***. Place the creation code into the constructor.

3. Change the App constructor to now instantiate your new ***MainPage*** object and assign it to the *MainPage* property.

4. Run the application and make sure it still builds and executes. You can run the application on any of the three platforms, for now just pick the easiest platform for your configuration.

```
// App.cs
public class App : Application
{
    public App()
    {
        // The root page of your application
        MainPage = new MainPage();
    }
...
}

// MainPage.cs
public class MainPage : ContentPage
{
    public MainPage()
    {
        Content = new StackLayout
        {
            VerticalOptions = LayoutOptions.Center,
            Children = {
                new Label {
                    HorizontalTextAlignment = TextAlignment.Center,
                    Text = "Welcome to Xamarin Forms!" }
            }
        }
    }
}
```

Create the UI for Phoneword

Next, we need to describe the user interface for *Phoneword*. Remember, we are using controls defined in Xamarin.Forms which will then be rendered with native implementations in each project.

The controls you need to create are:

- A ***Label*** with the Text set to "Enter a Phoneword:". Make it large by setting the *FontSize* to *Device.GetNamedSize(NamedSize.Large, typeof(Label))* which is a device-specific size (e.g. sized for the runtime environment).
- An ***Entry*** which is used to collect the input from the user. It should be initialized with the text "**1-855-XAMARIN**".
- A ***Button*** with the title "*Translate*" which will activate the number translation.
- A ***Button*** with the title "*Call*", which will eventually call the number. This should be initially disabled.
- You should store the ***Entry*** and the two ***Button*** controls in fields so you can interact with them later.

You should use a ***StackLayout*** container in the vertical orientation to arrange these controls in the UI top-to-bottom. Note that Vertical is the default orientation for ***StackLayout***.

Use the *StackLayout's Spacing* property to give the Children a 15-unit space between them.

The entire page should have some padding. Give it a thickness of 20 units on all edges.

```
public class MainPage : ContentPage
{
    Entry phoneNumberText;
    Button translateButton;
    Button callButton;

    public MainPage()
    {
        this.Padding = new Thickness(20, 20, 20, 20);

        StackLayout panel = new StackLayout
        {
            Spacing = 15
        };

        panel.Children.Add(new Label
        {
            Text = "Enter a Phoneword:",
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
        });

        panel.Children.Add(phoneNumberText = new Entry
        {
            Text = "1-855-XAMARIN",
        });

        panel.Children.Add(translateButton = new Button
        {
            Text = "Translate"
        });

        panel.Children.Add(callButton = new Button
        {
            Text = "Call",
            IsEnabled = false,
        });

        this.Content = panel;
    }
}
```

Implement the Logic for the Translate Button

We need to handle the Clicked event for our TranslateButton - this will be raised when the user taps on the translate button in the UI and is the abstracted event for all the platform variations. The handler should:

1. Get the phone number from the *Entry* control.
    a. Hint: use the Text property.

2. Translate it using the *PhonewordTranslator.ToNumber* method and store it into a field named *translatedNumber*.

    a. You will need to add a namespace to access the ***PhonewordTranslator*** class.

    b. The ***PhonewordTranslator.ToNumber*** method returns *null* if the number could not be translated.

3. Change the text of the *Call Button* to include the phone number when it is successfully translated, just use the value you stored in the *translatedNumber* field.

    a. Hint: assign the created string to the Text property.

4. Enable the *Call Button* so it can be tapped to dial the phone.

    a. Hint: use the IsEnabled property.

    b. Make sure to reset the title of the *CallButton* if the number translation fails - e.g. remove the phone number.

```
public class MainPage : ContentPage
{
    ...
    string translatedNumber;

    public MainPage()
    {
        ...
        translateButton.Clicked += OnTranslate;

        this.Content = panel;
    }
    void OnTranslate(object sender, System.EventArgs e)
    {
        translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
        if (!string.IsNullOrEmpty(translatedNumber))
        {
            callButton.IsEnabled = true;
            callButton.Text = "Call " + translatedNumber;
        }
        else
        {
            callButton.IsEnabled = false;
            callButton.Text = "Call";
        }
    }
}
```

Test the Application

1. The **Translate Button** is now wired with behavior, so go ahead and test it. Run the application in either the simulator or on a physical device.

2. The application should build and if everything has gone well, our app will display and show a disabled Call Button.

3. Touch the Translate button in our app and verify the Call button title changes to "Call ***1-855-9262746***", and the button becomes enabled.

4. Try changing the phone number in the Text Field. Notice that iOS pops up an on-screen keyboard automatically. It should dismiss either when you tap the Translate button, or when you press the Return key on the iOS keyboard. This behavior is built into Xamarin.Forms and makes the keyboard experience consistent across the platforms.

5. If you have time, try running it on the other platforms to see the consistent UI and experience you have created.

Exercise 02: Adding support for dialing the phone

Duration: 30 minutes

Goal

Our goal is to add platform-specific UI for iOS rendering and to add behavior into our Phoneword application to display an alert and let the user dial the phone.

Required Assets

There are several assets you will be adding to your project in the ***Exercise 02*** folder to dial the phone on each of the platforms:

- ForPlatform.cs
- PhoneDialer.iOS.cs
- PhoneDialer.Droid.cs
- PhoneDialer.UWP.cs
- PhoneDialer.WinPhone.cs
- PhoneDialer.Windows.cs

There is also a completed project with all the changes for every platform if you'd like to compare your work at the end.

Steps

Adapt with Per-Platform UI

You may have noticed the Phoneword page was slightly more crowded, visually, with the status bar on iOS than it was be on other platforms. You will fix that now.

1. We will be using OnPlatform to give 20 extra device pixels padding on iOS devices (for a total of 40). The OnPlatform methods provide compile-time values that can change based on the running platform.

```
// The generic specification <T> can typically be inferred,
// it is shown here for clarity.
double paddingThickness = new Thickness(20,
                    Device.OnPlatform<double>(40, 20, 20), 20, 20);
```

While Xamarin.Forms offers OnPlatform to offer per-platform functionality for iOS, Android, or Windows, if you need enough granularity to separately address Windows 8 Silverlight devices from newer Windows devices, you will want something offering additional parameters. Inside the ***Exercise 02*** folder is ForPlatform.cs. The ForPlatform class offers methods that can return distinct values or execute distinct code specific to these platforms as well.

2. Run the application and verify the UI has the proper layout and controls are all visible. On iOS, with the extra padding

Prompt the User when the Call Button is pressed

First, let's add some behavior to the CallButton. In this case, we will need to perform the following tasks:

3. Subscribe a handler to the Click event of the Call Button.
4. Prompt the user using Page.DisplayAlert to ask if they'd like to dial the number.
    a. This method is Task-based, so use async and await to work with it.
    b. It takes a title, message and the string for the *accept* and *cancel* buttons.
    c. It returns a bool indicating whether the *accept* button was pressed to dismiss the dialog.
5. Use the parameters:
    a. Title = "Dial a Number"
    b. Message = "Would you to call xxxx", using the translated number.
    c. "Yes" and "No" for the buttons.
6. An alternative to using Page.DisplayAlert would be to abstract out your message, much like we will do with the dialer. However, in this instance the API does exactly what we want - presents an OK/Cancel choice and allows the user to select one, so we'll use it.
7. Go ahead and run the application using the easiest platform for your configuration to test the code - it should display the alert and allow you to dismiss it with either a Yes or No response.

Implement platform-specific logic to dial the phone

Next, let's create the abstraction for dialing the phone and add it into each of our platform-specific projects.

1. Create an interface abstraction for dialing the phone - name it ***IDialer***. It should have a single method called Dial that takes a string and returns a *bool*.
    a. This interface should be placed in the shared project - then we will provide an implementation in each of the platform projects.

```
// IDialer.cs
public interface IDialer {
    Task<bool> DialAsync(string number);
}
```

2. There are pre-supplied implementations of the ***IDialer*** interface in the ***Exercise 02*** folder. Copy each one into the appropriate project.

   ✓ **iOS** -> PhoneDialer.iOS

   ✓ **Android** -> PhoneDialer.Droid

   ✓ **Windows 10 (UWP)** -> PhoneDialer.UWP

   ✓ **Windows 8.1 desktop** -> PhoneDialer.Windows

   ✓ **Windows 8.1 phone** -> PhoneDialer.WinPhone

3. Verify that your interface declaration matches the implementation in the provided files. Note that the Windows 8.1 desktop version displays a message prompt since there is no dialer/phone support on the desktop.

4. In each implementation, add a ***[Dependency]*** assembly-level attribute to register the implementation. These can be placed into the same source file as the implementation of each dialer, but must be placed outside any namespace declaration - check the code hint below if you need some help.

```
using Phoneword.iOS;

// This same attribute definition needs to go in each platform-specific file.
[assembly: Dependency(typeof(PhoneDialer))]

namespace Phoneword.iOS
{
    public class PhoneDialer : IDialer
    {
    ...
    }
}
```

Add permissions to Android and UWP

Android applications that dial the phone must have the "call phone" permission in their manifest. On the Windows side, UWP apps require both a permission setting and a contract extension be added to the project. If you are implementing an Android or UWP version of the app, you will need to add this permission if it is not already set.

Android - Xamarin Studio

1. Open the ***Project Options*** for the Android project.

2. Click on ***Build*** > ***Android Application***.

3. In the Required permissions section, check the box next to the ***CallPhone*** permission.

4. If the other fields are not filled out (depending on your environment, the template might leave them blank), then supply valid values using the below screen shot as an example.

Android - Visual Studio

1. Open the ***Properties*** for the Android project.
2. Click on ***Android Manifest***.
3. In the Required permissions section, check the box next to the ***CALL_PHONE*** permission.

Windows Phone 10 - Visual Studio

If you are testing the UWP application, then you will need to use the following steps.

1. Open the ***Package.appxmanifest*** file in the UWP project.
2. Click on the *Capabilities* tab.
3. In the *Capabilities* section, check the box next to ***Phone Call*** to allow you to use the dialer without UI intervention. You can actually use the feature without this capability, this simply lets you do it without a UI prompt.
4. Next, add a reference to the extension library "***Windows Mobile Extensions for the UWP***".
5. This is found in the ***References > Extensions*** area.

Dial the phone

Finally, let's invoke our phone dialer logic from the ***Call Button***.

1. If the user responds affirmatively, then look up the phone dialer using the ***DependencyService***. The service provides a static method named Get<T> that will locate and then provide access to a platform-specific implementation of an interface. In our case, we already registered our platform-specific implementations in a previous step using the ***DependencyAttribute***. Here we are using the *Get<T>* method to retrieve the implementation of the *IDialer* interface for our current platform.
2. Use the *IDialer* to dial the phone.

```
async void OnCall(object sender, System.EventArgs e)
{
    if (await this.DisplayAlert(
        "Dial a Number",
        "Would you like to call " + translatedNumber + "?",
        "Yes",
        "No"))
    {

        var dialer = DependencyService.Get<IDialer>();
        if (dialer != null) {
            await dialer.DialAsync(translatedNumber);
        }

    }
}
```

Test the Application

1. Run the application using the most convenient platform and go ahead and translate and then dial a number.

2. Some emulators and simulators will not properly simulate the dialer - Windows Phone and the Android SDK emulator are an exception to this.

3. You might get an exception from a simulator or emulator when trying to dial the phone - it depends on your configuration. If this happens, just comment out the call to dial the phone, or wrap it in a ***try / catch*** statement. If you have a real device, try running the application there.

4. If you have time, try each of the platforms out and try stepping through the code to see it jump into the platform specific code.