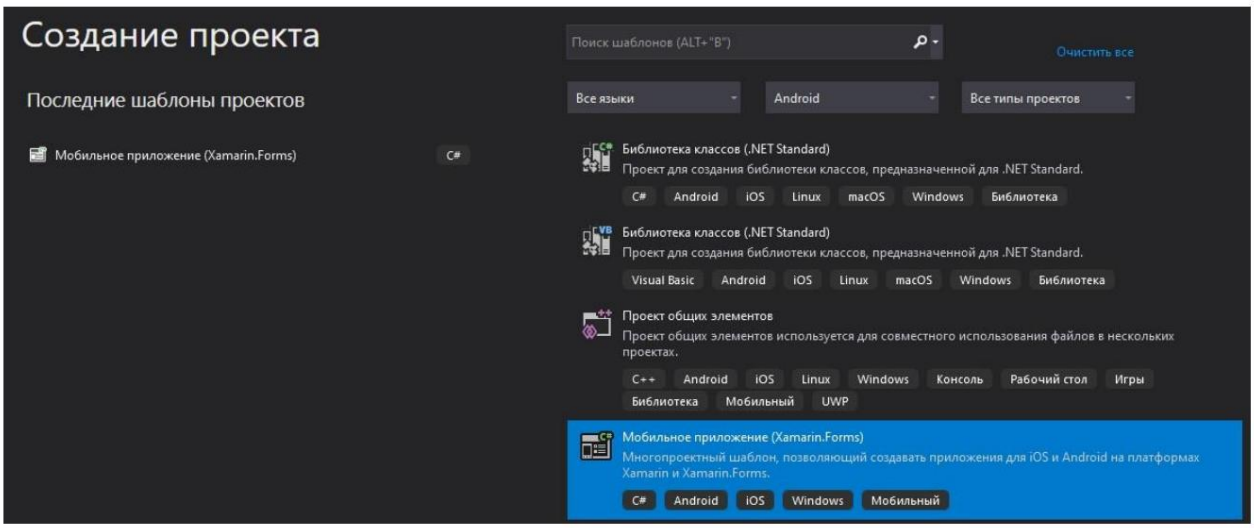
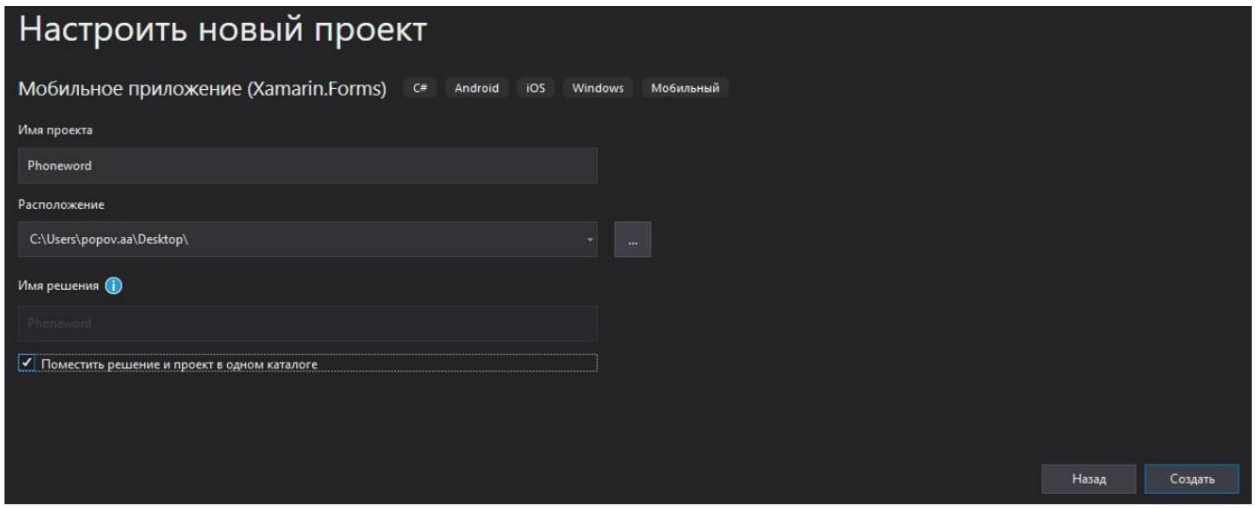
Creating a Project Using Xamarin

This lesson is dedicated to creating a trial project with using the Xamarin platform. To create a project, you need to run Visual Studio.Net. Next, you need to click **Create Project.**

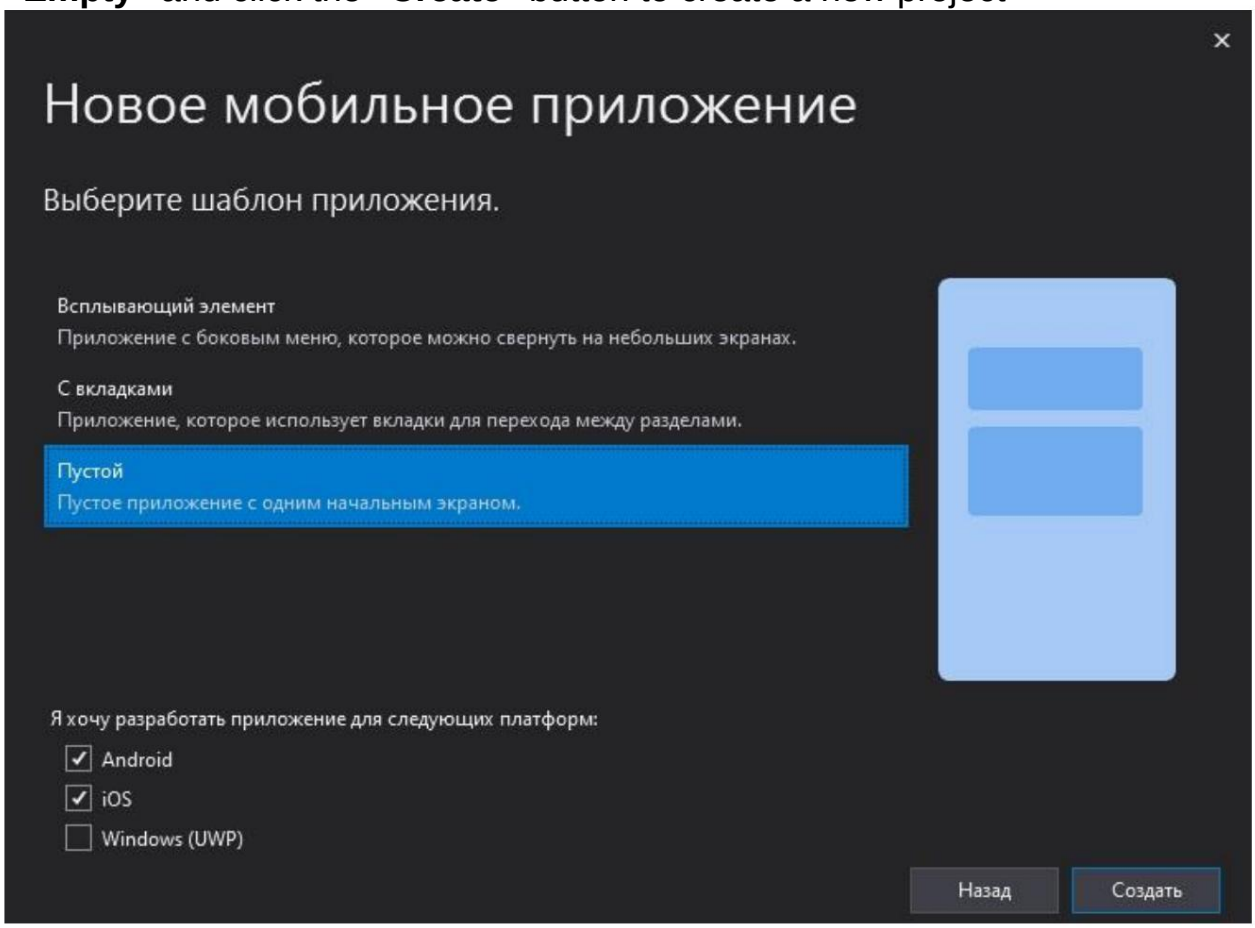
In the **Create a Project** dialog box, select **the Mobile Application (Xamarin.Forms)** template and click Next.



Set the project name to **Phoneword,** define the project location and click the **Create button.**

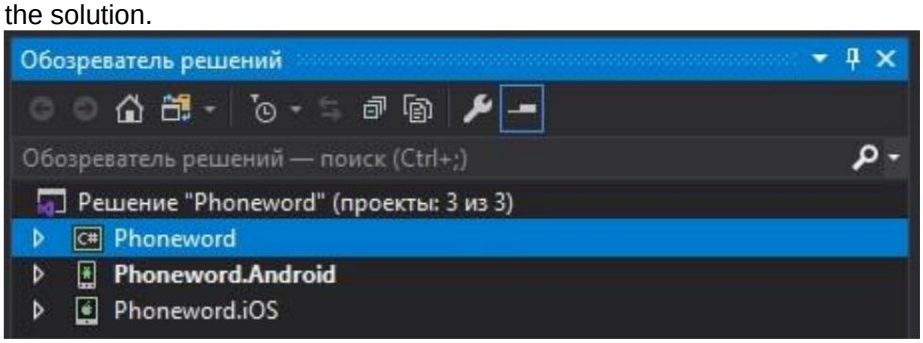


In the **New Mobile App** dialog box, click Template **"Empty"** and click the **"Create"** button to create a new project



After creating the project, go to **Solution Explorer.** We can see that there are three projects created in

the solution.



They all have the same root name, but two have a suffix (".iOS" or ".Android").

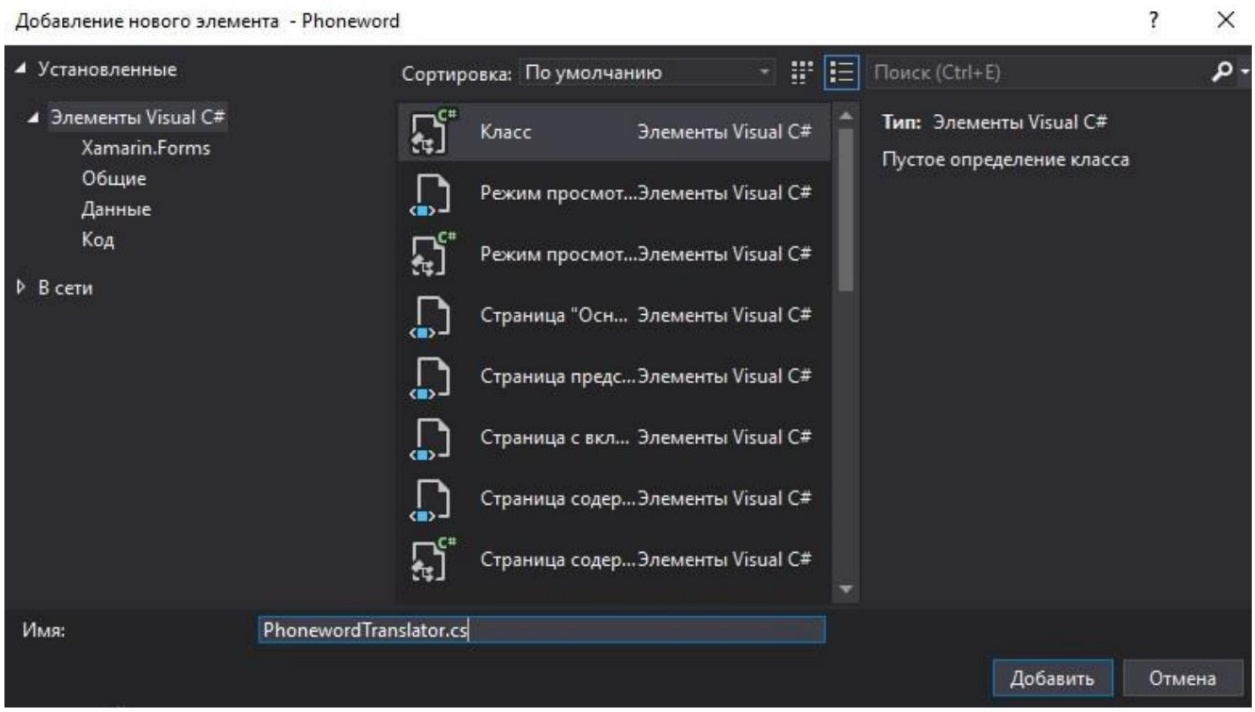
We will create a single-page software application that converts an alphanumeric string entered by the user into a numeric string that is displayed in the user interface.

As a basis for translation, the application will use letters that correspond to numbers displayed on the keypad of a push-button telephone.

For example, the string "CAB" should be converted to the string "222", since the number "2" on the keyboard corresponds to all three letters from the string "ABC". Let's add a new file with the C# program

code to the project. To do this, right-click on the project named Phoneword

and select **Add>Create Item** and in the dialog box that opens, **"Add New Item" ,** select the **"Class"** template , enter the class name PhonewordTranslator.cs and click the **"Add" button.**



An empty C class named PhonewordTranslator is created. Type the following program code inside the PhonewordTranslator class:

code:

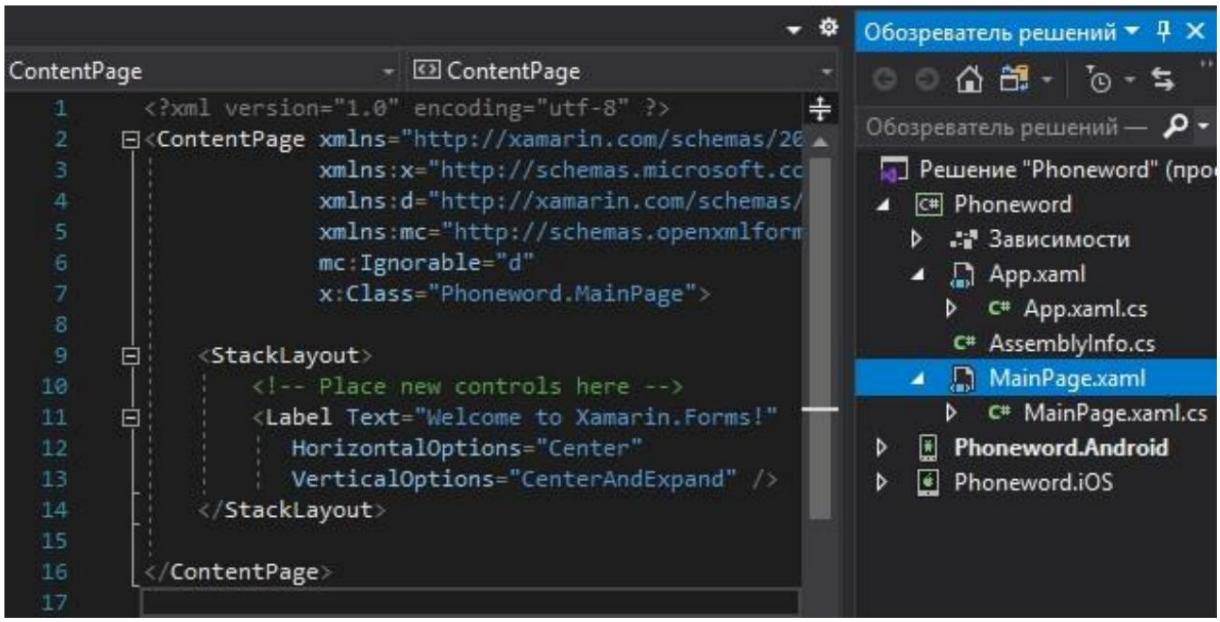
```
public static class PhonewordTranslator { public

static string ToNumber(string raw) {

        // Check if string is missing if
        (string.IsNullOrWhiteSpace(raw)) return null;

        // Convert string characters to uppercase raw =
        raw.ToUpperInvariant();

        var newNumber = new StringBuilder();

//Iterate over the characters of the raw
        string foreach (var c in raw)
        {
            // Checking the value of the flag // of the
            character's inclusion in the "numeric" string if ("
            -0123456789".Contains(c)) // If the
                character is in the string, then // the character is added to the
                string newNumber newNumber.Append(c); else { // If the
                character is not in the string,


            then // the function for translating a letter into a number is called var
                result = TranslateToNumber(c);
```

```
                // checking the result of executing //
                the TranslateToNumber function
               if (result != null) //
                   adding a digital character // to the
                   newNumber string instead of a letter
                   newNumber.Append(result);


                else // the result does not exist if the input // of the
                   function is a character that is not a digit // and does not correspond to
                   the digits array return null;

            }

        } return newNumber.ToString();
    }


    // The function searches for a character in a string
    and // returns the indicator of whether the character
    is included (not
    included) // in the string static bool Contains(this
    string keyString, char c) {
        return keyString.IndexOf(c) >= 0;
    }

    // Array for converting letters to numbers
    static readonly string[] digits = {
        "ABC", "DEF", "GHI", "JKL", "MNO", "PQRS", "TUV", "WXYZ" };


    // Function for converting letters to
    numbers static int?
    TranslateToNumber(char c) {
        for (int i = 0; i < digits.Length; i++)

        { if (digits[i].Contains(c))
              return 2 + i;

        } return null;
    }
}
```

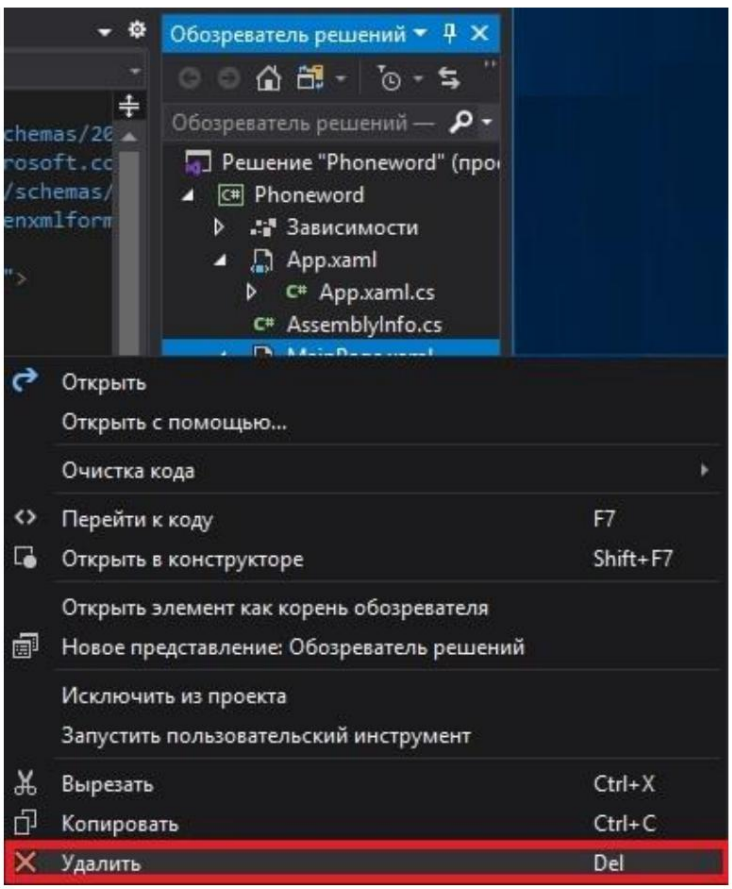
Save the entered program code.

Rework the program code located in the MainPage.xaml file and corresponding to the user interface.

Create your own user interface. To do this, you must first delete the automatically generated user interface code located in the MainPage.xaml file.

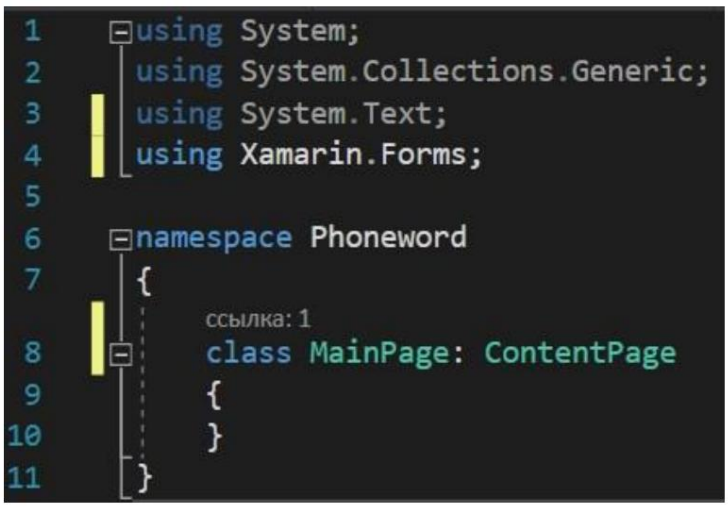


To do this, in Solution Explorer, right-click the MainPage.xaml file and select **Delete.** This action also deletes the MainPage.xaml.cs code file.



Create a class in a new file called MainPage.cs. You can use the same process described above to create a new class file.

Make the class derive from the ContentPage class. Add a using Xamarin.Forms statement because ContentPage is in the Xamarin.Forms namespace.



The next step is to add the program code to connect the user interface in the created MainPage.cs class. The program application uses the StackLayout layout. To work with it, you will need to create an instance of it and make sure that the orientation property is set, which we need to arrange the controls in the layout from top to bottom. If the layout orientation is Vertical, then each element

controls inside the layout fill the entire width of the layout. In order for the layout of a software application to look attractive, there must be some space between its elements. There are several properties that can be used to achieve this in

depending on the goals that the software application achieves.

Each control has a Margin property, which is used in layouts to set the distance between controls placed within the layout. All layouts have a Padding property, which keeps all controls within the layout at a certain distance from the border.

layout.

The page's Padding property is set to 20 for all margins. Another setting for setting the free space in the UI is the StackLayout's Spacing property. This property sets the space between all the children of the StackLayout. In the code, the StackLayout's Spacing property is set to 15. The property is in addition to the control's own Margin, so the actual free space between controls will be the combined value of the margin and the spacing (if both are applied).

Controls are added to the layout using the Children property. The following controls are created and added

to the layout in the code
controls for the user interface of the software application:

1. A Label control with a Text property set to the help text "Enter a Phoneword." 2. An Entry control that allows an alphanumeric string to be entered or edited and that collects input from the user. The Entry control's Text property is

initialized to "1-855-XAMARIN." The user can replace the initial text with their own, but prepopulating the control is necessary for testing the software application. 3. A Button control that runs the software logic to convert the alphanumeric string to a numeric string. Its Text property should be set to "Translate." 4. A second Button control displays the numeric string produced by the previously entered alphanumeric string. Its Text property should be set to "Call." Clickability on this Button control should be disabled by default by setting its IsEnabled property to false. After constructing the StackLayout with controls inside it,

you need to add the layout to the Content property of the ContentPage. In this case, the MainPage will display the contents of the StackLayout using the control layout rules set in the StackLayout class layout, as well as the values of the controls' properties and

layout.

Next, you need to connect the Clicked events of the Translate button to convert the alphabetic string into a numeric one. To do this, use the OnTranslate event

handler, which
takes object and EventArgs as parameters.

The event handler must convert the alphanumeric string entered in the Entry control to a numeric string using the PhonewordTranslator.ToNumber method of the previously developed PhonewordTranslator class.

The numeric string is stored in a string named translatedNumber. The alphanumeric string

is then converted to
The numeric string is obtained from the Text property of the Entry control.

The PhonewordTranslator.ToNumber method returns null if the alphanumeric string cannot be converted to a numeric string. Next, we'll add some code that changes the Text

property of the Call button so that it displays a numeric string if it's successfully converted from an alphanumeric string.

The Call button is available or not available to press depending on the success of converting alphanumeric construction into digital construction.

If the TranslateNumber function returns null, the Call button becomes unavailable, and if the conversion is successful, it becomes available. To make a button available or

unavailable for clicking

As mentioned earlier, the IsEnabled property is used.

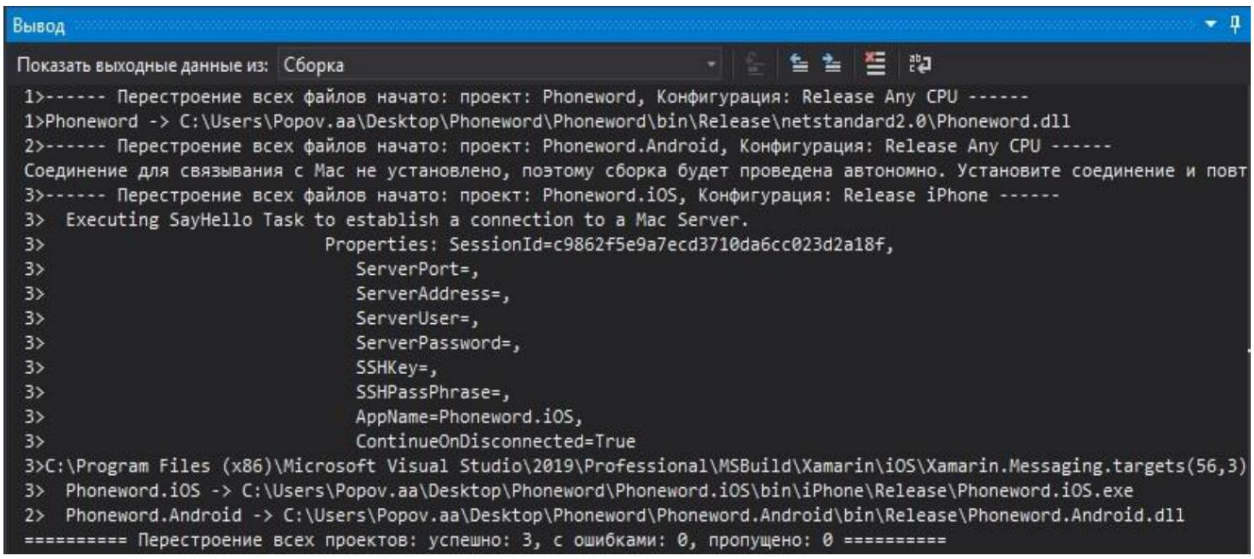
If the alphanumeric string cannot be converted (it contains characters that are not numeric or alphanumeric), the conversion fails and the Text property of the button (the Button2 control) is reset to "Call" and the message "Error String" appears in the Entry control's input field.

```
using System;
using System.Collections.Generic; using
System.Text; using
Xamarin.Forms;

namespace Phoneword {

    public class MainPage: ContentPage {

    Entry phoneNumberText;
    Button translateButton;
    Button callButton;
    string translatedNumber;

    public MainPage() {

        this.Padding = new Thickness(20, 20, 20, 20);

        // Setting layout properties
        StackLayout panel = new StackLayout {

            Spacing = 15,
            Orientation = StackOrientation.Vertical };



        // Add controls to the StackLayout 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, });


        // Handling clicking on the Translate button
        translateButton.Clicked += OnTranslate;

        // Adding a layout to the ContentPage this.Content = panel;

    }

    private void OnTranslate(object sender, EventArgs e) { string

    enteredNumber = phoneNumberText.Text; translatedNumber =

            Phoneword.PhonewordTranslator.ToNumber(enteredNumber);

        if (!string.IsNullOrEmpty(translatedNumber)) {

            callButton.IsEnabled = true;
            callButton.Text = "Call " + translatedNumber;

        }
        else {
            callButton.IsEnabled = false;
            callButton.Text = "Call";
            phoneNumberText.Text = "Error string";
        }
    }

  }
}
```

Next, you need to build the solution. To do this, click the Build menu and select **Build solution from the drop-down menu.** If the solution has already been built and changes have been made to the program code, you need to select **Rebuild solution.**

Make sure the Output dialog box is error-free

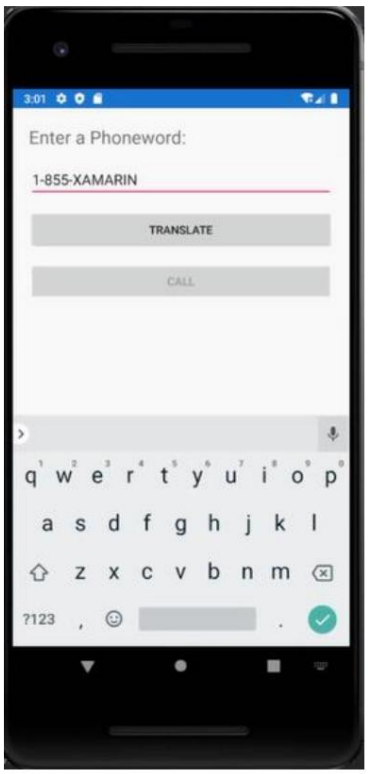


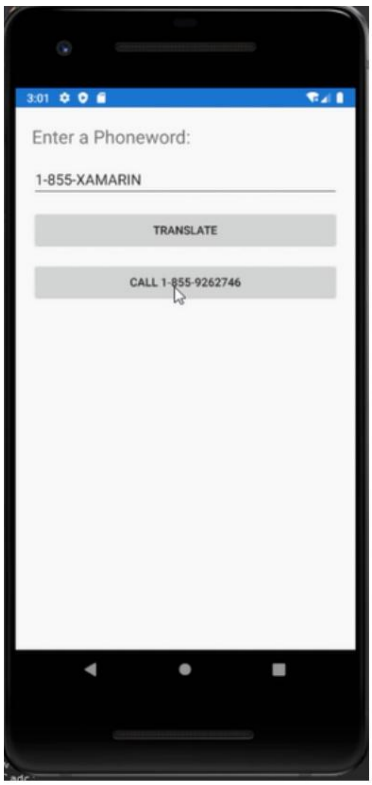
Watch a video showing the software in action
applications can be launched by running an existing video file.

You need to test the software application by running it on a previously configured Android device or emulator. Next, you need
to click on the "TRANSLATE" button to convert
alphanumeric string "1-855-XAMARIN" into a numeric string.

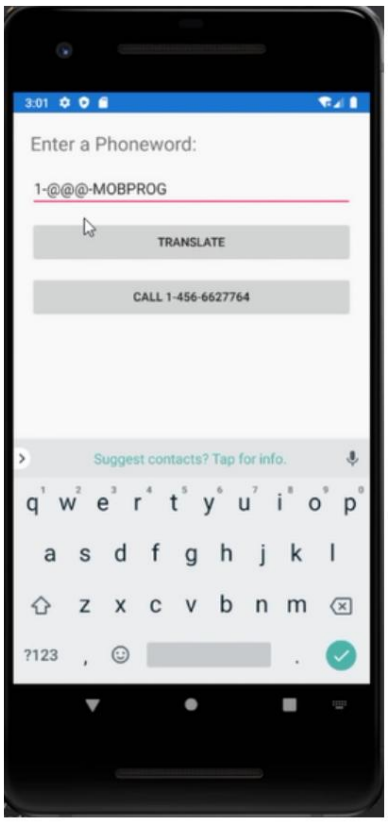
The button with the text "Call" displays a text string consisting of "Call" and the resulting digital string.

The lower button becomes available for pressing





Now, in the Entry control, you must enter an "incorrect" alphanumeric string "1-@@@-MOBPROG" containing three characters that are neither letters nor numbers.



Next, you need to click on the button with the text “TRANSLATE” to convert the alphanumeric string “1-@@@-MOBPROG” into

numeric string. In this case, the conversion of the "incorrect" alphanumeric string will fail, and the value of the Text property
the lower one is assigned the value "Call", it becomes unavailable for pressing, and the line “Error String” appears in the Entry input field (Fig.).