Visual Studio を用いたクロスプラットフォーム開発入門 Android 編

演習の目標

- Xamarin と Visual Studio を使用して Android アプリケーションを Xamarin ネイティブの手法、Xamarin.Forms の手法を用いて開発する方法について学習します。

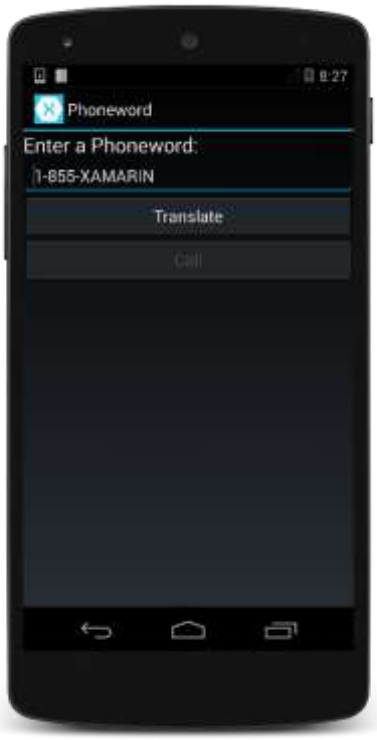
演習の概要

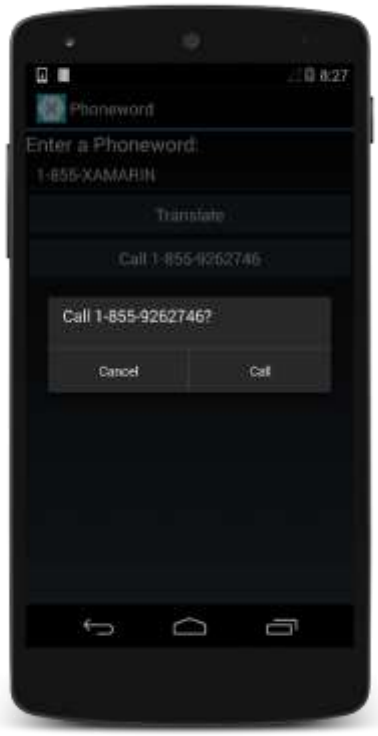
- 開発に必要な Android SDK、Java、Xamarin の設定を確認
- Xamarin.Android プロジェクトの作成、開発
- Android Emulator または実機へのビルド、デプロイ
- Android Emulator または実機でのデバッグ
- Xamarin.Forms (Portable) プロジェクトの作成、開発
- Android、Windows 10 Mobile へのビルド、デプロイ、デバッグ

予想所要時間

Xamarin ネイティブの手法、Xamarin.Forms の手法合わせて 120 分

本ガイドでは、Visual Studio を使用した Xamarin.Android アプリケーションの開発の基本的な部分を紹介します。Xamarin.Android アプリケーションのビルドと配布に必要なツール、コンセプト、ステップも紹介します。本ガイドでは、ユーザーが入力した英数字の電話番号を数字の電話番号に変換し、その番号に電話するアプリケーションを作成します。完成したアプリケーションは、以下のような画面になります。





1　開発に必要な Visual Studio、Android SDK、Java、Xamarin の設定を確認

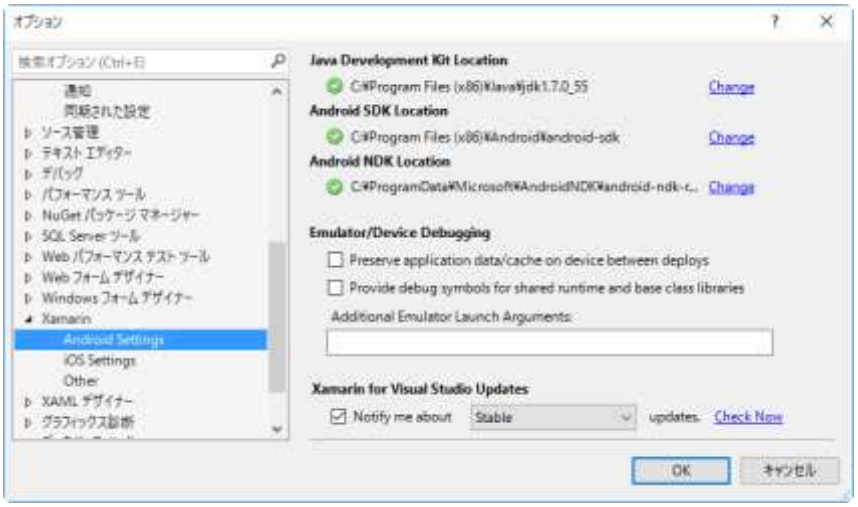
1.1　Visual Studio

本ガイドでは Visual Studio 2015 Community を使用します。Professional 以上のエディションをお持ちの方はそちらをご利用ください。

<<Visual Studio バージョン情報>>

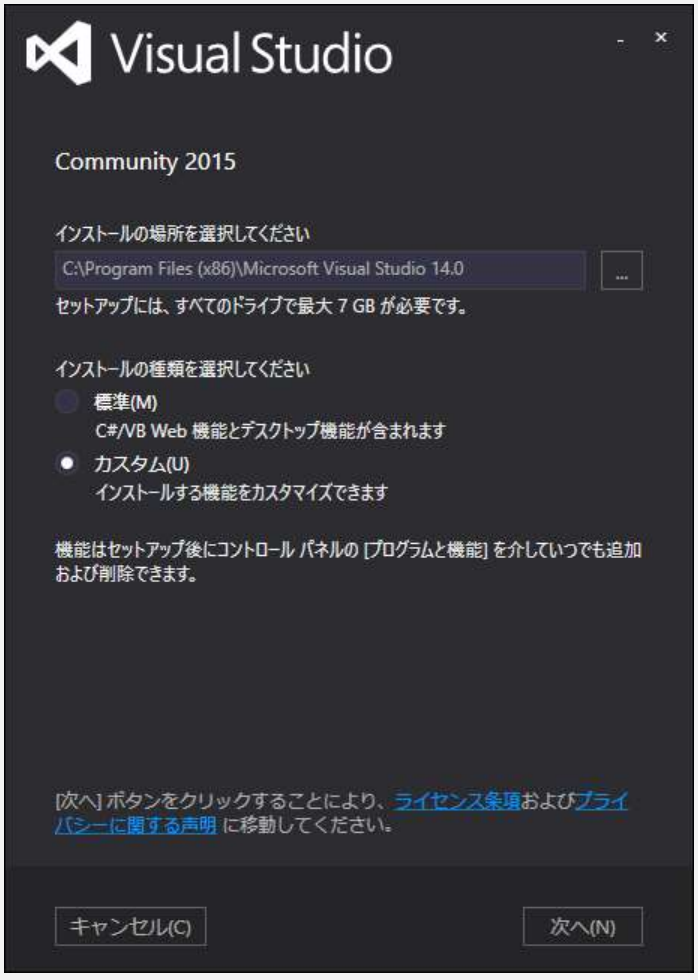
1.2　Android SDK

[ツール ＞ オプション ＞ Xamarin ＞ Android Setting]を選択し、[Java Development Kit Location]と[Android SDK Location]のパスを確認します。

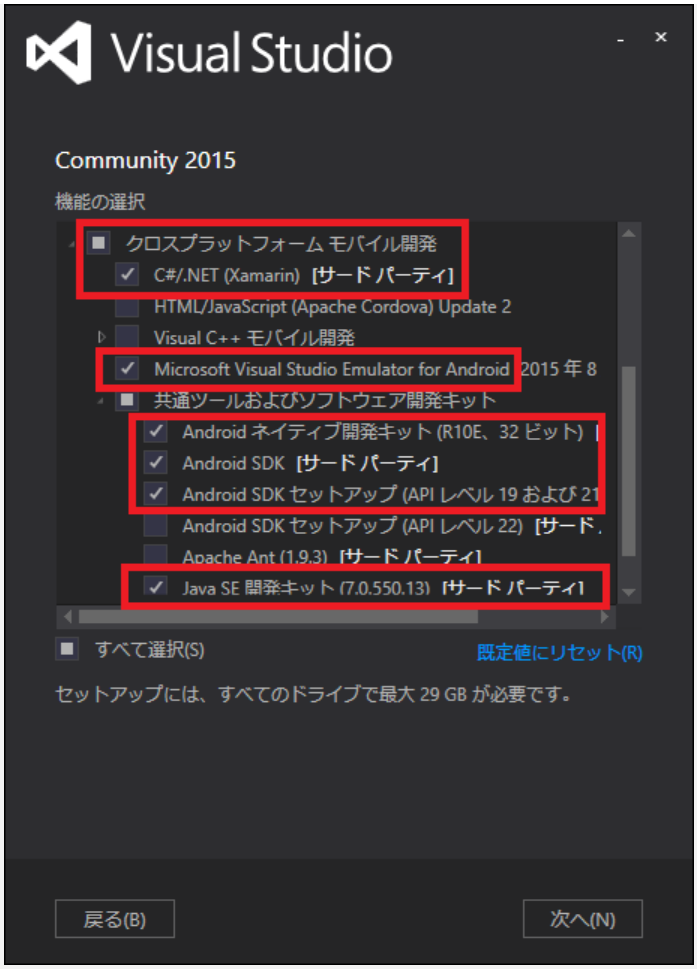


<<Xamarin.Android オプション>>

なお、Xamarin は Visual Studio 2015 インストール時に[カスタム > モバイル開発]をチェックすることで自動的にインストールされます。



<<Visual Studio 2015 インストーラー>>



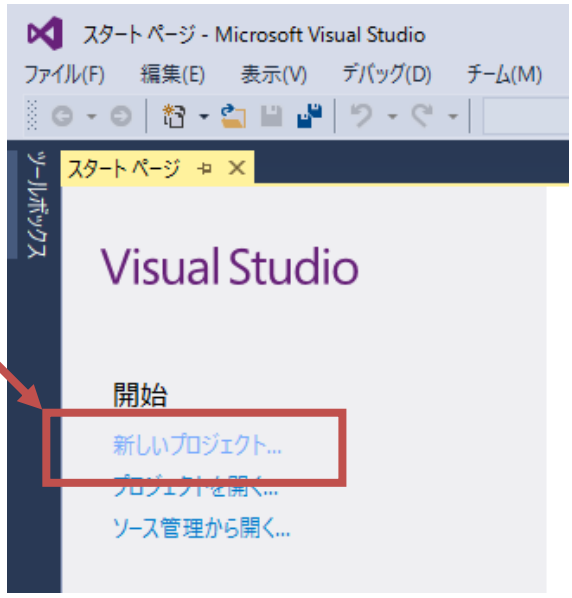
<<Visual Studio 2015 インストーラー>>

1.3　エミュレーターの準備

Android のエミュレーターには、いくつかオプションがあります。スタンダードな Android エミュレーターは、シンプルな設定ですが、起動が遅くパフォーマンスが低いため、本ガイドでは、Visual Studio Emulator for Android、Xamarin Android Player、実機を使用してください。これらをお持ちでない場合は Android SDK に付属するハードウェア アクセラレーション（Intel HAXM）を使用した x86 ベースのエミュレーターを使用してください。ハードウェア アクセラレーションの設定方法に関しては、Accelerating Android Emulators ガイド（ http://developer.xamarin.com/guides/android/getting_started/installation/accelerating_android_emulators ）をご参照ください。

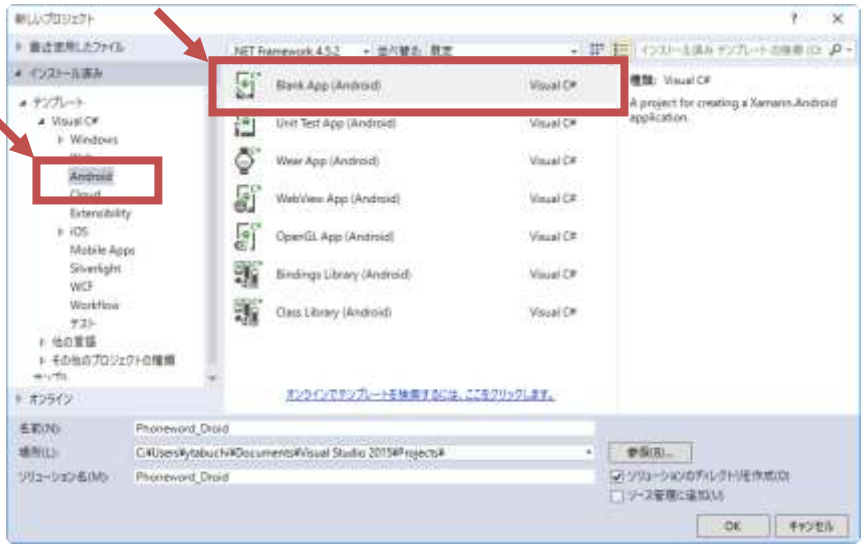
2　Xamarin.Android プロジェクトの作成

2.1　Visual Studio を起動し、[スタートページ ＞ 新しいプロジェクト]をクリックして、新しいソリューションを作成します。



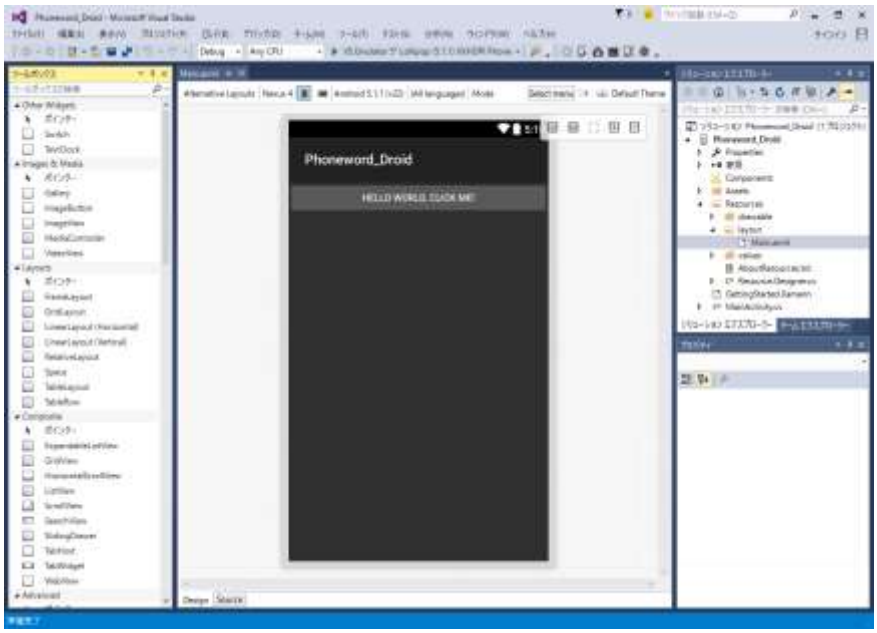
<<Visual Studio メイン画面>>

2.2　[新しいプロジェクト]画面で、[Visual C# ＞ Android]をクリックします。[Blank App (Android)]テンプレートを選択します。新しいソリューションには、名前を「Phoneword_Droid」と付けます。



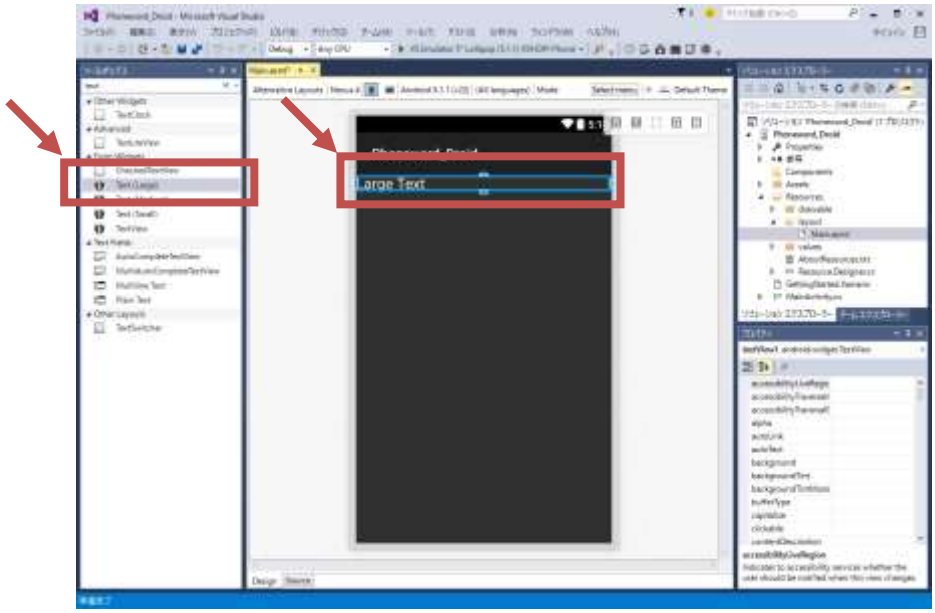
<<ソリューション追加>>

2.3　次に、ソリューションエクスプローラーから[Phoneword_Droid]プロジェクト内の[Resources]フォルダを開き、[layout]以下の[Main.axml]をダブルクリックで開きます。Android Designer が起動します。



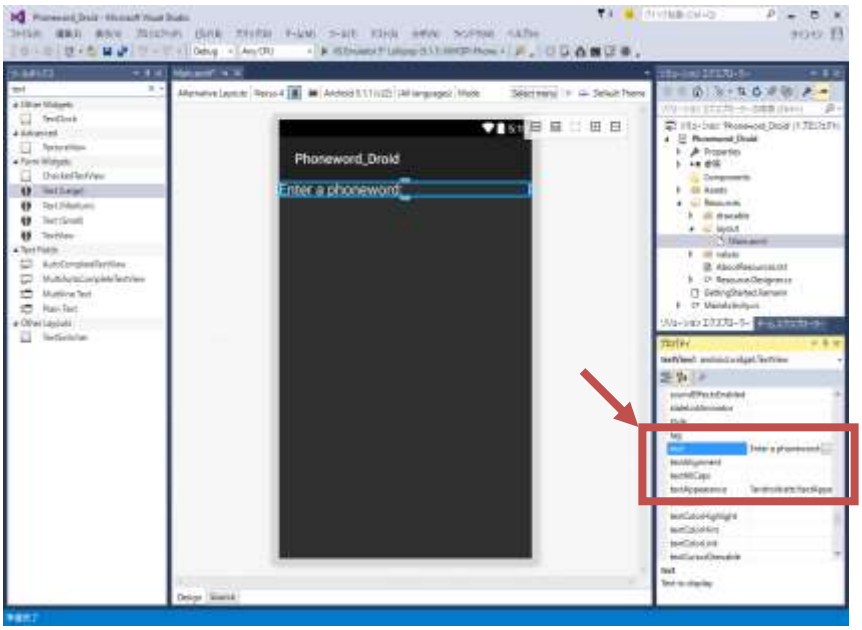
<<Android Designer>>

2.4　デザイン画面に標準で配置されている[HELLO WORLD, CLICK ME!]と書いてある[Button]を選択し、キーボードの[Delete]キーで削除します。画面左側の[ツールボックス]の上部にある検索欄に「text」と入力し、[Text (Large)]を選択し、デザイン画面にドラッグ＆ドロップして配置します。



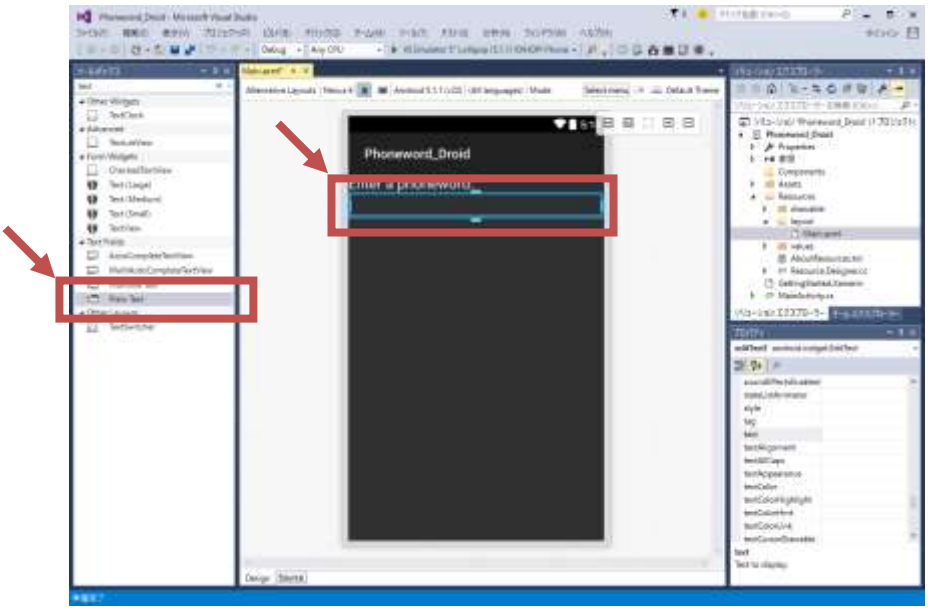
<<Text (Large) をドラッグ>>

2.5　デザイン画面に配置した[Text (Large)]コントロールを選択し、右下の[プロパティ]ウィンドウを使用して[Text (Large)]の[Text]プロパティを「Enter a Phoneword」に変更します。



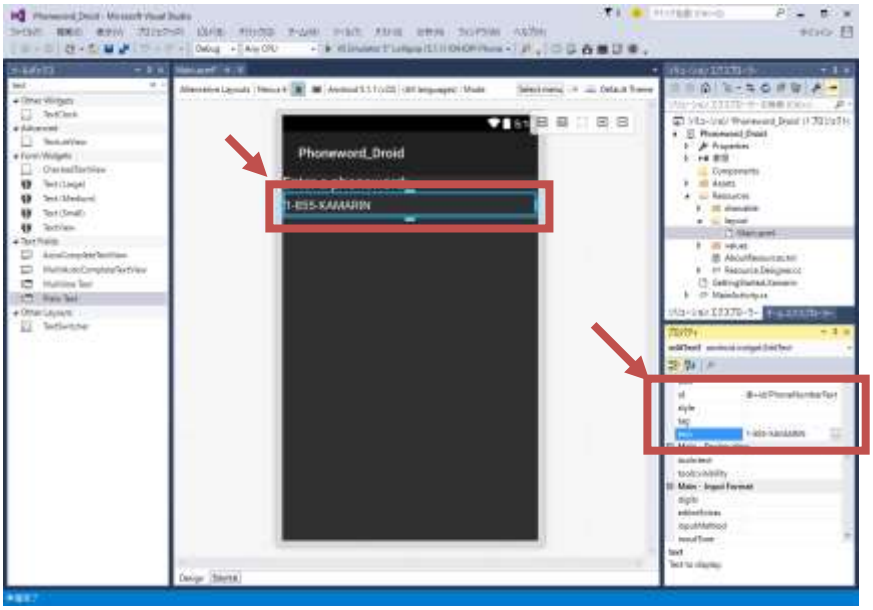
<<プロパティ変更>>

2.6　次に、[ツールボックス]から同様に[Text Field]内の[Plain Text]をドラッグして、デザイン画面の[Text (Large)]コントロールの下に配置します。



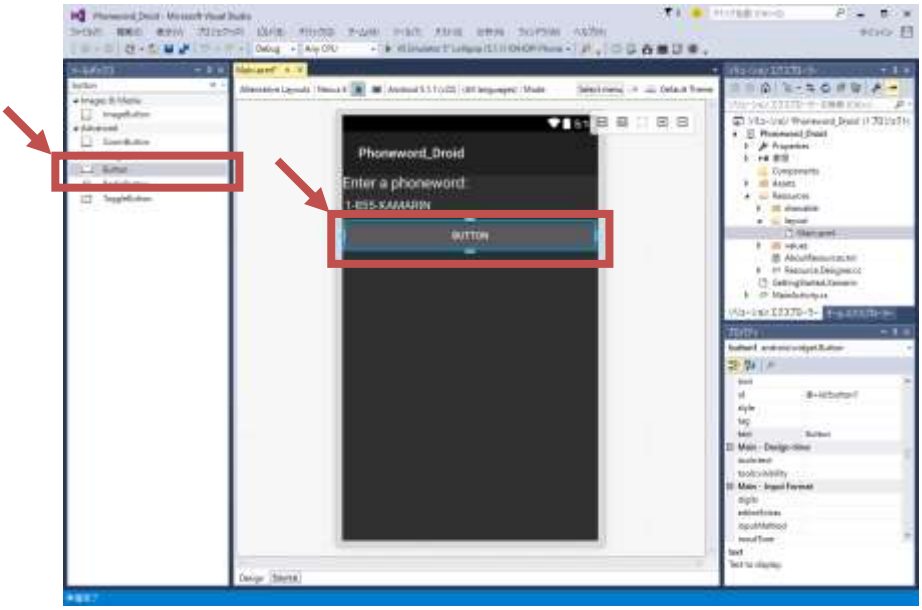
<< Plain Text 配置 >>

2.7　デザイン画面で配置した[Text Field]を選択し、[Text Field]コントロールの[Id]プロパティを「@+id/PhoneNumberText」に、[Text]を「1-855-XAMARIN」にそれぞれ変更します。



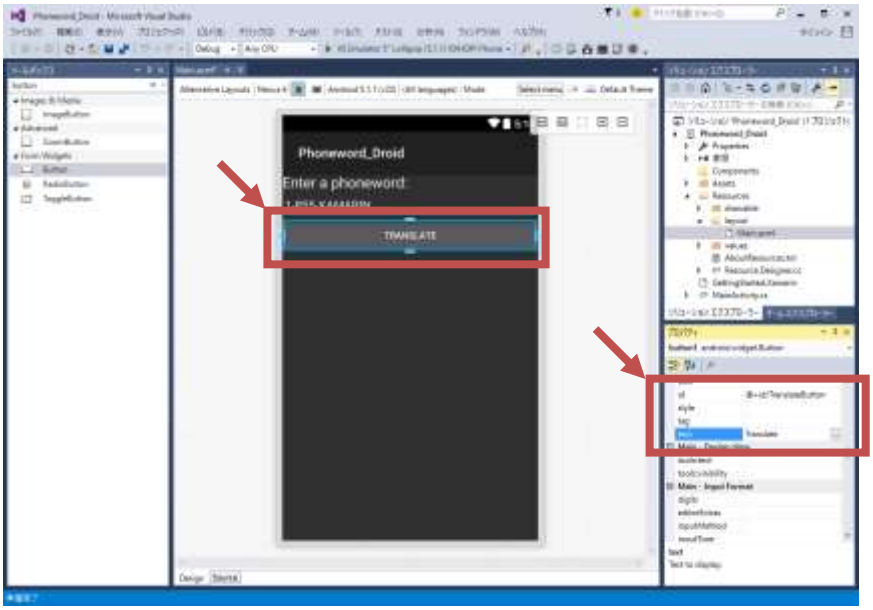
<< プロパティ変更 >>

2.8　ツールボックスで先ほど検索した[text]を削除し[button]で再度検索します。[Button]コントロールをデザイン画面にドラッグして、[Text Field]コントロールの下に配置します。



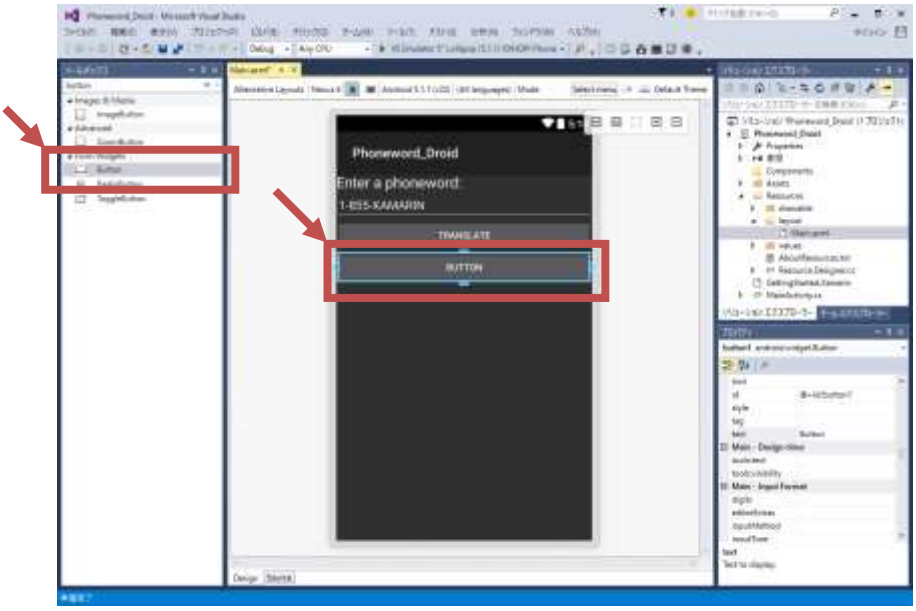
<< Button 配置 >>

2.9　デザイン画面で[Button]を選択し、[プロパティ]ウィンドウを使用して[Button]の[Id]プロパティを「@+id/TranslateButton」に、[Text]プロパティを「Translate」にそれぞれ変更します。



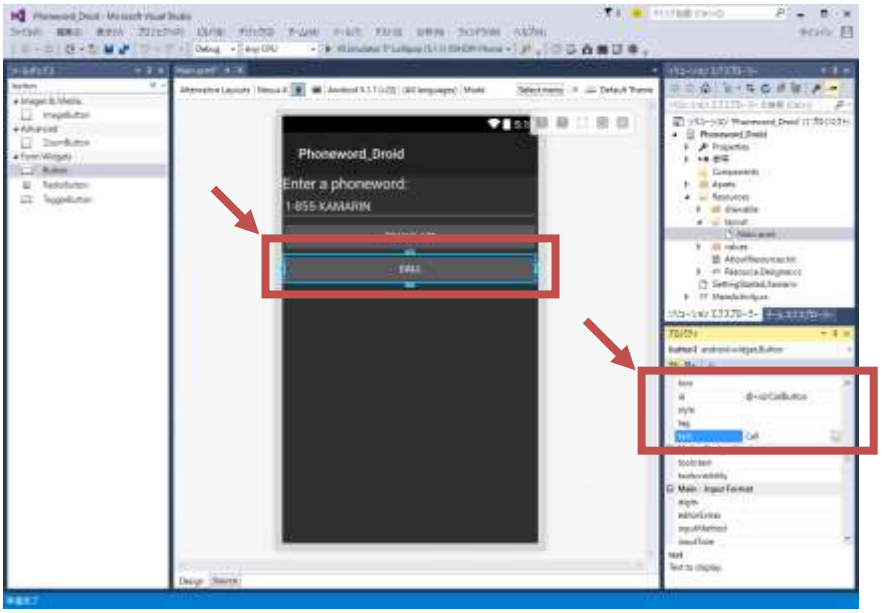
<< プロパティ変更 >>

2.10　次に、[ツールボックス]から 2 つ目の[Button]コントロールをドラッグして、デザイン画面の[TranslateButton]コントロールの下に配置します。



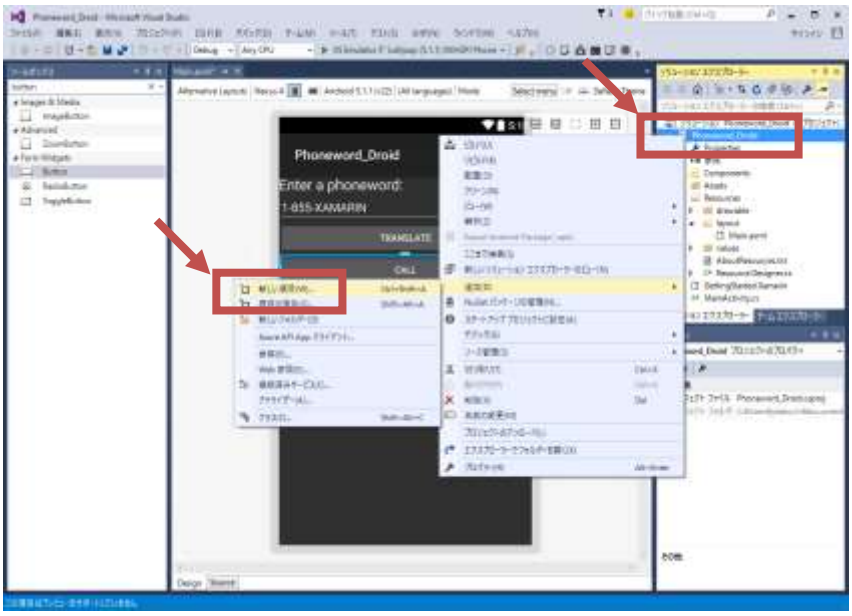
<< Button 配置 >>

2.11　デザイン画面の[Button]を選択し、[プロパティ]ウィンドウを使用して、[Button]の[Id]プロパティを「@+id/CallButton」に、[Text]プロパティを「Call」にそれぞれ変更します。変更したら[Ctrl＋S]を押して保存します。



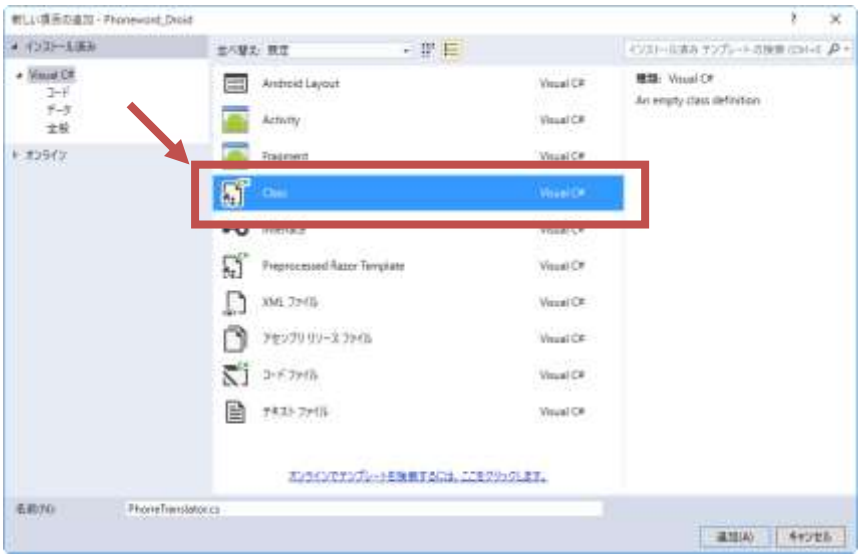
<< プロパティ変更 >>

2.12　ここで、英数字から数字に電話番号を変換するコードを追加します。ソリューションエクスプローラーから[Phoneword_Droid]プロジェクトを右クリックし、[追加 > 新しい項目]を選択します。



<< 新しい項目を追加 >>

2.13　[新しい項目の追加]ダイアログで、[Visual C# > Class]を選択し、新しいクラスの名前を「PhoneTranslator.cs」と付けます。



<< クラスを追加 >>

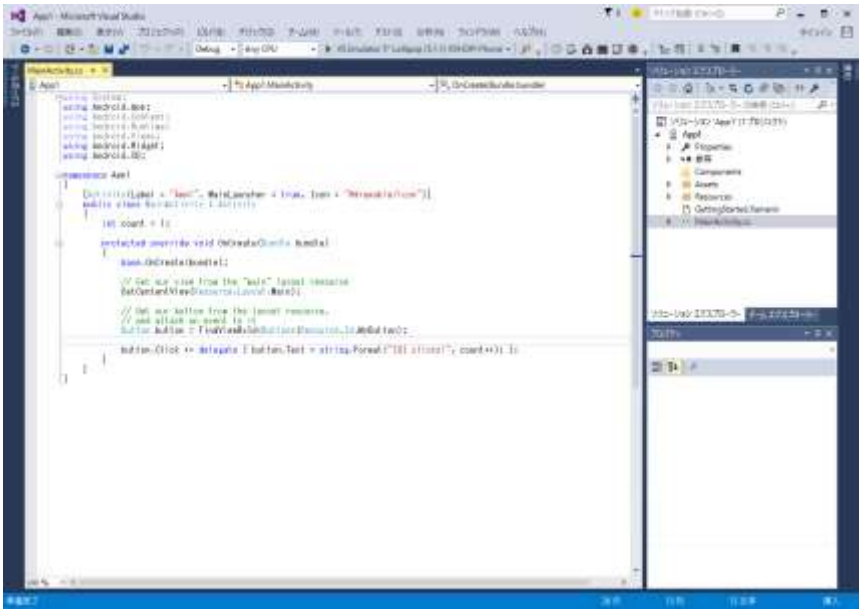
2.14　すべてのテンプレートのコードを削除し、以下のコードに置き換えます。

```
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);
                }
                // 数字以外の文字はスキップします。
            }
            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;
        }
    }
}
```

[ファイル ＞ 保存]を選択するか、または[Ctrl＋S]を押して、[PhoneTranslator.cs]ファイルを保存して、閉じます。

2.15　次に、ユーザーインターフェースを操作する[MainActivity]クラスにコードを追加します。ソリューションエクスプローラーから[MainActivity.cs]をダブルクリックして開きます。



<< MainActivity >>

2.16　[TranslateButton]を操作するコードを追加します。追加する場所は[MainActivity]クラスの[OnCreate]メソッドの中にある[base.OnCreate(bundle)]と[SetContentView (Resource.Layout.Main)]の下です。必要なものだけを残してテンプレートのコードを削除しておきましょう。[MainActivity]クラスは以下のコードのようになります。この時点で一度プロジェクトをビルドしておきます。

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

namespace Phoneword_Droid
{
 [Activity (Label = "Phoneword_Droid", 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);


        }
    }
}
```

2.17　次に、先ほど[Android Designer]で作成したコントロールを参照します。下記のコードを[OnCreate]メソッドの最後に追加します。

```
// ロードされたレイアウトから UI コントロールを取得します。
EditText phoneNumberText = FindViewById<EditText>(Resource.Id.PhoneNumberText);
Button translateButton = FindViewById<Button>(Resource.Id.TranslateButton);
Button callButton = FindViewById<Button>(Resource.Id.CallButton);
```

2.18　[TranslateButton]と名付けた最初のボタンをユーザーが押した際に応答するコードを追加します。下記のコードを OnCreate メソッドの中のコントロールの定義の下に追加します。

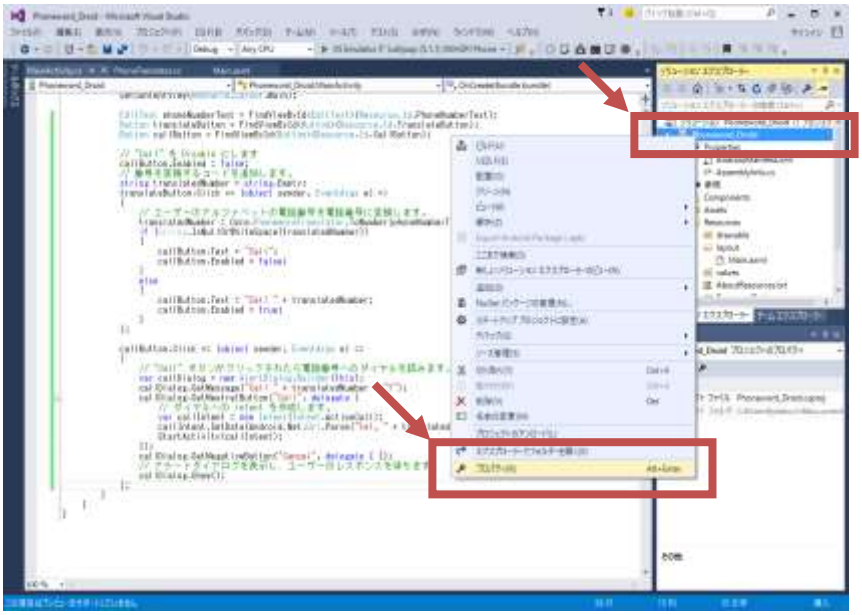
```
// "Call" を Disable にします
callButton.Enabled = false;
// 番号を変換するコードを追加します。
string translatedNumber = string.Empty;
```

```
translateButton.Click += (object sender, EventArgs e) =>
{
    // ユーザーのアルファベットの電話番号を電話番号に変換します。
    translatedNumber = Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
    if (String.IsNullOrWhiteSpace(translatedNumber))
    {
        callButton.Text = "Call";
        callButton.Enabled = false;
    }
    else
    {
        callButton.Text = "Call " + translatedNumber;
        callButton.Enabled = true;
    }
};
```

2.19　[CallButton]と名付けた 2 つ目のボタンをユーザーが押した際に応答するコードを追加します。以下のコードを[TranslateButton]のコードの下に追加します。

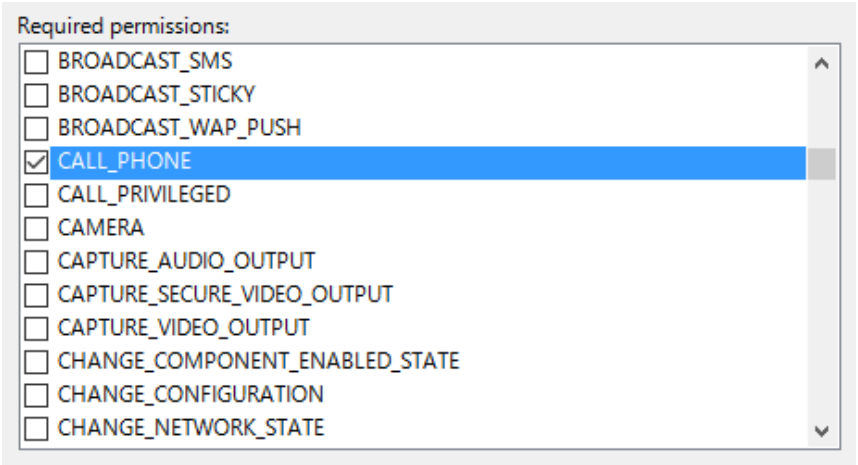
```
callButton.Click += (object sender, EventArgs e) =>
{
    // "Call" ボタンがクリックされたら電話番号へのダイヤルを試みます。
    var callDialog = new AlertDialog.Builder(this);
    callDialog.SetMessage("Call " + translatedNumber + "?");
    callDialog.SetNeutralButton("Call", delegate {
          // 電話への intent を作成します。
          var callIntent = new Intent(Intent.ActionCall);
          callIntent.SetData(Android.Net.Uri.Parse("tel:" + translatedNumber));
          StartActivity(callIntent);
       });
    callDialog.SetNegativeButton("Cancel", delegate { });
    // アラートダイアログを表示し、ユーザーのレスポンスを待ちます。
    callDialog.Show();
};
```

2.20　最後にアプリケーションの権限を設定して、電話をかけられるようにします。アプリケーション権限は、[Android Manifest]ファイルに記録されますが、Xamarin.Android の場合はプロジェクトのプロパティから変更できます。プロジェクトを右クリックして、[プロパティ]をクリックします。



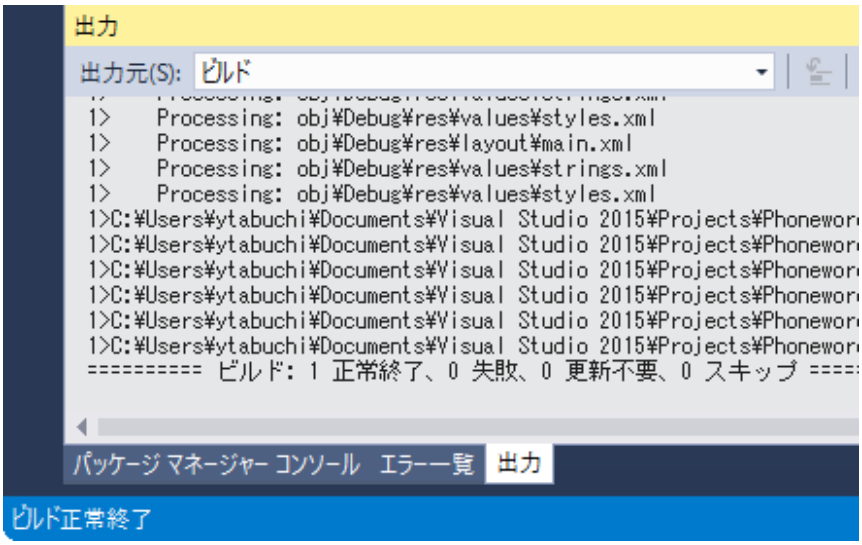
<< プロジェクトのプロパティ >>

2.21　[Android Manifest > Required Permissions]にある CALL_PHONE の権限をチェックします。



<< 権限設定 >>

2.22　これまでの作業を保存し、[ビルド ＞ ソリューションのビルド]を選択、または[CTRL-SHIFT-B]でアプリケーションをビルドします。 アプリケーションをコンパイルすると、Visual Studio の左下に[ビルド正常終了]と表示されます。



<< ビルド成功 >>

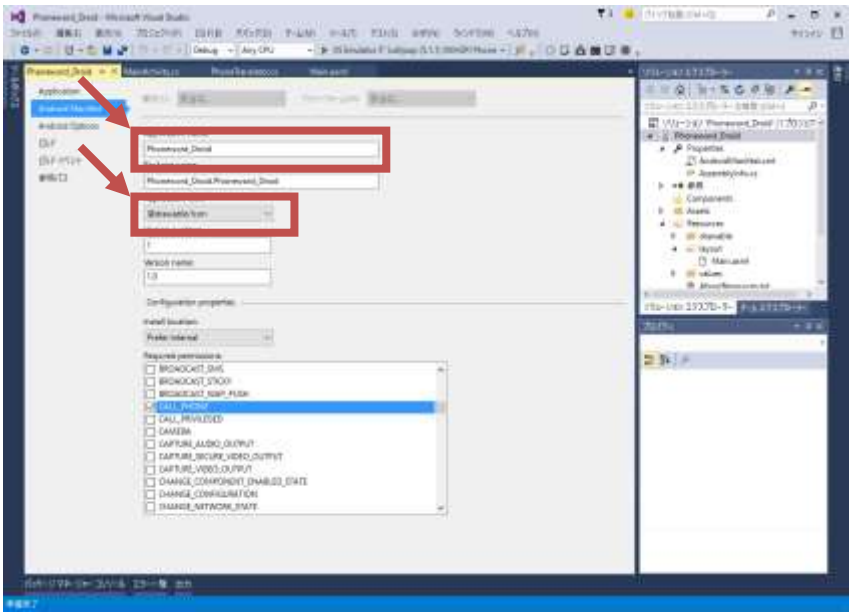
エラーが発生する場合、前のステップに戻って、アプリケーションのビルドが成功するまで、不正な箇所を修正します。

2.23　これで、アプリケーションが動作したので、最後の仕上げを加えていきましょう。[MainActivity]の[Label]を編集します。[Label]は、Android のアプリ一覧画面でアプリケーションがどこにあるかユーザーに知らせます。

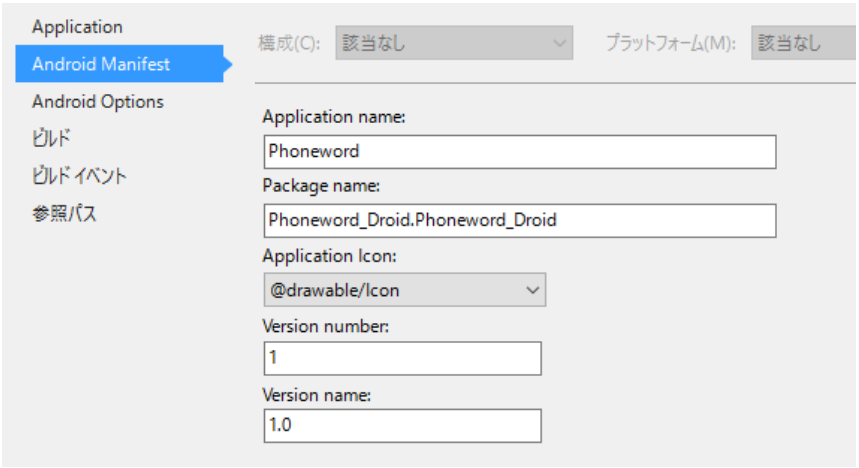
[MainActivity]クラスの上位にある[Label]を[Phoneword]に変更します。

```
namespace Phoneword_Droid
{
 [Activity (Label = "Phoneword", MainLauncher = true)]
    public class MainActivity : Activity
    {
        ...
    }
}
```

2.24　プロジェクトのプロパティからアプリケーションの名前とアイコンを編集することができます。[Android Manifest > Application name]の[Phoneword_Droid]を[Phoneword]に変更します。

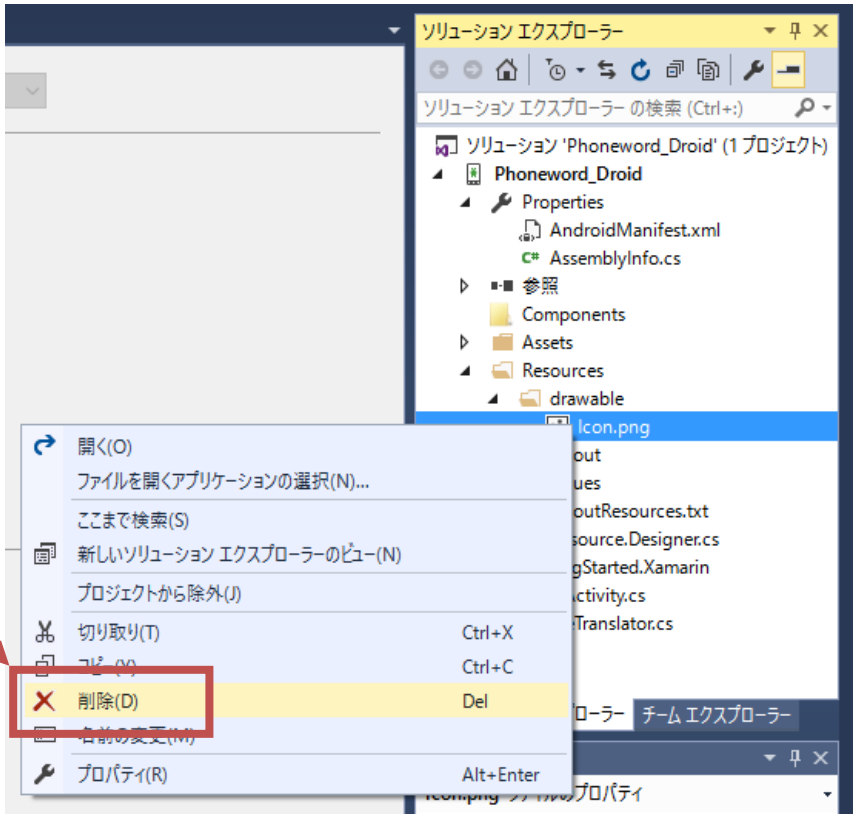


<< プロジェクトプロパティ >>



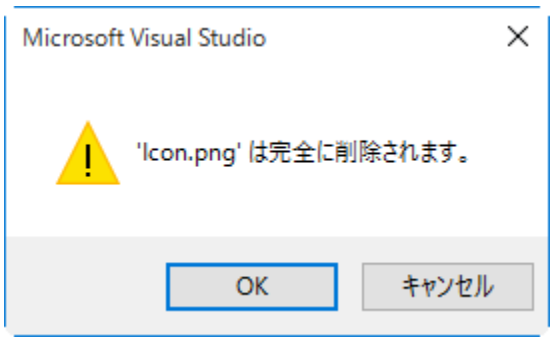
<< アプリケーションの名前を Phoneword に変更 >>

2.25　次に、アプリケーションのアイコンを編集します。まずアイコンセット（ https://github.com/ytabuchi/XamarinHOL/blob/master/XamarinAppIcons.zip ）をダウンロードして解凍します。次に、[Resources]フォルダ以下の[drawable]フォルダを開き、既存の[Icon.png]を右クリックし、[削除]を選択して削除します。



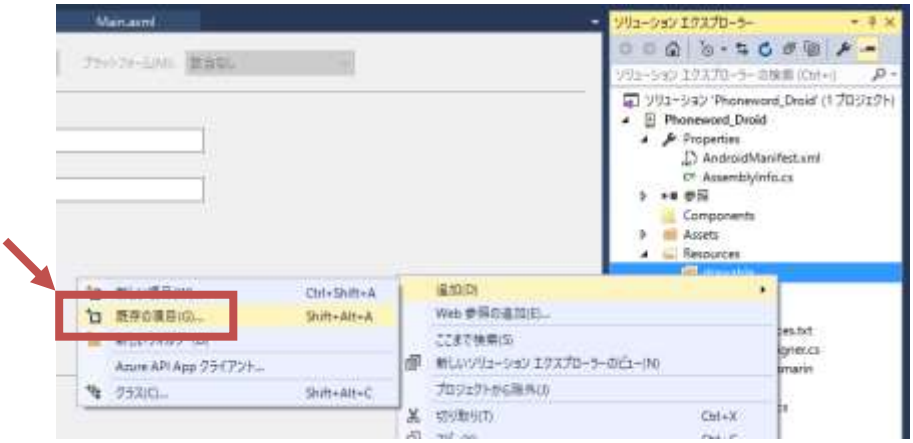
<< Icon.png を削除 >>

[drawable]フォルダが無い場合は、[Resources]フォルダを右クリックし、[追加 > 新しいフォルダ]を選択し、drawable フォルダを作成します。
ダイアログが表示された時は、[OK]を選択してください。



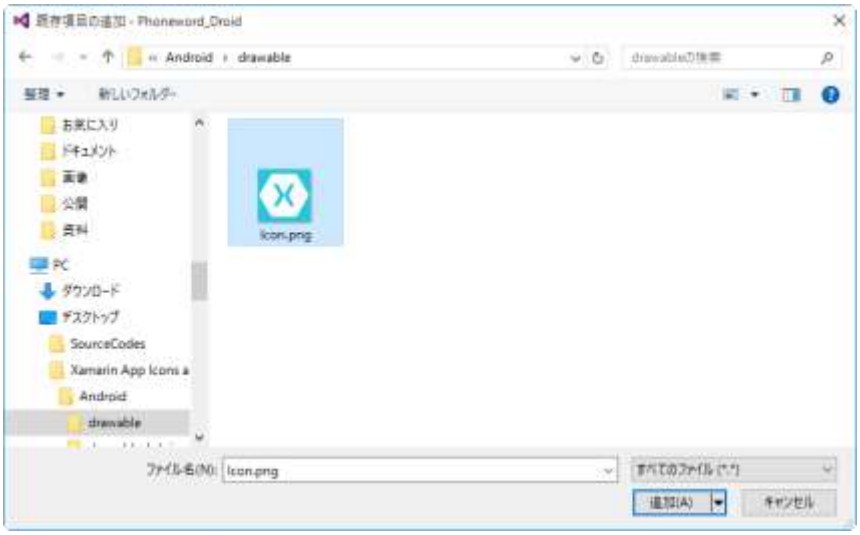
<< ファイル削除ダイアログ >>

2.26　[drawable]フォルダを右クリックして、[追加 ＞ 既存の項目]を選択します。



<< 既存の項目を追加 >>

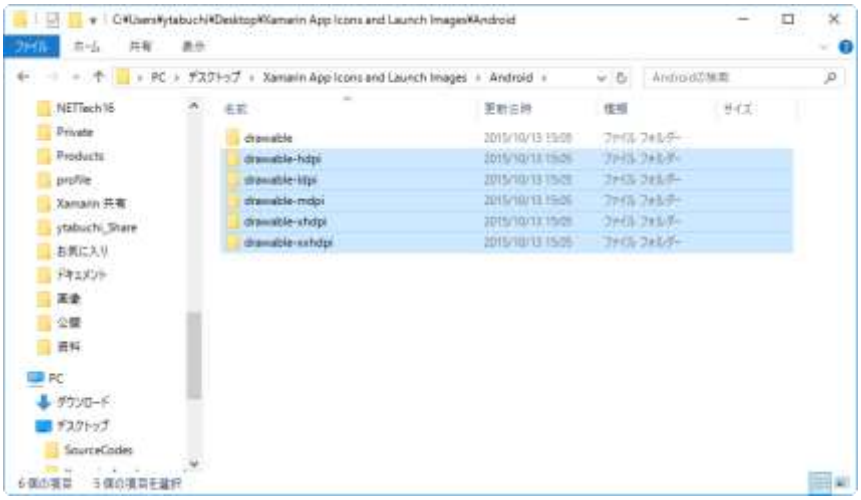
2.27　選択ダイアログでは、[XamarinAppIcons.zip]を解凍したフォルダに移動し、[drawable]フォルダを開きます。追加する[Icon.png]ファイルを選択し、[追加]ボタンをクリックします。



<< Icon.png を選択 >>

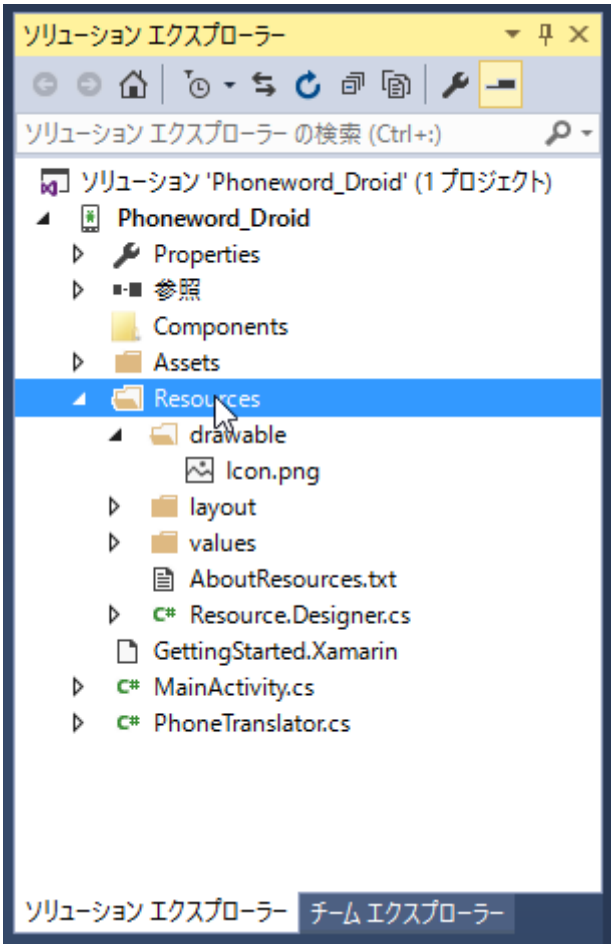
2.28　次に[Xamarin App Icons and Launch Images¥Android]ディレクトリにあるその他の[drawable-*]フォルダをプロジェクトに追加します。これにより、他の解像度のデバイスで表示された場合のアイコンの見た目が良くなります。

エクスプローラーから、drawable-* フォルダを選択します。



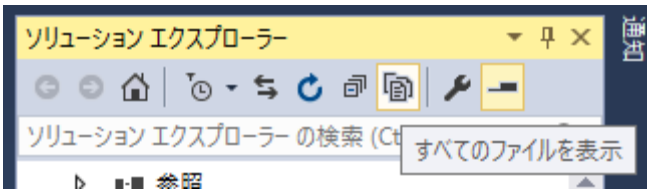
<< フォルダを選択 >>

選択したフォルダを Visual Studio のソリューションエクスプローラーにある[Resources]の中にドラッグします。

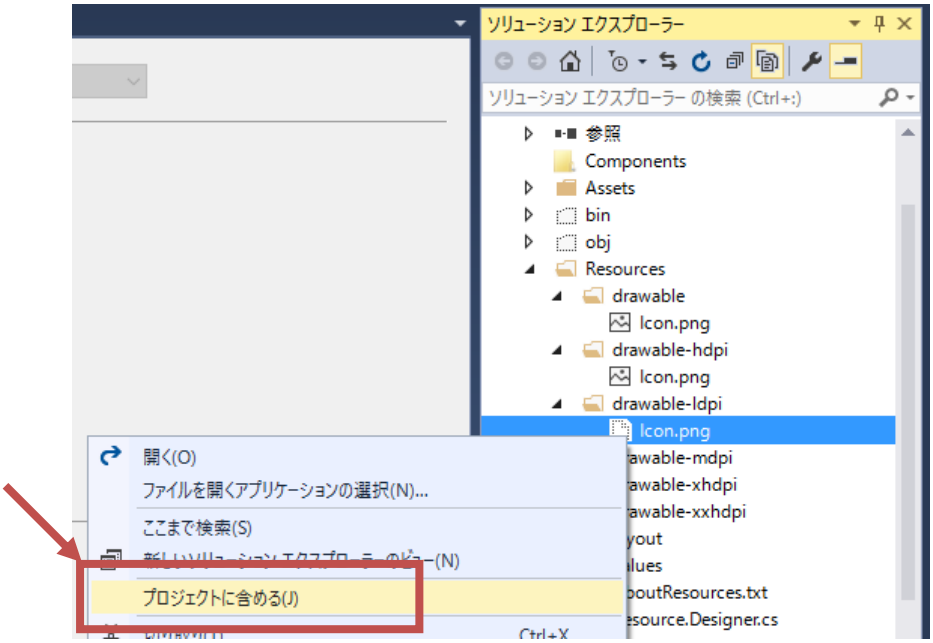


<< ドラッグして追加 >>

ソリューションエクスプローラーの中のプロジェクトに選択したフォルダが反映されます。
アイコンが追加されない場合は、ソリューションエクスプローラーの[すべてのファイルを表示]ボタンをクリックして、[Icon.png]を右クリックから[プロジェクトに含める]をクリックします。

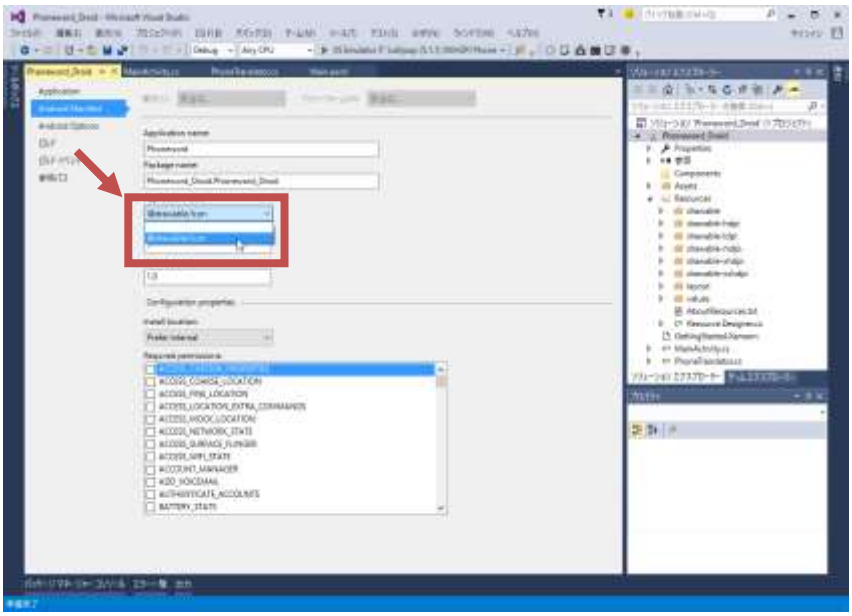


<< すべてのファイルを表示 >>



<< プロジェクトに含める >>

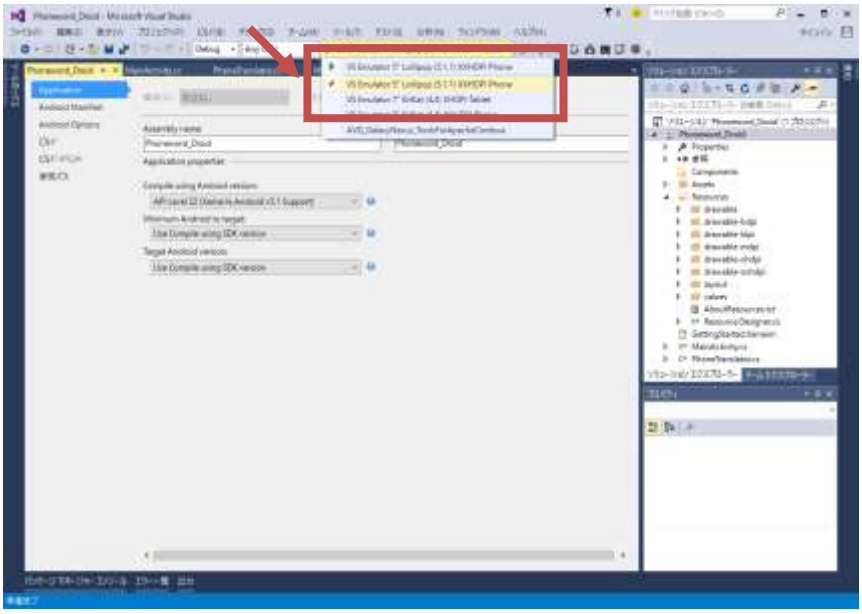
2.29　プロジェクトのプロパティの[Android Manifest]で[Application Icon]のドロップダウンメニューから[@drawable/icon]を選択します。



<< @drawable/Icon を選択 >>

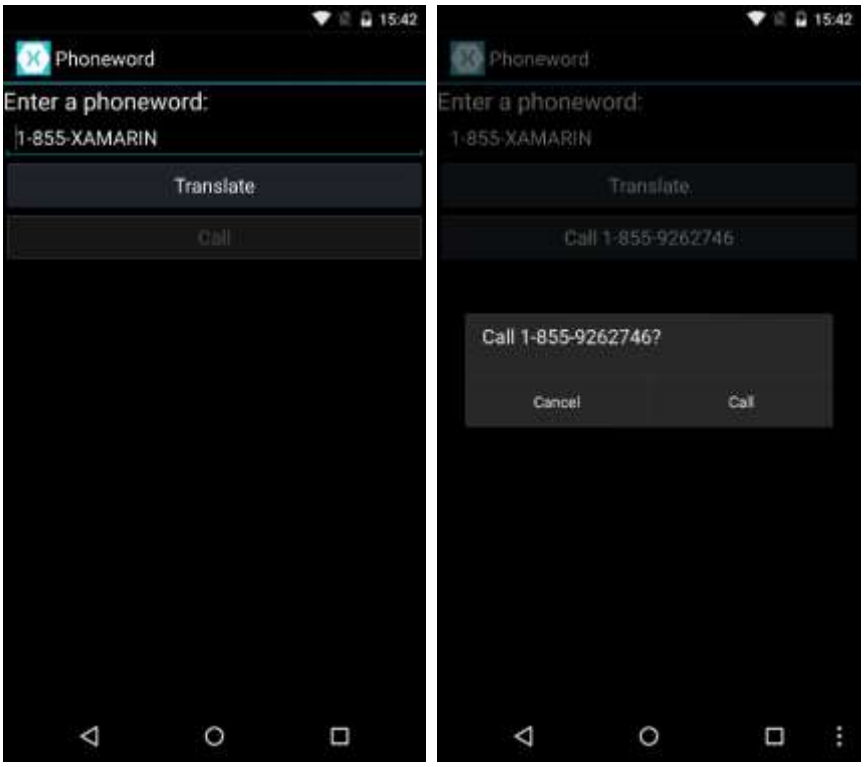
2.30　最後に Android のエミュレーターを使用して、アプリケーションをテストします。

プロジェクトプロパティにある[Application]ノードを開きます。[Compile using Android version]から、使用するエミュレーターの API レベルと同じまたは以下の API レベル を選択します。使用するエミュレーターをツールバー上の[Target Android Device]のドロップダウン メニューから選択します。



<< プロジェクト コンパイル設定 >>

2.31 ▶ アイコンをクリックしてエミュレーターでアプリケーションを表示します。下のスクリーンショットは、エミュレーター上で Phoneword アプリケーションを実行した際の図です。いくつかのエミュレーター上では、[ホーム]ボタンや[MENU]ボタンがアプリケーション内で動作するか確認する必要があります。[Translate]ボタンをクリックして、[Call]ボタンのテキストが更新され、[Call]をクリックした時に[call]ダイアログがスクリーン ショットのように表示されるのを確認してください。



<< VS Emulator >>

アプリケーションリストで Phoneword アプリケーションと設定したアイコンが表示されます。

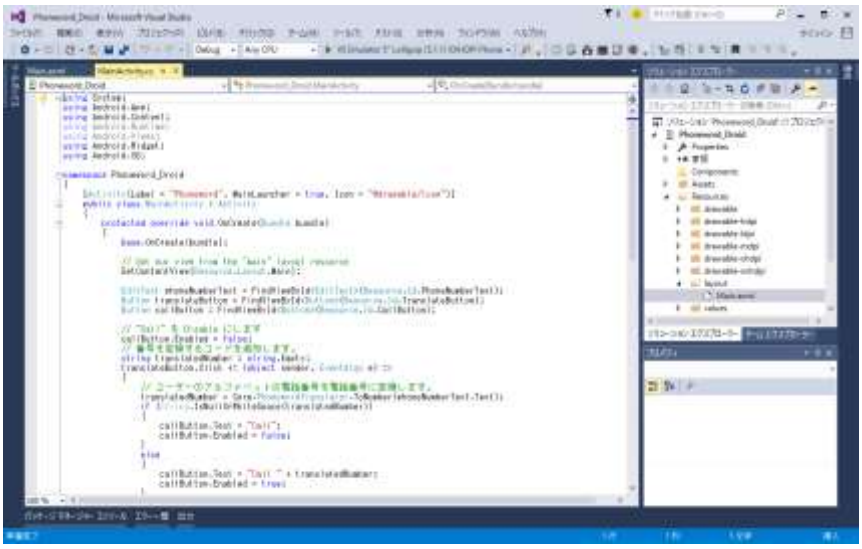
<< アプリケーション一覧 >>

初めての Xamarin.Android アプリケーションの完成です！次のステップ「Hello, Android Multiscreen Quickstart」で、このガイドで習得したツールとスキルをさらに試しましょう。

3　Hello, Android Multiscreen Quickstart

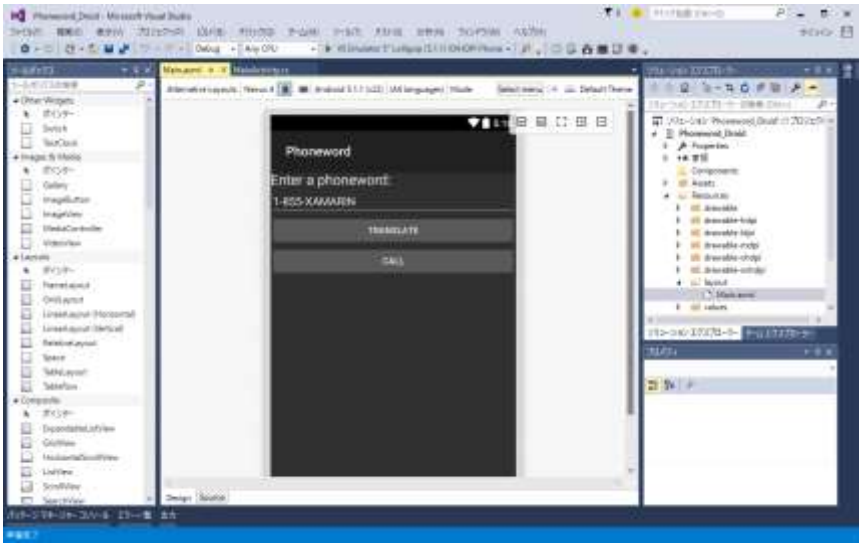
このセクションでは、Phoneword アプリケーションにもう一つ画面を追加し、その画面にこのアプリの通話履歴を残す方法を説明します。本ガイドで完成したアプリケーションでは、以下のスクリーン ショットのように、2 番目の画面に通話履歴を表示します。

3.1　Visual Studio で[Phoneword]プロジェクトを開きます。



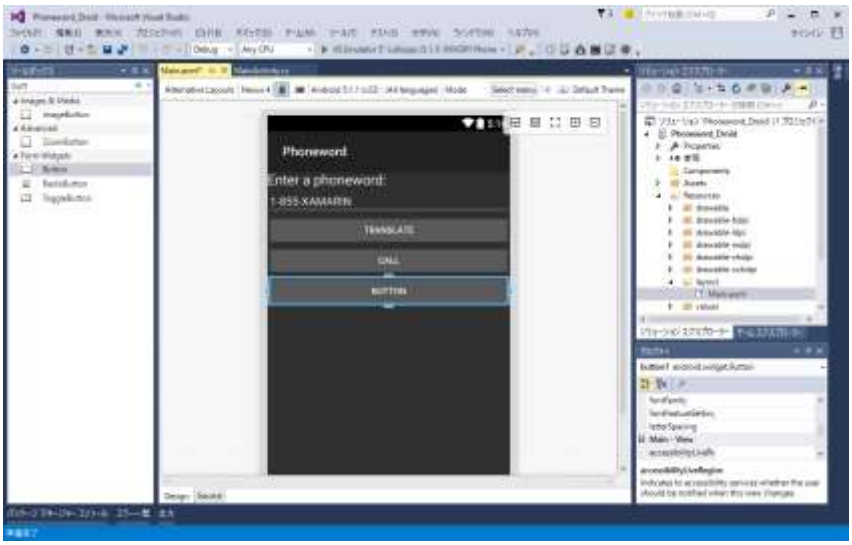
<< Visual Studio 2015 >>

3.2　まずはユーザーインターフェースの編集から始めます。ソリューションエクスプローラーから[Main.axml]ファイルを開きます。



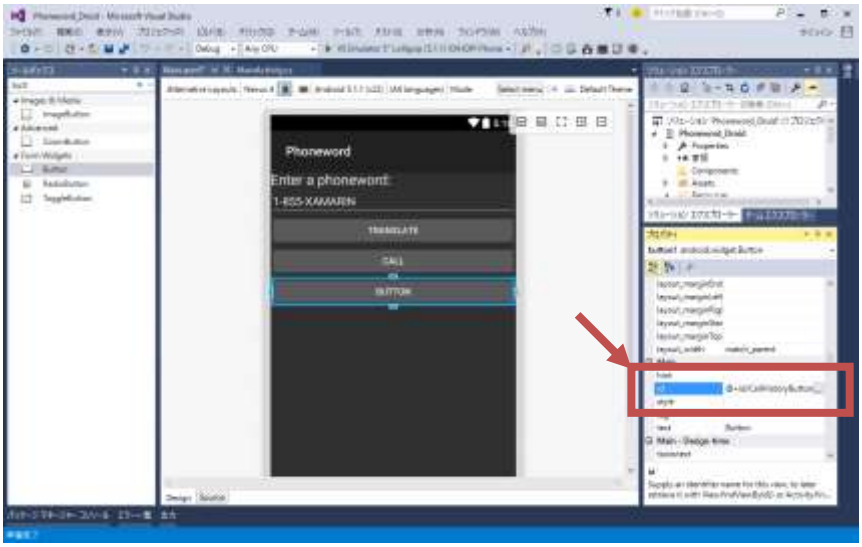
<< Main.axml >>

3.3　[ツールボックス]から、デザイン画面に[Button]をドラッグし、[Call]ボタンの下に配置します。



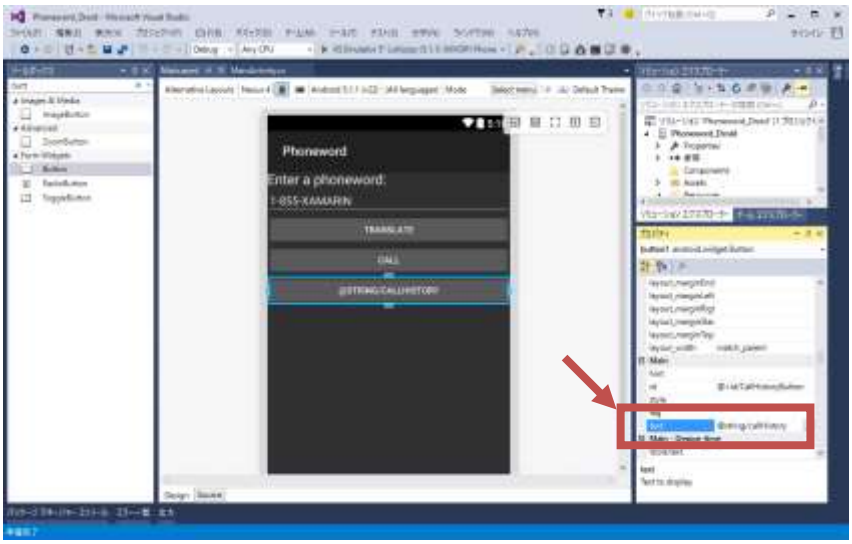
<< Button 配置 >>

3.4　[プロパティ]でボタン[Id]を「@+id/CallHistoryButton」に変更します。



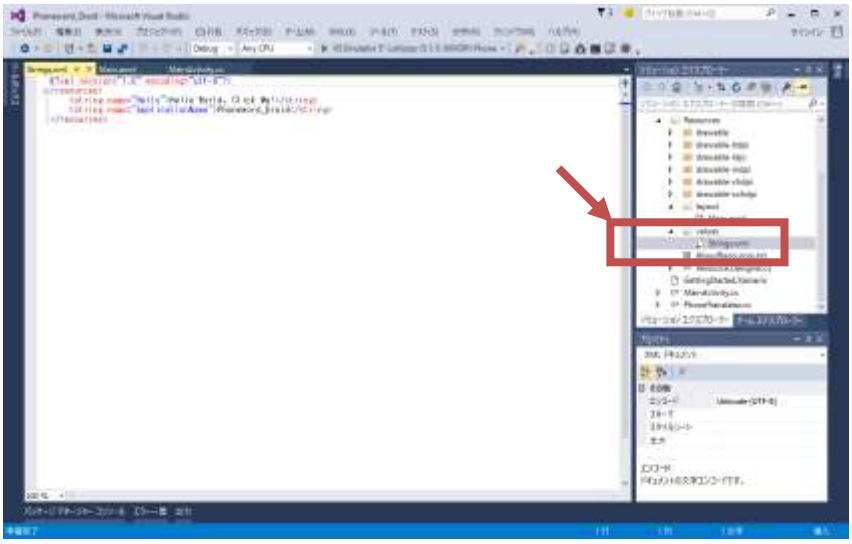
<< id を変更 >>

3.5　ボタンの[Text]プロパティを「@string/callHistory」に変更します。Android Designer には、記述した値がそのまま表示されますが、この後に行う変更によりボタンのテキストは正確に表示されます。



<< text を変更 >>

3.6　ソリューションエクスプローラーから[Resources]フォルダ以下の[values]フォルダを展開します。文字列のリソースファイル[Strings.xml]をダブルクリックして開きます。



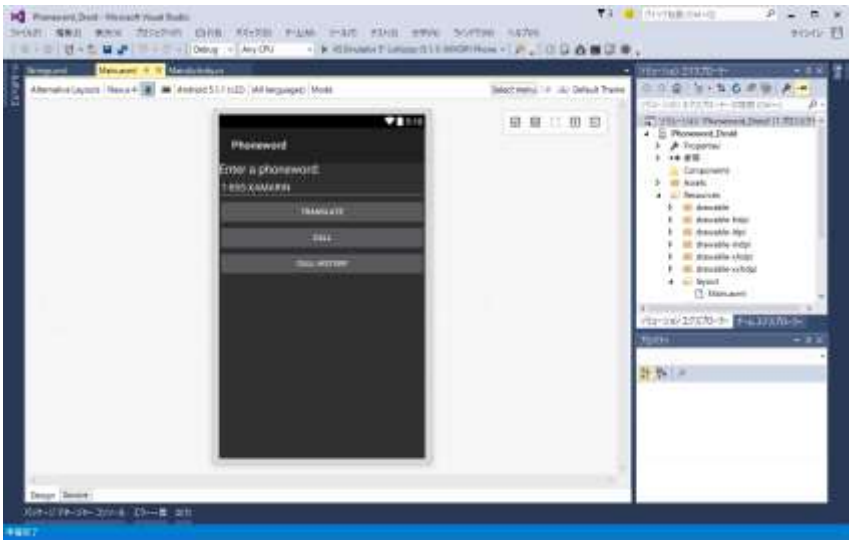
<< String.xml を開く >>

3.7　以下のコードで[Strings.xml]ファイルを上書きして保存します。

```
<?xml version="1.0" encoding="utf-8"?>
```

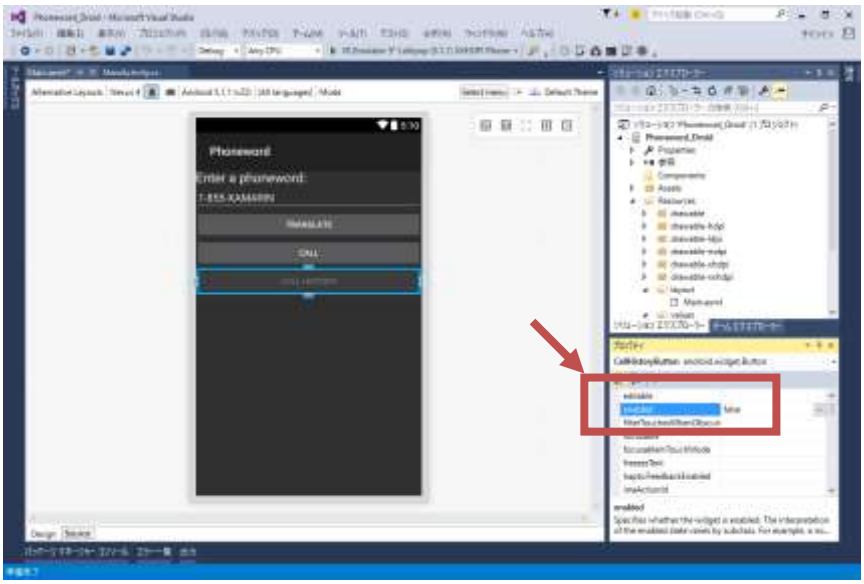
```
<resources>
    <string name="callHistory">Call History</string>
</resources>
```

[Call History]ボタンのテキストを更新すると新しい[string]の値が反映されます（反映されない場合は、再度ファイルを開くと反映されます）。



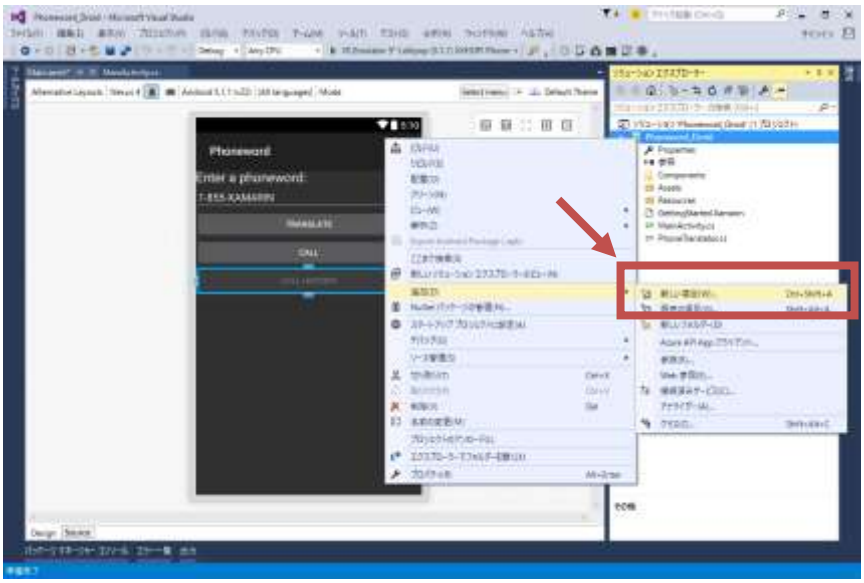
<< @string が反映されたボタン >>

3.8 デザイン画面で[Call History]ボタンを選択し、[プロパティ]の[enabled]の設定を見つけ、値を「false」に設定しボタンを無効にします。これにより、デザイン画面のボタンが暗く変化します。



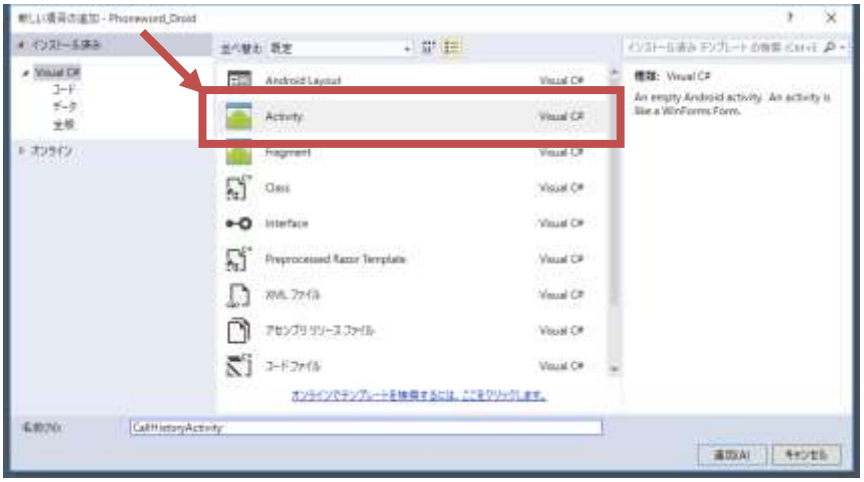
<< enable を false >>

3.9　2 つ目の画面を作成します。ソリューションエクスプローラー内の[Phoneword]プロジェクトを右クリックし、[追加 > 新しい項目]をクリックします。



<< 新しい項目 をクリック >>

3.10　[新しい項目の追加]ダイアログから[Android > Activity]を選択し、Activity に「CallHistoryActivity.cs」と名前を付けます。



3.11　[CallHistoryActivity.cs]のテンプレート コードを以下のコードに置き換えます。

```
using System;
using System.Collections.Generic;
using Android.App;
using Android.OS;
using Android.Widget;
namespace Phoneword_Droid
```

```
{
 [Activity(Label = "@string/callHistory")]
    public class CallHistoryActivity : ListActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);
            // Create your application here
            var phoneNumbers = Intent.Extras.GetStringArrayList("phone_numbers") ?? new
string[0];
            this.ListAdapter = new ArrayAdapter<string>(this,
Android.Resource.Layout.SimpleListItem1, phoneNumbers);
        }
    }
}
```

このクラスでは、[ListActivity]を生成し、プログラムで自動的にデータを格納するので、この Activity のための新しいレイアウトファイルを作成する必要はありません。

3.12 このアプリは、最初の画面でユーザーが電話をかけた電話番号を集め、その番号を 2 番目の画面に送ります。これで電話番号をリストのようにして記憶することができます。MainActivity クラスの上位に以下の using 宣言を加えて、リストのサポートをします。

```
using System.Collections.Generic;
```

その後、電話番号を格納する空のリストを生成します。MainActivity クラスは以下のようになります。

```
[Activity(Label = "Phoneword", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
{
    static readonly List<string> phoneNumbers = new List<string>();
    ...// OnCreate, etc.
}
```

3.13 [Call History]ボタンを紐づけします。[MainActivity]クラスに、以下のコードを追加し、ボタンを認識させ紐づけをします。[FindViewById]はコード上部に、[Click]イベントハンドラはコード下部に追加します。

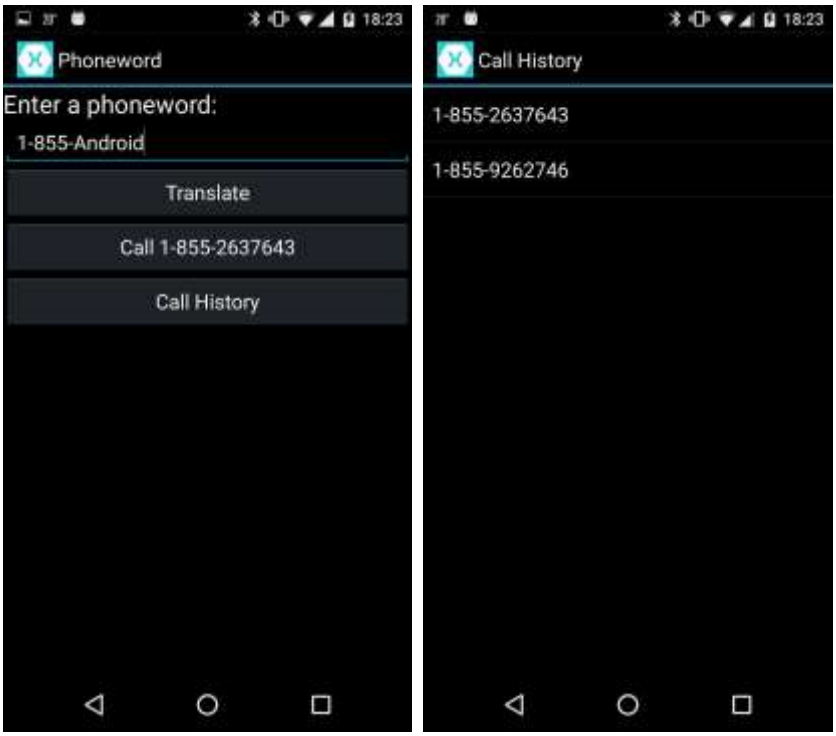
```
Button callHistoryButton = FindViewById<ButBton> (Resource.Id.CallHistoryButton);

callHistoryButton.Click += (sender, e) =>
{
    var intent = new Intent(this, typeof(CallHistoryActivity));
    intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
    StartActivity(intent);
};
```

3.14　Call ボタンの機能を拡張して、ユーザーが新しい番号に電話を掛けた時に、電話番号のリストに番号を追加し、Call History ボタンを有効にします。以下のように、Alert Dialog の Neutral Button のコードを変更して反映させます。

```
callDialog.SetNeutralButton("Call", delegate
{
    // 掛けた番号のリストに番号を追加します。
    phoneNumbers.Add(translatedNumber);
    // Call History ボタンを有効にします。
    callHistoryButton.Enabled = true;
    // 電話への intent を作成します。
    var callIntent = new Intent(Intent.ActionCall);
    callIntent.SetData(Android.Net.Uri.Parse("tel:" + translatedNumber));
    StartActivity(callIntent);
});
```

保存後、アプリケーションをビルドし、エラーがないか確認します。

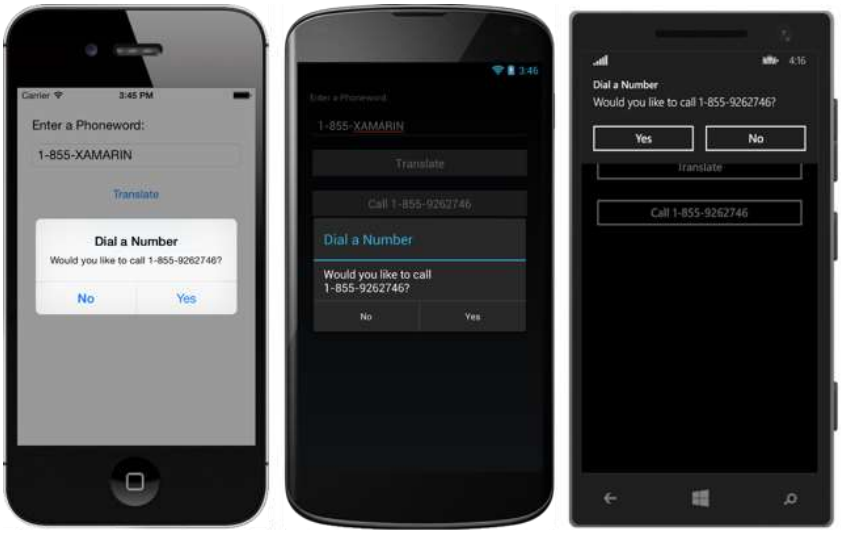
3.15　エミュレーターまたはデバイスでアプリケーションを実行してみましょう。以下のスクリーンショットは、Emulator で[Phoneword]アプリケーションを実行した時のイメージです。



おめでとうございます。複数画面を操作する最初の Xamarin.Android アプリケーションが完成しました！

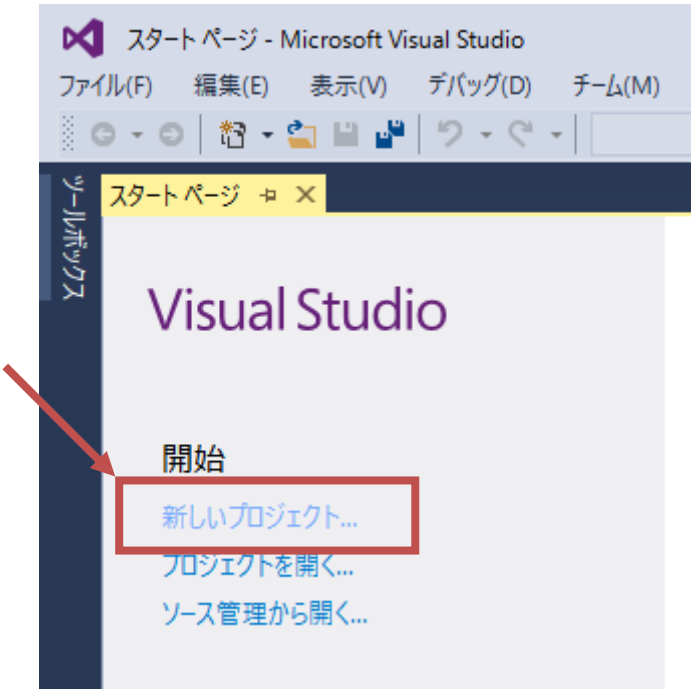
4　Xamarin.Forms Quickstart

このセクションでは、先ほど Android プロジェクトで作成した Phoneword アプリを Xamarin.Forms を使用して作成する方法を説明します。アプリケーションの完成図は以下のようになります。



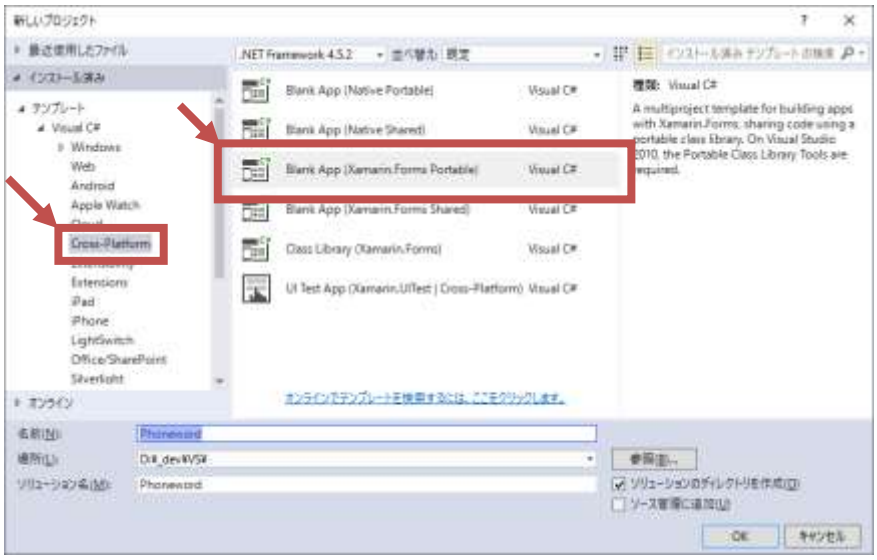
<< iOS Android Windows Phone アプリ完成図 >>

4.1　Visual Studio を起動し、[スタートページ > 新しいプロジェクト]をクリックして、新しいソリューションを作成します。



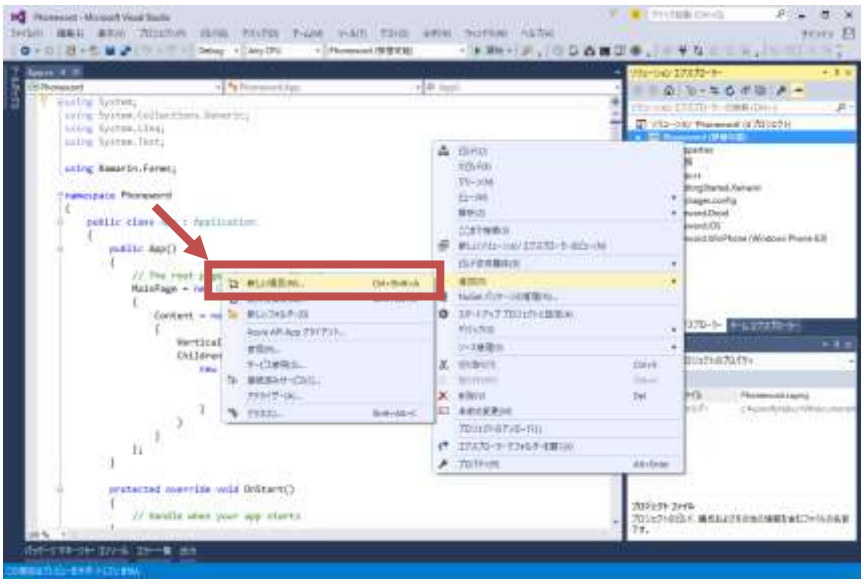
<< Visual Studio メイン画面 >>

4.2　[新しいプロジェクト]画面で、[Visual C# > Cross-platform]をクリックします。[Blank App (Xamarin.Forms Portable)]テンプレートを選択します。新しいソリューションには、名前を「Phoneword」と付けます。



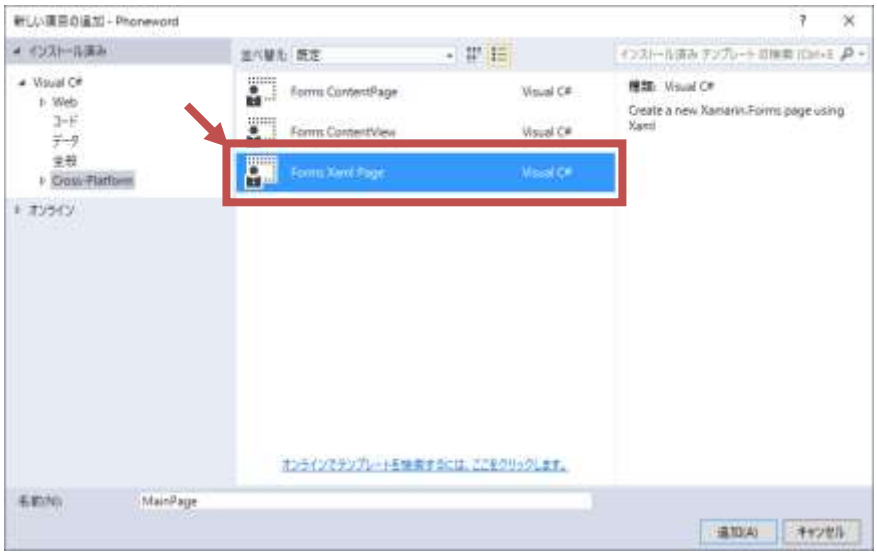
<< 新しいプロジェクト >>

4.3　ソリューションエクスプローラーで[Phoneword]プロジェクトを右クリックし、[追加 > 新しい項目]をクリックします。



<< 新しい項目 >>

4.4　[新しい項目の追加]画面から、[Visual C# > Cross-Platform > Forms Xaml Page]を選択し、新しいファイルの名前を[MainPage]と付け、[追加]ボタンをクリックします。



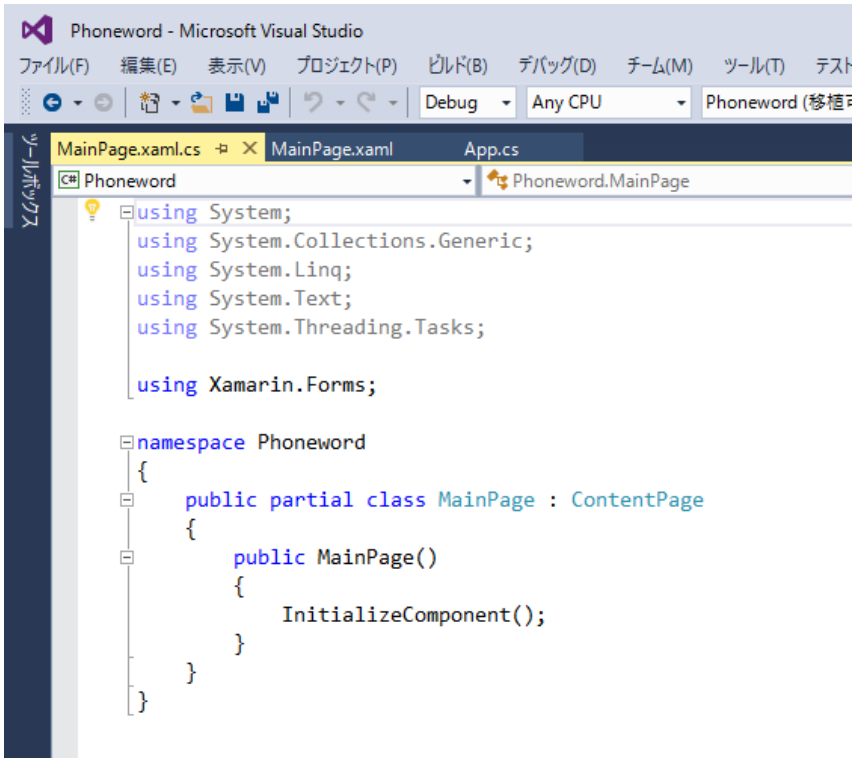
<< Forms Xaml Page を追加 >>

4.5　MainPage.xaml ですべてのテンプレートコードを削除し、以下のコードで置き換えます。

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Phoneword.MainPage">
    <ContentPage.Padding>
       <OnPlatform x:TypeArguments="Thickness"
                   iOS="20, 40, 20, 20"
                   Android="20, 20, 20, 20"
                   WinPhone="20, 20, 20, 20" />
    </ContentPage.Padding>
   <ContentPage.Content>
       <StackLayout VerticalOptions="FillAndExpand"
                    HorizontalOptions="FillAndExpand"
                    Orientation="Vertical"
                    Spacing="15">
           <Label Text="Enter a Phoneword:" />
           <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
           <Button x:Name="translateButon" Text="Translate" Clicked="OnTranslate" />
           <Button x:Name="callButton" Text="Call" IsEnabled="false" Clicked="OnCall" />
       </StackLayout>
   </ContentPage.Content>
</ContentPage>
```

[Ctrl＋S]を押して変更を保存します。

4.6　ソリューションエクスプローラーで[MainPage.xaml]を展開し、[MainPage.xaml.cs]をダブルクリックして開きます。



<< MainPage.xaml >>

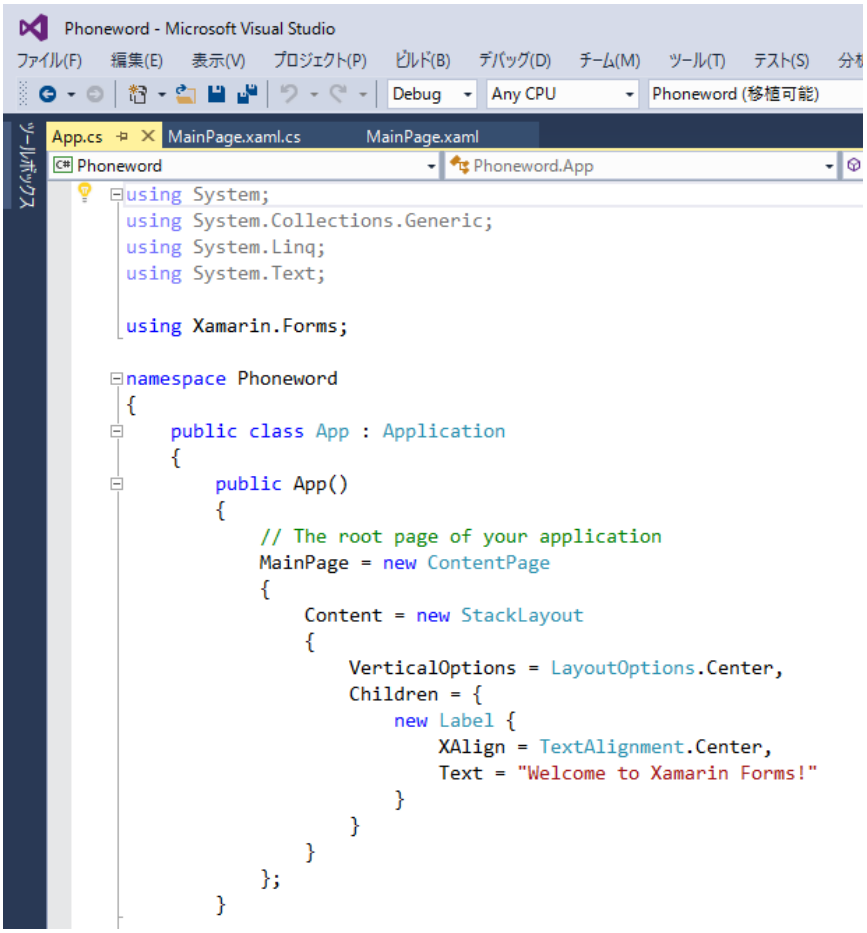
4.7　[MainPage.xaml.cs]ですべてのテンプレートコードを以下のコードで置き換えます。[OnTranslate]と[OnCall]メソッドはユーザーインターフェースの[Translate]と[Call]ボタンがクリックされた時にそれぞれ実行されます。

```
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Phoneword
{
    public partial class MainPage : ContentPage
    {
        string translatedNumber;

        public MainPage ()
        {
            InitializeComponent ();
        }

        void OnTranslate (object sender, EventArgs e)
```

```
        {
            translatedNumber = Core.PhonewordTranslator.ToNumber (phoneNumberText.Text);
            if (!string.IsNullOrWhiteSpace (translatedNumber)) {
                callButton.IsEnabled = true;
                callButton.Text = "Call " + translatedNumber;
            } else {
                callButton.IsEnabled = false;
                callButton.Text = "Call";
            }
        }

        async void OnCall (object sender, 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)
                    dialer.Dial (translatedNumber);
            }
        }
    }
}
```

[Ctrl＋S]を押して変更を保存します。

4.8　ソリューションエクスプローラーで[App.cs]をダブルクリックして開きます。



<< App.cs >>

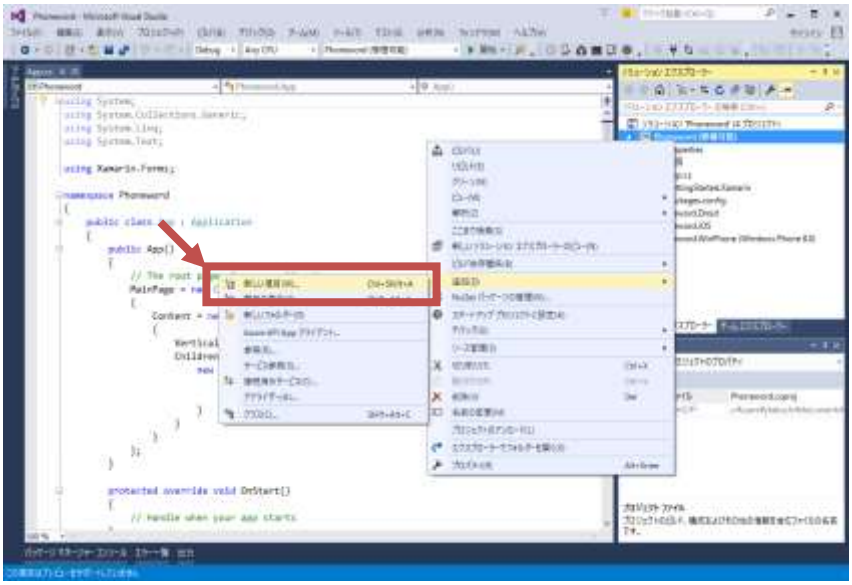
4.9　[App.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。

```
using System;
using Xamarin.Forms;

namespace Phoneword
{
    public class App : Application
    {
        public App ()
        {
            MainPage = new Phoneword.MainPage ();
        }

        protected override void OnStart ()
        {
            // Handle when your app starts
        }

        protected override void OnSleep ()
```

```
        {
            // Handle when your app sleeps
        }

        protected override void OnResume ()
        {
            // Handle when your app resumes
        }
    }
}
```

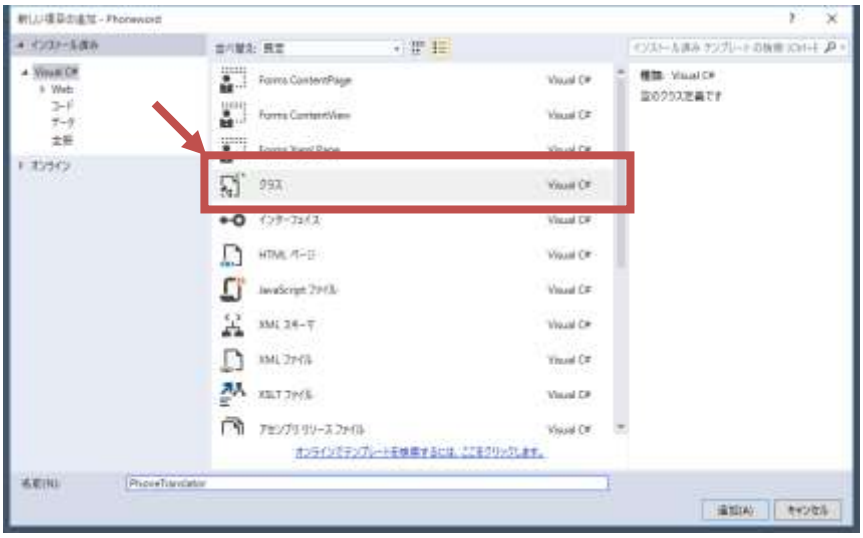
[Ctrl＋S]を押して変更を保存します。

4.10　ソリューションエクスプローラーで、[Phoneword（移植可能）]プロジェクトを右クリックし、[追加 ＞ 新しい項目]をクリックします。



<< 新しい項目 >>

4.11　[新しい項目の追加]画面から、[Visual C# > クラス]を選択し、新しいファイルの名前を「PhoneTranslator」と付け、[追加]ボタンをクリックします。



<< クラス を追加 >>

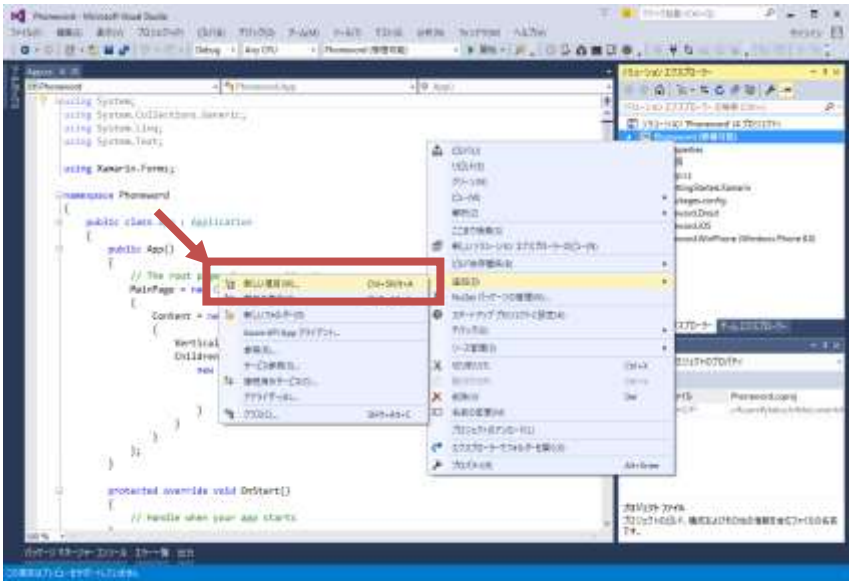
4.12　[PhoneTranslator.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。

```
using System.Text;

namespace Core
{
    public static class PhonewordTranslator
    {
        public static string ToNumber(string raw)
        {
            if (string.IsNullOrWhiteSpace(raw))
                return null;

            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);
                    // Bad character?
                    else
                        return null;
                }
```

```
            }
            return newNumber.ToString();
        }

        static bool Contains(this string keyString, char c)
        {
            return keyString.IndexOf(c) >= 0;
        }

        static readonly string[]digits = {
            "ABC", "DEF", "GHI", "JKL", "MNO", "PQRS", "TUV", "WXYZ"
        };

        static int? TranslateToNumber(char c)
        {
            for (int i = 0; i < digits.Length; i++)
            {
                if (digits[i].Contains(c))
                    return 2 + i;
            }
            return null;
        }
    }
}
```

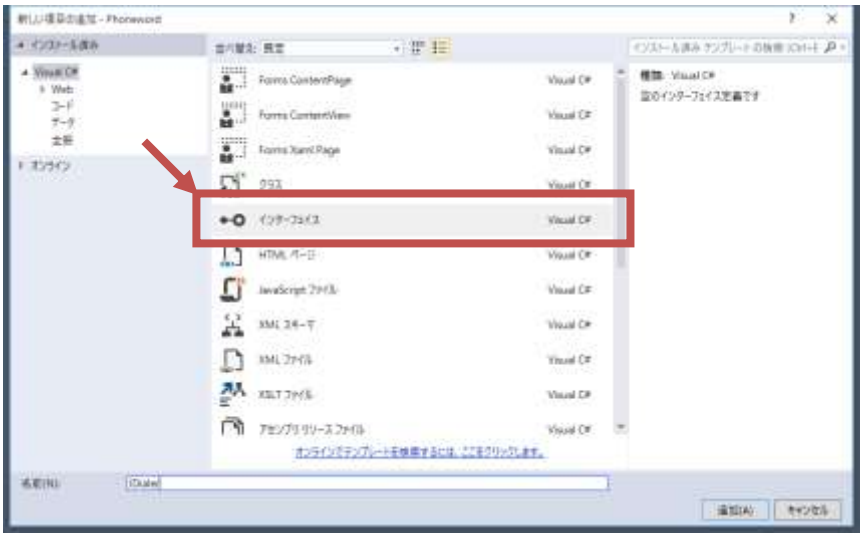
[Ctrl＋S]を押して変更を保存します。

4.13　ソリューションエクスプローラーで、[Phoneword]ソリューションを右クリックし、[追加 ＞ 新しい項目]をクリックします。



<< 新しい項目 >>

4.14　[新しい項目の追加]画面から、[Visual C# > インターフェイス]を選択し、新しいファイルの名前を「IDialer」と付け、[追加]ボタンをクリックします。



<< インターフェイス を追加 >>

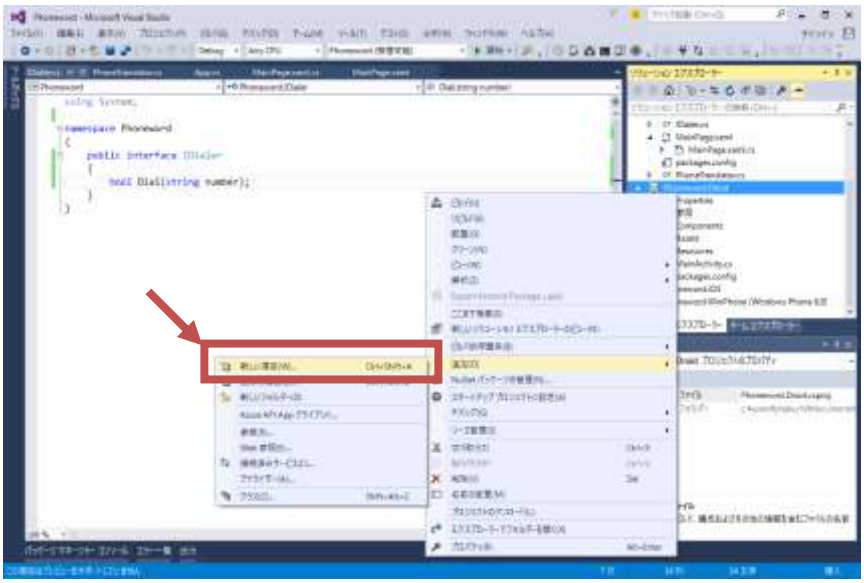
4.15　[IDialer.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。[Dial]メソッドは変換された電話番号に電話を掛けるために各プラットフォームで実装が必要です。

```
using System;

namespace Phoneword
{
    public interface IDialer
    {
        bool Dial(string number);
    }
}
```

[Ctrl＋S]を押して変更を保存します。

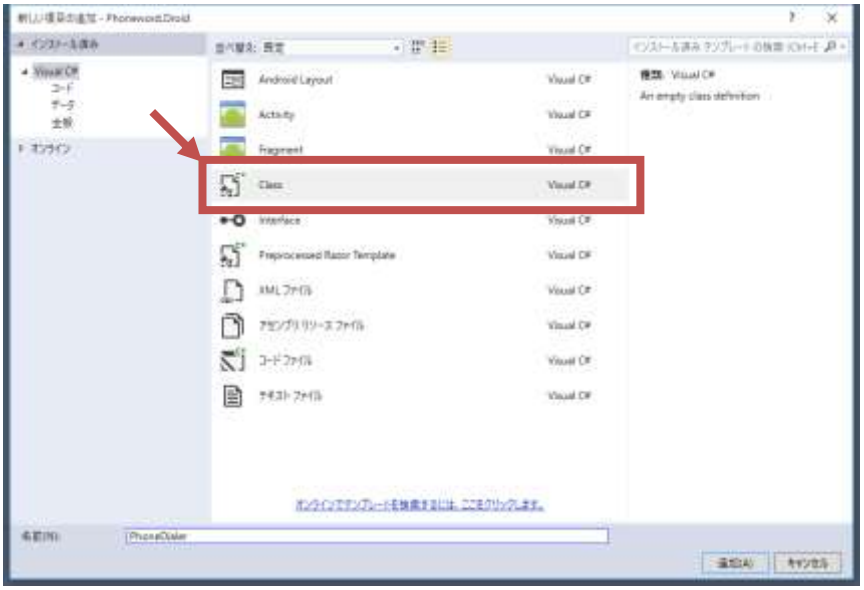
アプリケーションの共通コードはこれで完了です。各プラットフォームで電話を掛けるコードは DependencyService として実装します。なお、本ガイドでは、Android、UWP、Windows Phone 8.1 の実装を行います。

4.16　ソリューションエクスプローラーで、Android プロジェクト[Phoneword.Droid]プロジェクトを右クリックし、[追加 ＞ 新しい項目]をクリックします。



<< 新しい項目 >>

4.17　[新しい項目の追加]画面から、[Visual C# > Class]を選択し、新しいファイルの名前を「PhoneDialer」と付け、[追加]ボタンをクリックします。



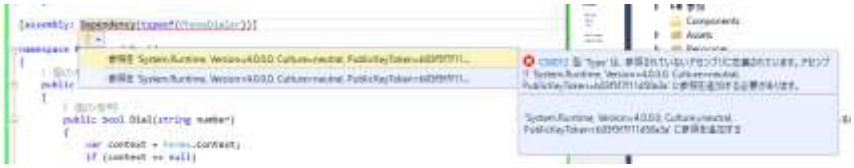
<< Class を追加 >>

4.18 [PhoneDialer.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。このコードは Android で変換された電話番号に電話を掛ける[Dial]メソッドを含んでいます。

```
using Android.Content;
using Android.Telephony;
using Phoneword.Droid;
using System.Linq;
using Xamarin.Forms;

using Uri = Android.Net.Uri;

[assembly: Dependency(typeof(PhoneDialer))]

namespace Phoneword.Droid
{
    public class PhoneDialer : IDialer
    {
        public bool Dial(string number)
        {
            var context = Forms.Context;
            if (context == null)
                return false;

            var intent = new Intent(Intent.ActionCall);
            intent.SetData(Uri.Parse("tel:" + number));

            if (IsIntentAvailable(context, intent))
            {
                context.StartActivity(intent);
                return true;
            }

            return false;
        }

        public static bool IsIntentAvailable(Context context, Intent intent)
        {
            var packageManager = context.PackageManager;

            var list = packageManager.QueryIntentServices(intent, 0)
                .Union(packageManager.QueryIntentActivities(intent, 0));

            if (list.Any())
                return true;

            var manager = TelephonyManager.FromContext(context);
            return manager.PhoneType != PhoneType.None;
        }
    }
}
```

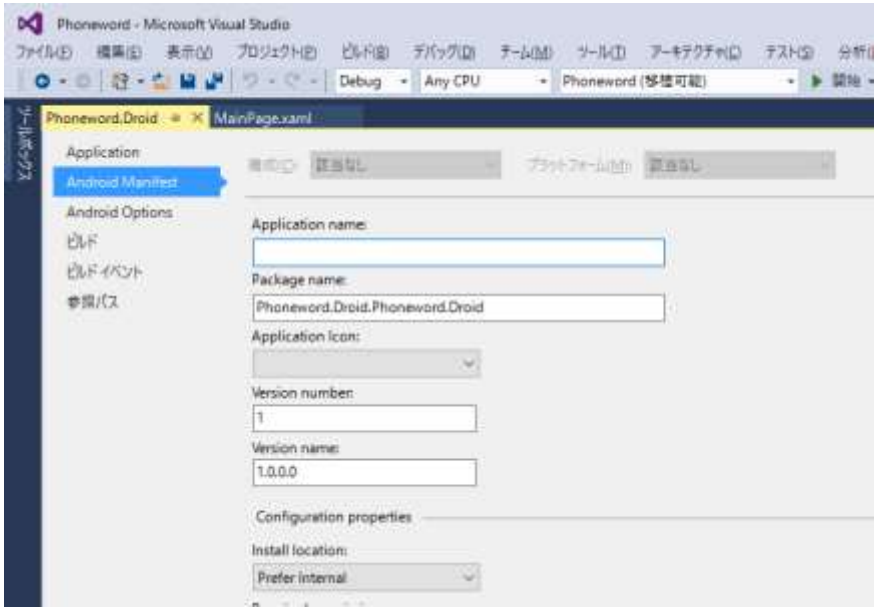
[Ctrl＋S]を押して変更を保存します。

4.19　アセンブリの参照エラーが表示される場合は、［Ctrl+.］を押して参照を追加してください。



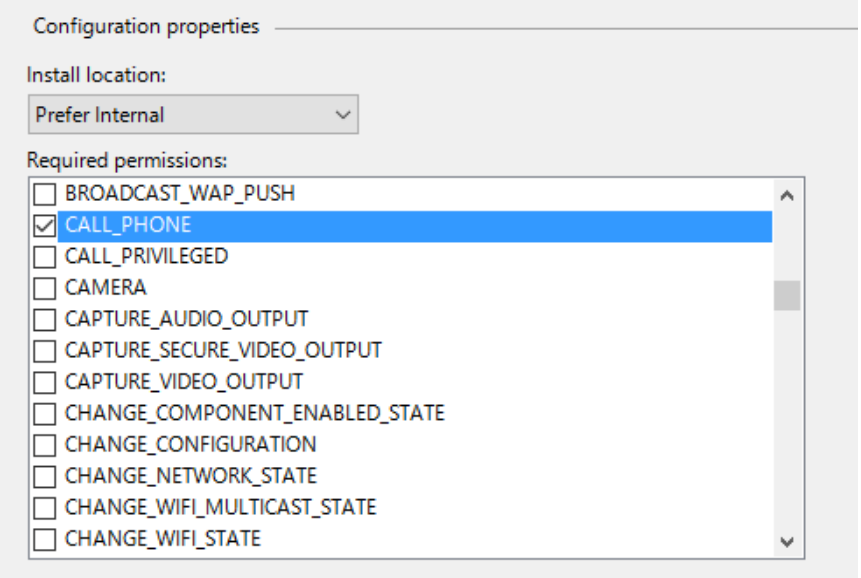
<< 参照を追加 >>

4.20　ソリューションエクスプローラーで[Phoneword.Droid]の[Properties]をダブルクリックしてプロジェクトのプロパティを開き、[Android Manifest]をクリックします。



<< Android Manifest >>

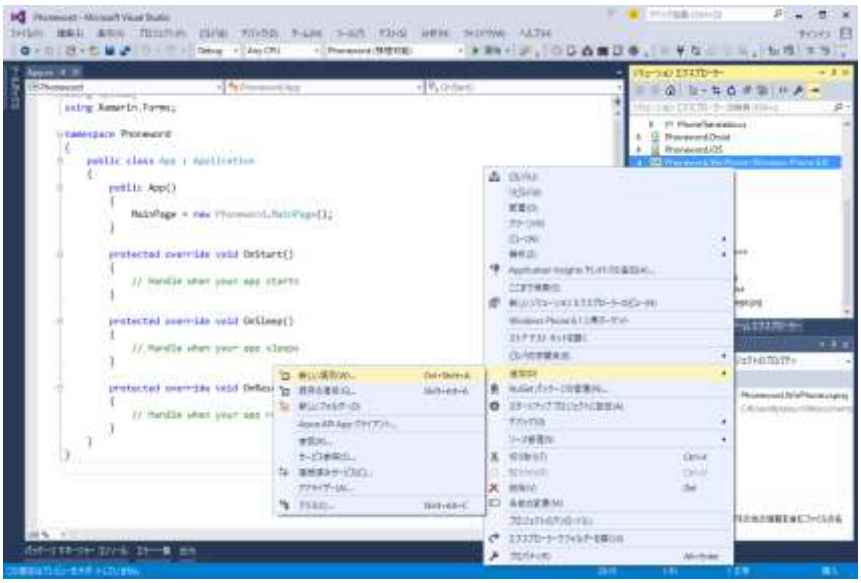
4.21　[Required permissions]欄で[CALL_PHONE]をチェックします。これでアプリケーションが電話を掛けられるように Permission が設定されます。



<< CALL_PHONE をチェック >>

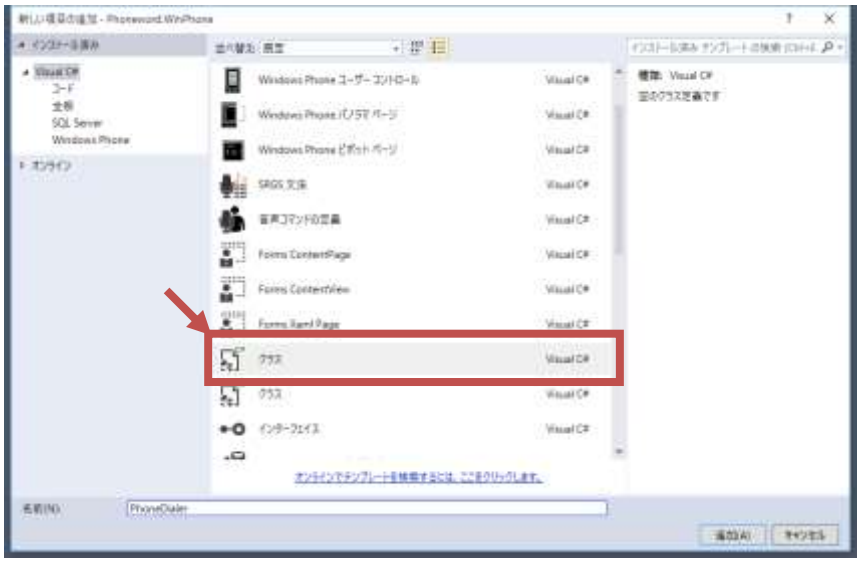
[Ctrl＋S]を押して変更を保存します。

4.22　ソリューションエクスプローラーで、Windows Phone プロジェクト[Phoneword.WinPhone]プロジェクトを右クリックし、[追加 ＞ 新しい項目]をクリックします。



<< Windows Phone プロジェクトに新しい項目を追加 >>

4.23　[新しい項目の追加]画面から、[Visual C# > クラス]を選択し、新しいファイルの名前を PhoneDialer と付け、[追加]ボタンをクリックします。



<< クラス を追加 >>

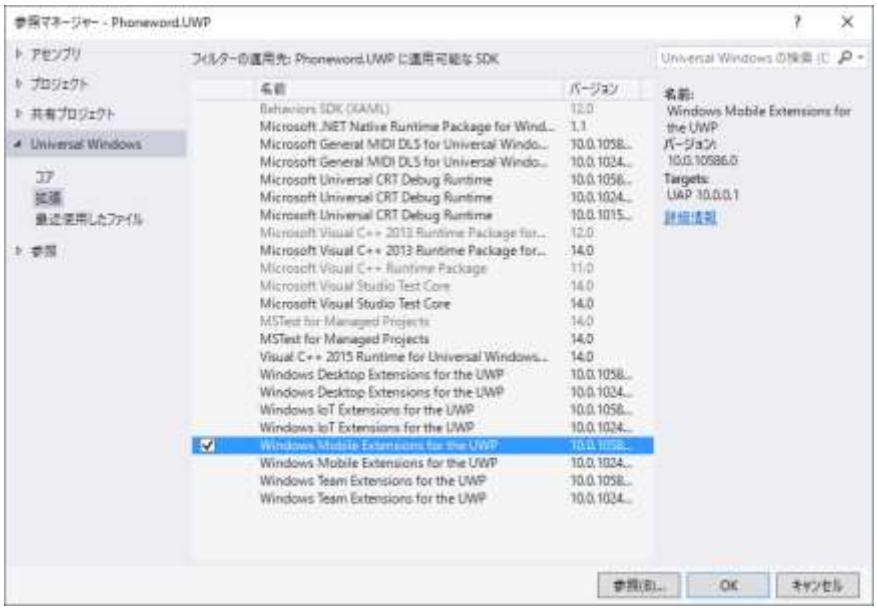
4.24　[PhoneDialer.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。このコードは Windows Phone 8.1 で変換された電話番号で電話アプリを表示するメソッドを含んでいます。

```
using Windows.ApplicationModel.Calls;
using Phoneword.WinPhone;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]

namespace Phoneword.WinPhone
{
    public class PhoneDialer : IDialer
    {
        public bool Dial(string number)
        {
            PhoneCallManager.ShowPhoneCallUI(number, "Phoneword");
            return true;
        }
    }
}
```

[Ctrl＋S]を押して変更を保存します。

4.25　同様に、UWP プロジェクト[Phoneword.UWP]にも[PhoneDialer.cs]クラスファイルを追加し、上記コードを記述します。

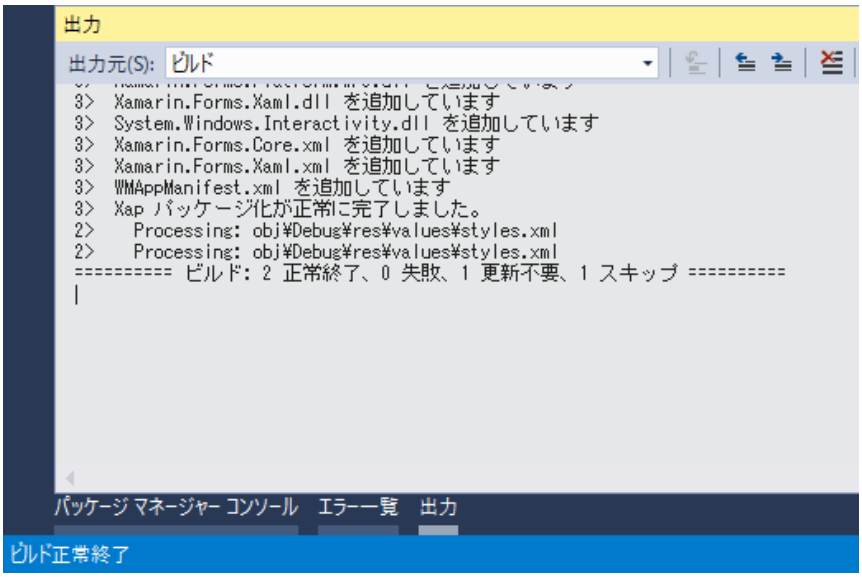
4.26　UWP で電話を掛ける（＝Windows 10 Mobile の機能を使用する）には、UWP の拡張ライブラリを追加する必要があります。UWP プロジェクトの参照を右クリックして[参照の追加]を選択し、[Universal Windows ＞ 拡張]から[Windows Mobile Extensions for the UWP（10.0.10586）]を追加します。



4.27　[IDialer]が見つからないというエラーが表示されることがありますが、Xamarin.Forms のプロジェクト[Phoneword]をビルドすると解決します。

<< IDialer エラー >>

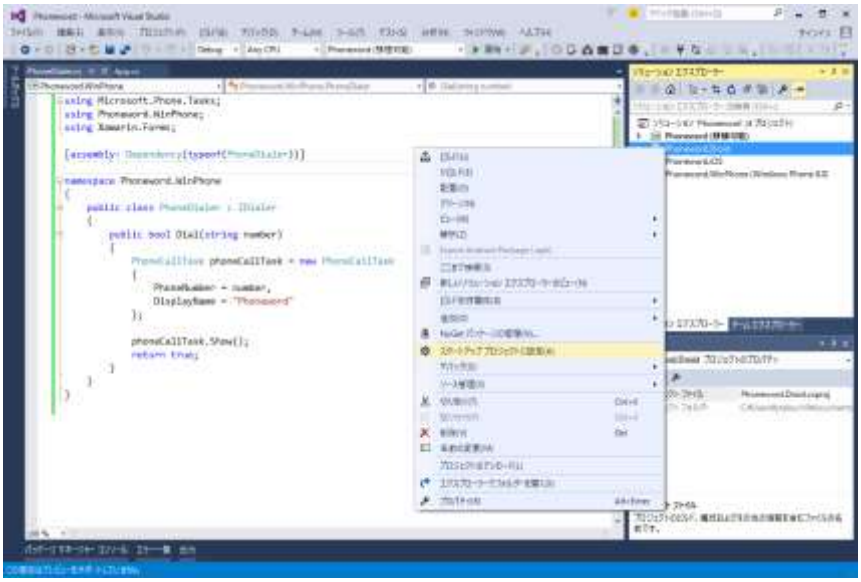
4.28　Visual Studio のメニューから[ビルド ＞ ソリューションのビルド]をクリックします。[出力]ウィンドウにビルド成功の表示がされます。



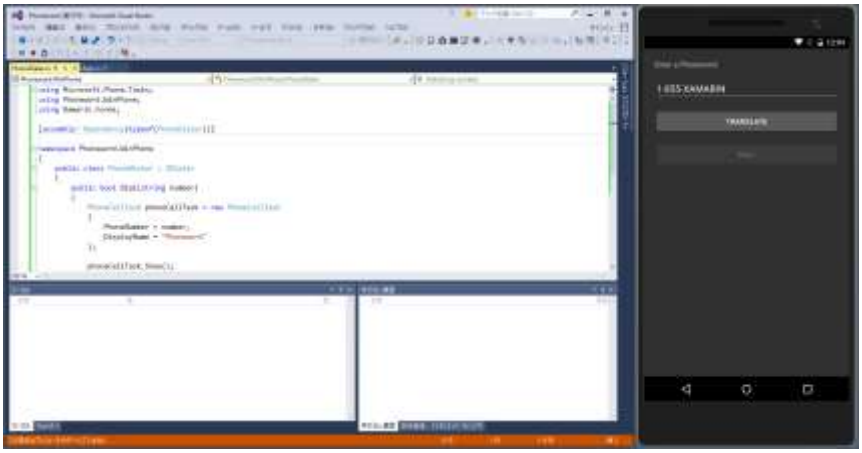
<< ビルド正常終了 >>

エラーが表示された場合は、表示が消えるまで修正を行ってください。

4.29　Visual Studio で[Phoneword.Droid]をスタートアッププロジェクトに設定し、▶ ボタンをクリックし、アプリケーションを起動します。



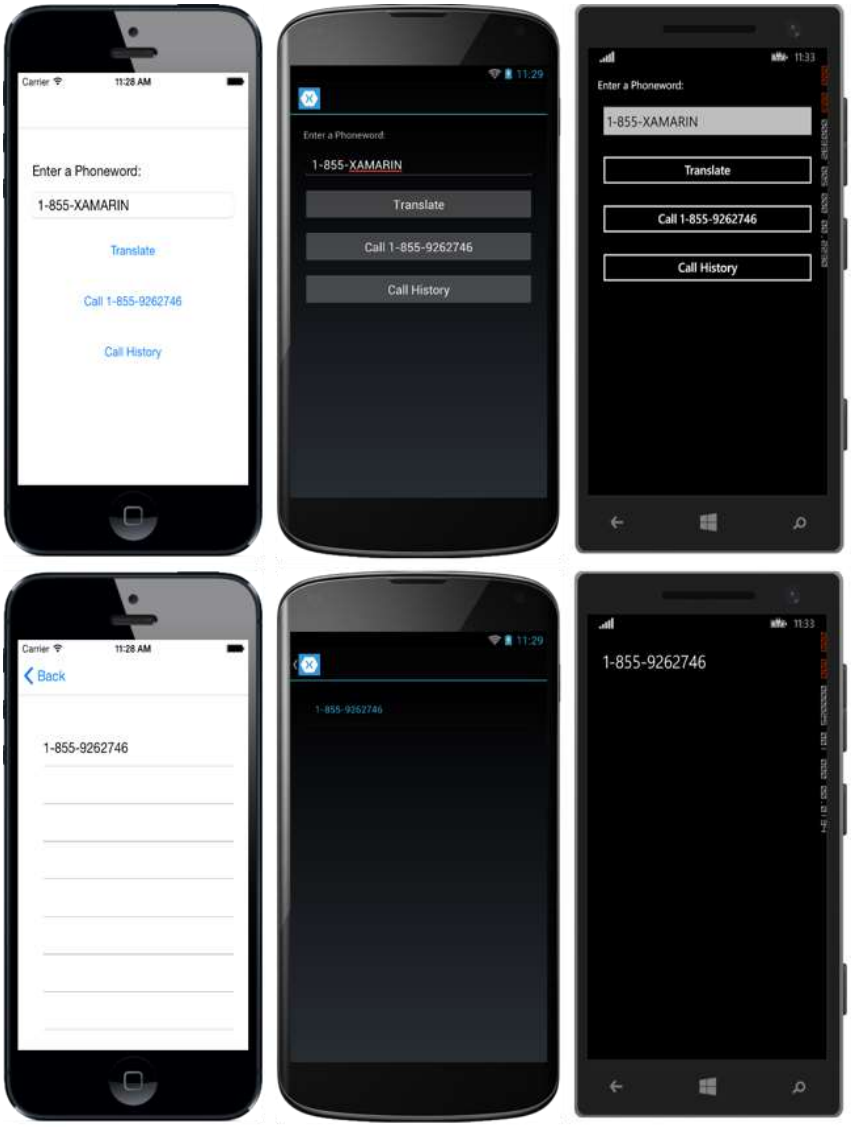
<< スタートアッププロジェクトに設定 >>



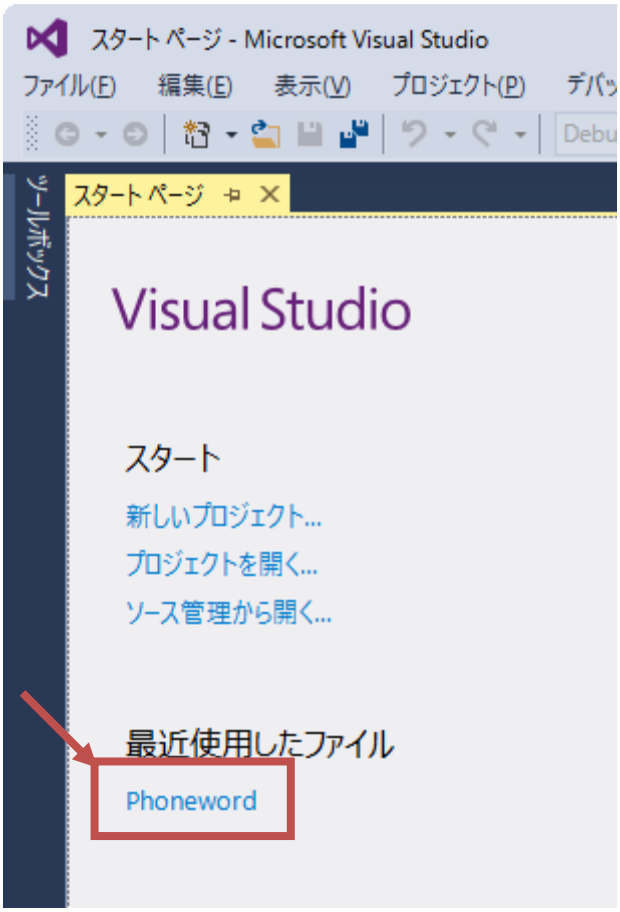
<< Android アプリ実行画面 >>

5　Xamarin.Forms Multiscreen Quickstart

このセクションでは、先ほど作成した Xamarin.Forms アプリケーションをマルチスクリーンに対応させます。完成図は次のようになります。

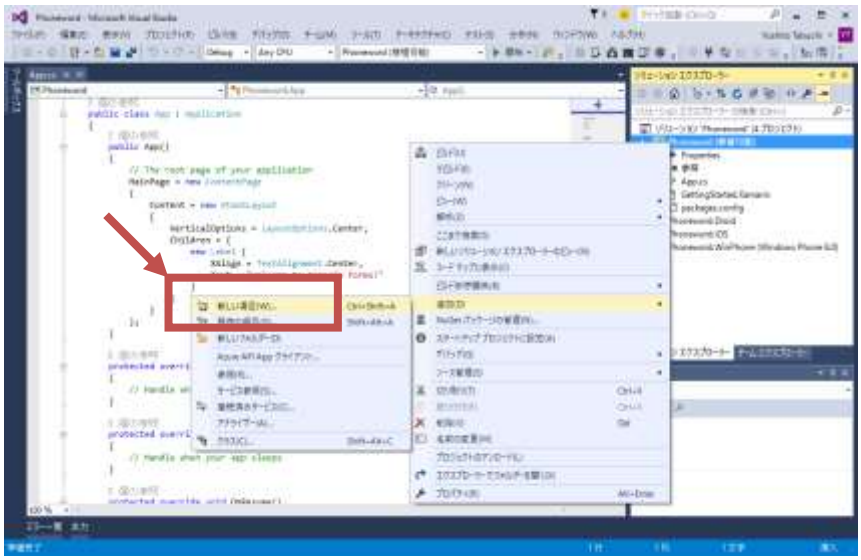


5.1　先ほどまで作業していた Phoneword プロジェクトを開きます。



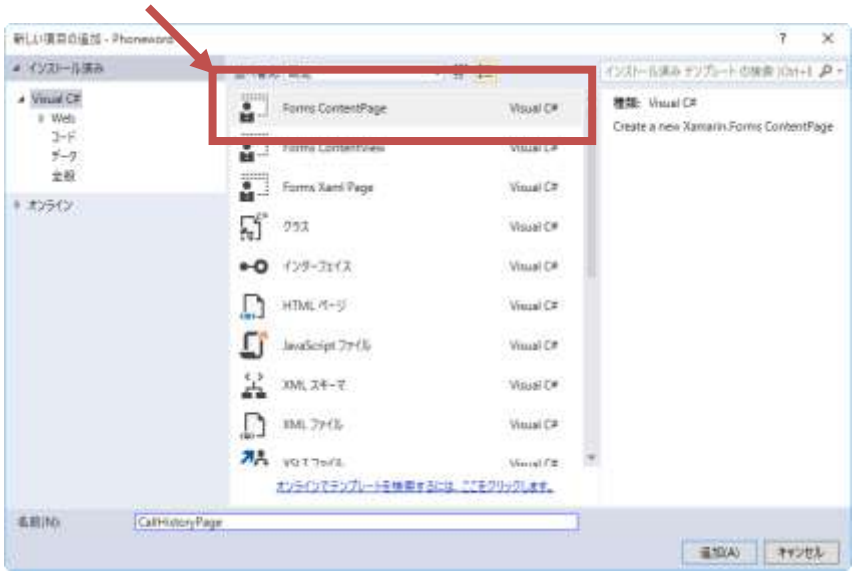
<< 最近使用したファイル >>

5.2　ソリューションエクスプローラーで Phoneword プロジェクトを右クリックし、[追加 > 新しい項目]をクリックします。



<< 新しい項目 >>

5.3　[新しい項目の追加]画面から、[Visual C# > Forms ContentPage]を選択し、新しいファイルの名前を CallHistoryPage と付け、[追加]ボタンをクリックします。



<< Forms ContentPage の追加 >>

5.4 [CallHistoryPage.cs]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。このコードはページのユーザーインターフェースの定義を宣言しています。

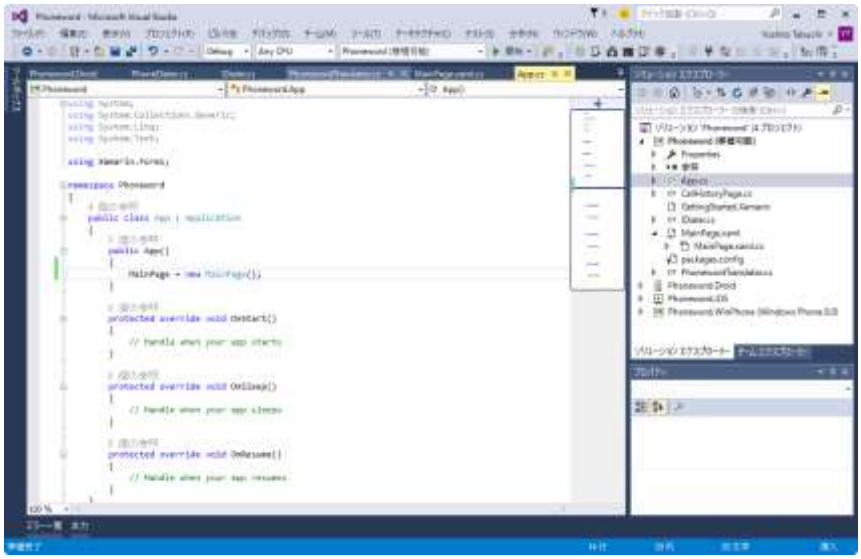
```
using System;
using Xamarin.Forms;

namespace Phoneword
{
    public class CallHistoryPage : ContentPage
    {
        public CallHistoryPage()
        {
            Title = "Recent Call";
            Content = new StackLayout
            {
                VerticalOptions = LayoutOptions.FillAndExpand,
                Orientation = StackOrientation.Vertical,
                Children = {
                    new ListView
                    {
                        ItemsSource = App.PhoneNumbers,
                    }
                }
            };
        }
    }
}
```

[Ctrl＋S]を押して変更を保存します。

今回、[CallHistoryPage]は XAML ではなく C#で作成しました。XAML で記述したい場合は次のコードとなります。

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:local="clr-namespace:Phoneword;assembly=Phoneword"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Phoneword.CallHistoryPage">
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand"
                     Orientation="Vertical">
            <ListView ItemsSource="{x:Static local:App.PhoneNumbers}" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

5.5　ソリューションエクスプローラーで App.cs をダブルクリックして開きます。



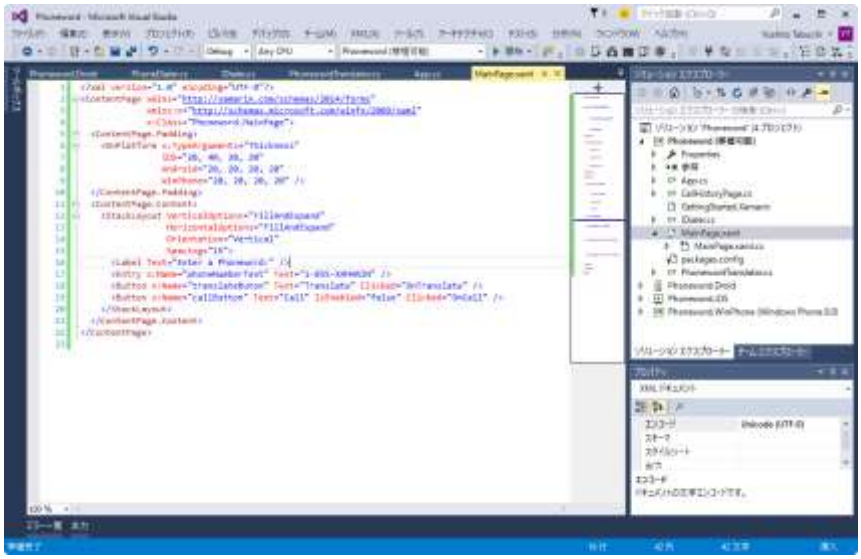
<< App.cs >>

5.6　[App.cs]で[System.Collections.Generic]を名前空間に追加し、[PhoneNumbers]プロパティを宣言し、App のコンストクラターで初期化します。そして[MainPage]を NavigationPage で初期化するように変更します。[PhoneNumbers]コレクションは変換された各電話番号を保存するために使用されます。コードの一部は次のようになります。

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

namespace Phoneword
{
    public class App : Application
    {
        public static List<string> PhoneNumbers { get; set; }

        public App ()
        {
            PhoneNumbers = new List<string>();
            MainPage = new NavigationPage(new Phoneword.MainPage());
        }
        ...
        ...
        ...
    }
}
```

[Ctrl＋S]を押して変更を保存します。

5.7　ソリューションエクスプローラーで MainPage.xaml をダブルクリックして開きます。



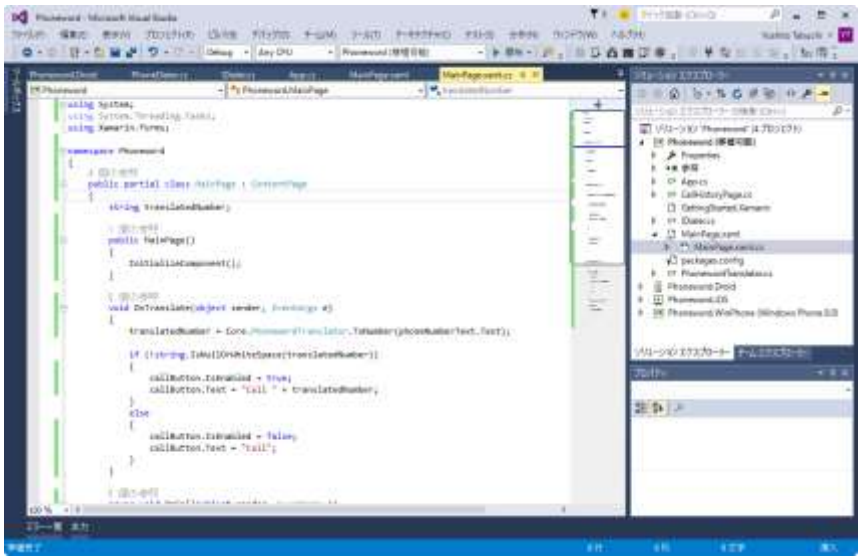
<< MainPage.xaml >>

5.8　MainPage.xaml で[Button]コントロールを[StackLayout]の最後に追加します。このボタンは CallHistoryPage にナビゲートするために使用されます。コードの一部は次のようになります。

```
<StackLayout VerticalOptions="FillAndExpand"
             HorizontalOptions="FillAndExpand"
             Orientation="Vertical"
             Spacing="15">
  ...
  <Button x:Name="callButton" Text="Call" IsEnabled="false" Clicked="OnCall" />
  <Button x:Name="callHistoryButton" Text="Call History" IsEnabled="false"
          Clicked="OnCallHistory" />
</StackLayout>
```

[Ctrl＋S]を押して変更を保存します。

5.9　ソリューションエクスプローラーで MainPage.xaml.cs をダブルクリックして開きます。



<< MainPage.xaml.cs >>

5.10　MainPage.xaml.cs で[OnCallHistory]イベントハンドラーメソッドを追加します。次に[OnCall]イベントハンドラーメソッドを変換した電話番号を[App.PhoneNumbers]コレクションに追加し、dialer 変数が null ではない時に callHistoryButton を enable にするように修正します。コードの一部は次のようになります。

```
using System;
using Xamarin.Forms;

namespace Phoneword
{
    public partial class MainPage : ContentPage
    {
        ...

        async void OnCall(object sender, EventArgs e)
        {
            If (await this.DisplayAlert(
                  ...
            {
                Var dialer = DependencyService.Get<IDialer>();
                if (dialer != null)
                {
                    App.PhoneNumbers.Add(translatedNumber);
                    callHistoryButton.IsEnabled = true;
                    dialer.Dial (translatedNumber);
                }
            }
        }
```

```
        async void OnCallHistory(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new CallHistoryPage());
        }
    }
}
```

[Ctrl＋S]を押して変更を保存します。

5.11　Visual Studio のメニューから[ビルド ＞ ソリューションのビルド]をクリックします。[出力]ウィンドウにビルド成功の表示がされます。

エラーが表示された場合は、表示が消えるまで修正を行ってください。

5.12　Visual Studio で Phoneword.Droid をスタートアッププロジェクトに設定し、▶ ボタンをクリックし、アプリケーションを起動します。

おめでとうございます！これで本講習はすべて終了です。Xamarin ネイティブで Android アプリの作成方法、Xamarin.Forms で Android／UWP／Windows Phone アプリの作成方法を学びました。iOS アプリのビルドには Mac が必要ですので、本講習では扱っておりませんが、Mac をお持ちの方は http://www.xlsoft.com/jp/products/xamarin/support.html の「初めての Xamarin.iOS アプリケーション開発 - 入門ガイド」（ http://www.xlsoft.com/jp/products/xamarin/ios_hello_world.html ）などを参考に是非トライしてみてください。