

Visual Studio を用いた iOS／ Android／UWP クロスプラットフ オームモバイルアプリ開発入門

エクセルソフト株式会社
ソフトウェア事業部
新規事業開発室室長
田淵義人
ytabuchi@xlsoft.com
080-7015-3586
Twitter: [@ytabuchi](#)

演習の目標

- Xamarin と Visual Studio を使用して iOS／Android／UWP 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 (Xamarin.iOS)

- 開発に必要な MacOS X、iOS SDK、Xcode、Xamarin の設定を確認
- 開発に必要な Windows の Xamarin Mac Agent の設定を確認
- Xamarin.iOS プロジェクトの作成、開発
- iOS Simulator へのビルド、デプロイ
- iOS Simulator でのデバッグ

演習の概要 (Xamarin.Android)

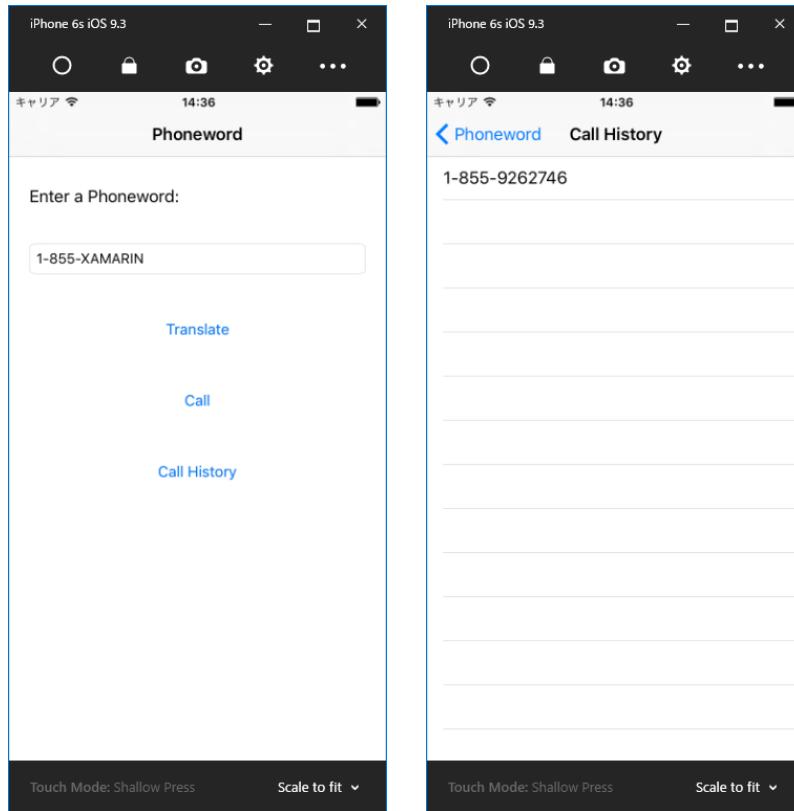
- 開発に必要な Android SDK、Java、Xamarin の設定を確認
- Xamarin.Android プロジェクトの作成、開発
- Android Emulator または実機へのビルド、デプロイ
- Android Emulator または実機でのデバッグ

演習の概要 (Xamarin.Forms)

- Xamarin.Forms (Portable) プロジェクトの作成、開発
- iOS／Android／UWP アプリのビルド、デプロイ、デバッグ

Xamarin.iOS で iOS アプリ開発

本章では、Visual Studio を使用した Xamarin.iOS 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.iOS 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。本章では、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、以下のようないい画面になります。



1 開発に必要な MacOS X、iOS SDK、Xcode、Xamarin、 Windows、Visual Studio の設定を確認

1.1 Visual Studio を使用して iOS アプリケーションを開発するには以下を推奨します。

- macOS X Yosemite(10.10)以上の macOS X
 - 最新の Xcode
 - 最新の iOS SDK
 - 最新の Xamarin
- Windows 10
 - Visual Studio 2017
 - 最新の Xamarin 拡張

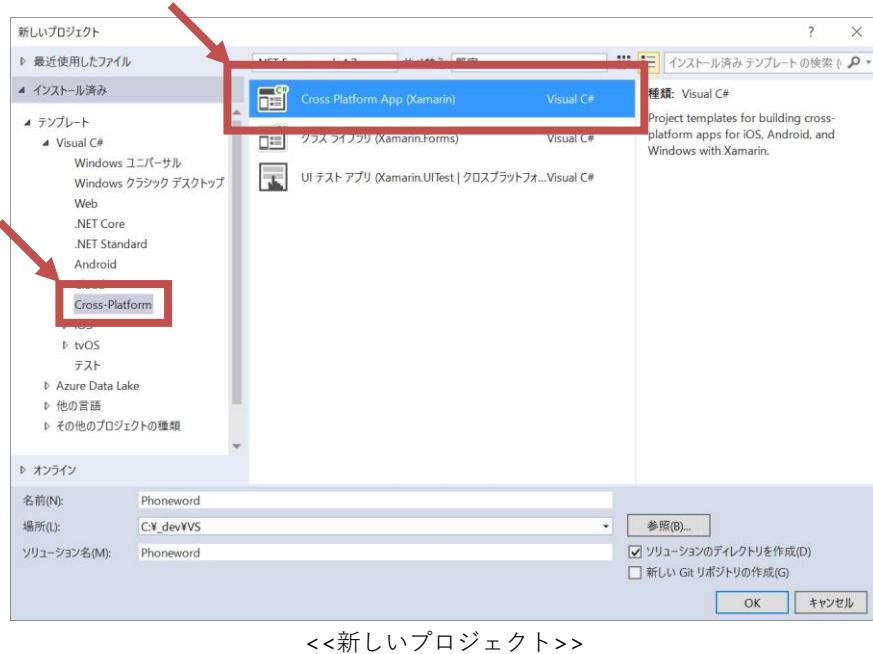
macOS X、Xcode、iOS SDK が最新になっていることを確認したら、Visual Studio for Mac をインストールします。

1.2 iOS Simulator または実機の準備

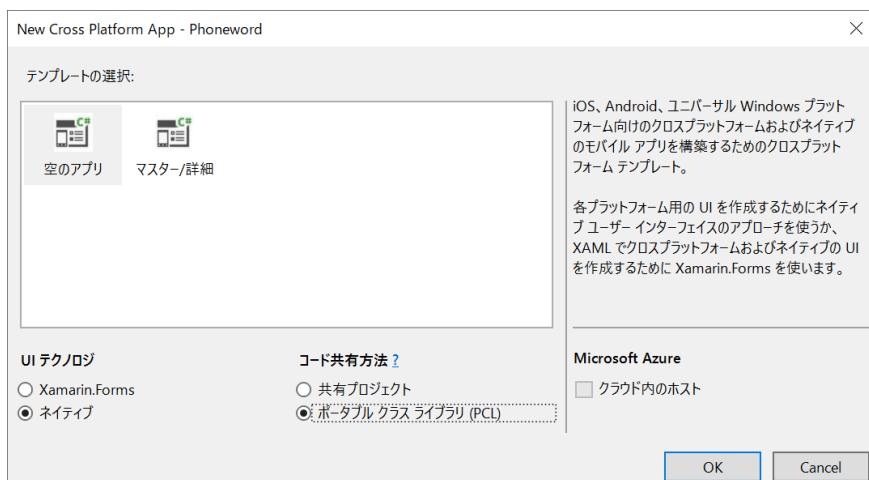
作成したアプリをデプロイする iOS Simulator または実機を用意します。実機へのデプロイは Apple Developer Program (<https://developer.apple.com/programs/jp/>) に加入するか、Free Provisioning を利用する方法があります。Xamarin での Free Provisioning の使用方法は「Xcode 7 と Xamarin Studio Starter で1円も払わずに自作 iOS アプリを実機確認する」(<http://ytabuchi.hatenablog.com/entry/2015/09/18/191258>) を参考にしてください。

2 Hello.iOS

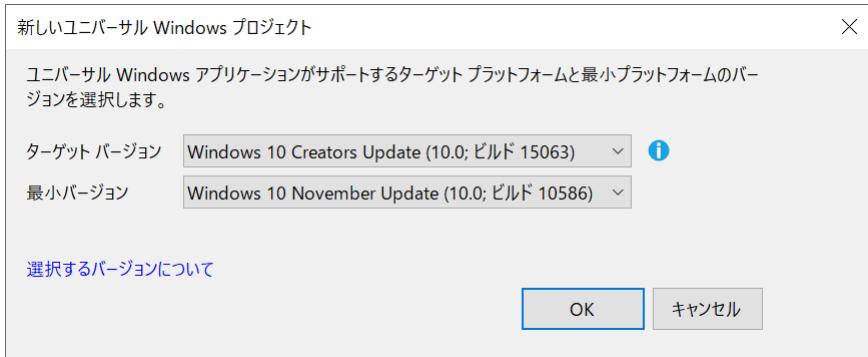
- 2.1 Visual Studio を起動し、メニューから [ファイル>新規作成>プロジェクト] をクリックして、新しいソリューションを作成します。
- 2.2 [新しいプロジェクト] 画面で、[Cross-Platform>Cross Platform App (Xamarin)] を選択し、名前を「Phoneword」と付けます。



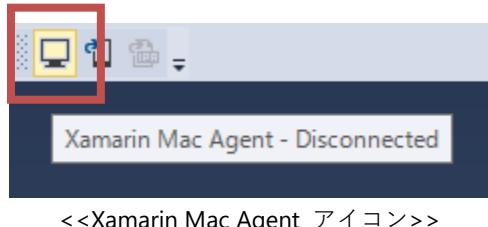
- 2.3 [空のアプリ]、[ネイティブ]、[ポータブルクラスライブラリ(PCL)] を選択し、[OK] をクリックします。



UWP バージョン確認のダイアログが出る場合はそのまま [OK] をクリックします。

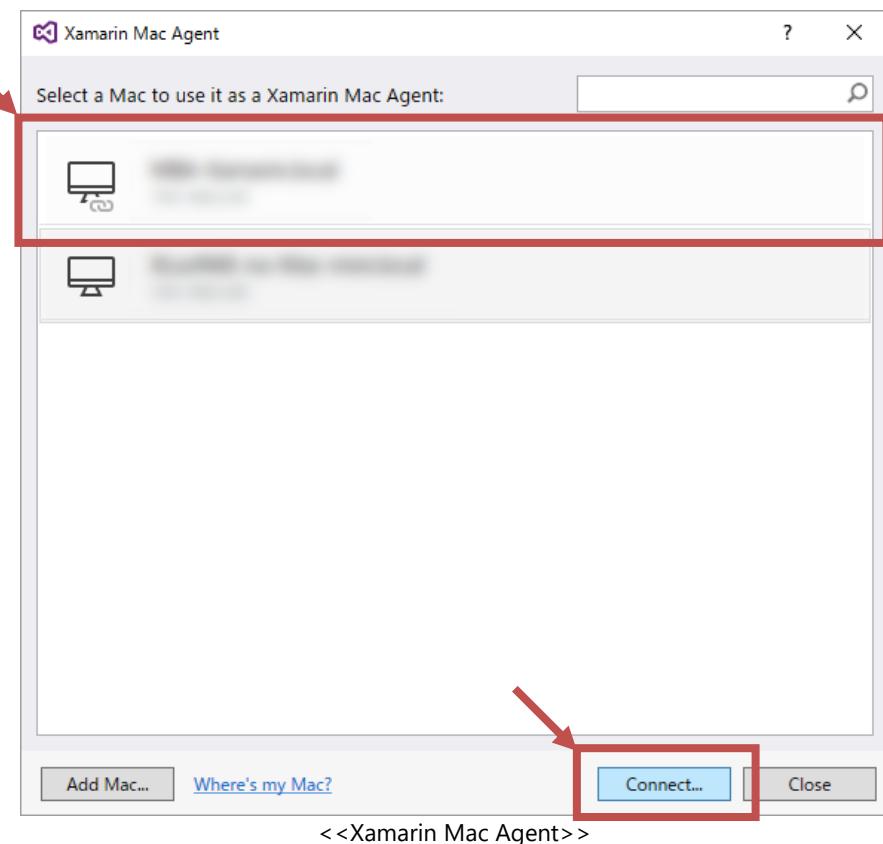


2.4 iOS の画面を編集するには Xamarin Mac Agent で Mac と接続する必要があります。アイコンをクリックします。

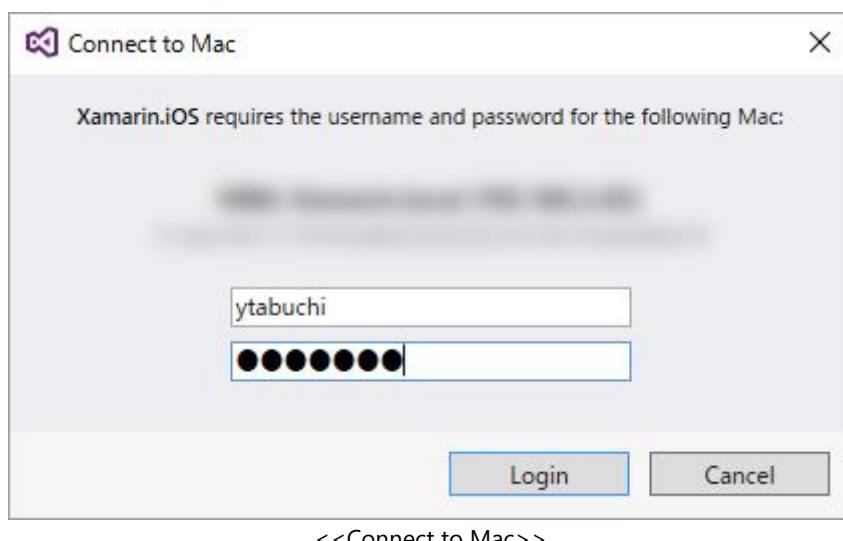


<<Xamarin Mac Agent アイコン>>

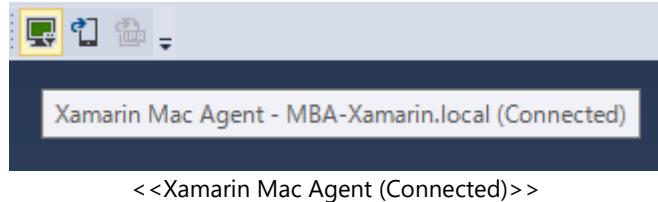
- 2.5 操作方法のダイアログが表示されるので [OK] をクリックします。
- 2.6 [Xamarin Mac Agent] ダイアログで接続する Mac を選択して、[Connect] ボタンをクリックします。



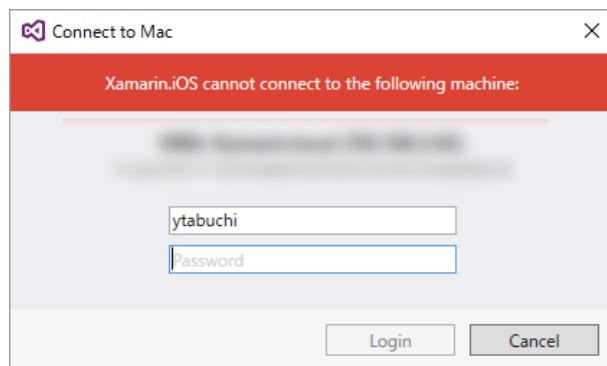
- 2.7 [Connect to Mac] ダイアログで Mac にログインしているユーザー名とパスワードを入力して [Login] ボタンをクリックします。



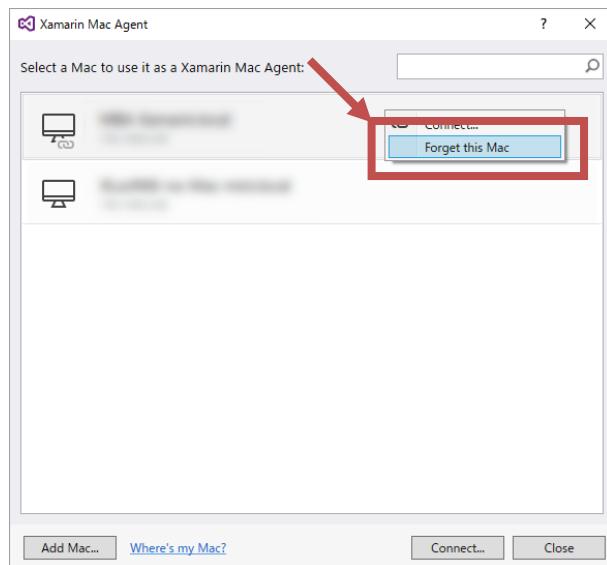
2.8 接続が完了すると、次のように緑色のアイコンに変わります。



接続が切れた場合は再度アイコンをクリックして接続します。その際に次のようにパスワードが弾かれことがあります。

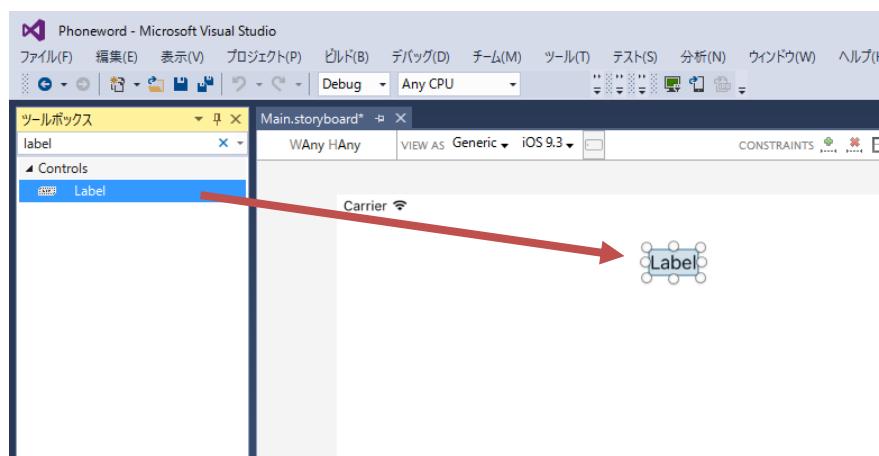


その場合は、該当マシンを右クリックして、[Forget this Mac] を選択し、再度接続してください。Mac が見つからない場合は、左下の [Add Mac] ボタンから該当 Mac の IP アドレスまたはマシン名を入力して接続を試みてください。

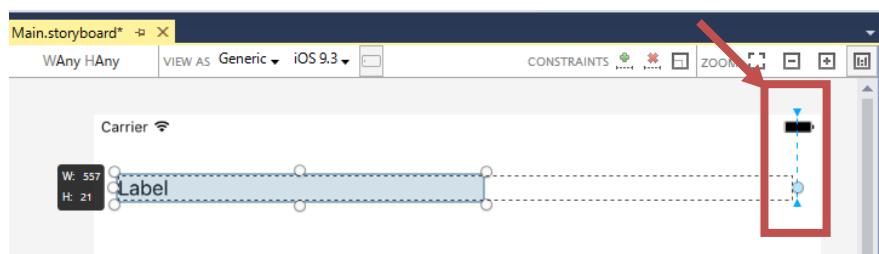


3 iOS Designer で Storyboard にコントロールを配置

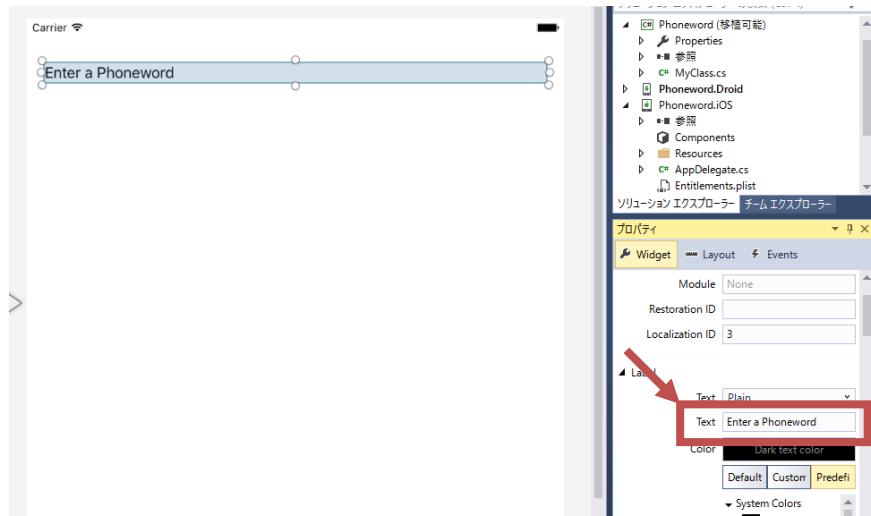
- 3.1 ソリューションエクスプローラーから [Phoneword.iOS] プロジェクト内の [Main.storyboard] をダブルクリックで開きます。iOS Designer が起動します。
- 3.2 画面中央の「Hello World, Click Me!」ボタンを削除します。
- 3.3 画面左側の [ツールボックス] の上部にある検索欄に「label」と入力し、デザイン画面（画面中央）に[Label]をドラッグします。



- 3.4 [Dragging Controls] (コントロールの囲いの○) のハンドルをつかみ、ラベルを広げます。Storyboard の端の方にドラッグすると、青い線が表示されるので参考にしてください。



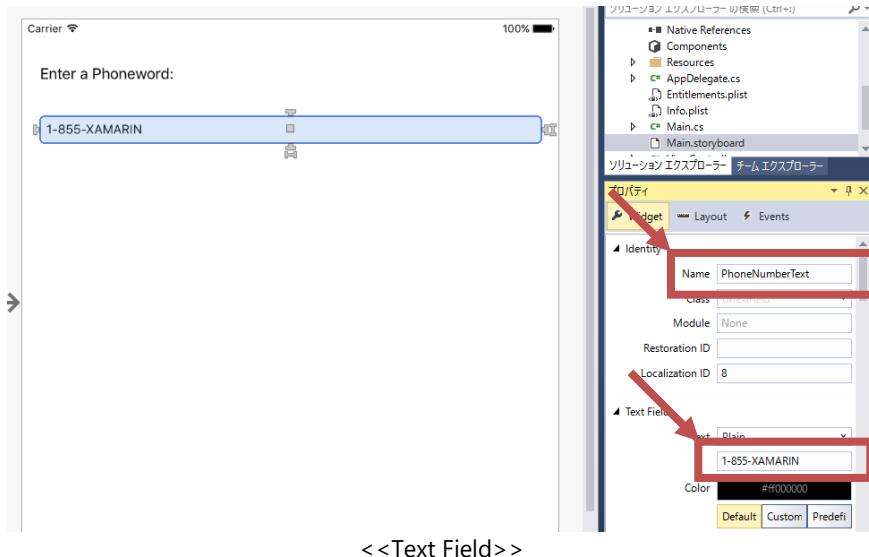
- 3.5 デザイン画面で [Label] を選択し、[プロパティ] ウィンドウを使用して [Label] の [Text] プロパティを「Enter a Phoneword:」に変更します。



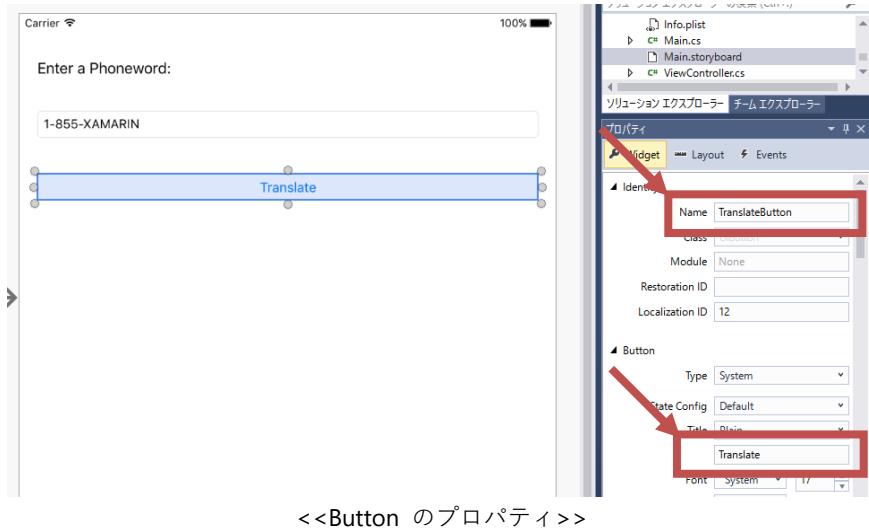
- 3.6 次に、[ツールボックス] で「text field」を検索し、[Text Field] をドラッグし、[Label] の下に [Text Field] を配置します。[Text Field] と [Label] の幅が同じになるように調整します。



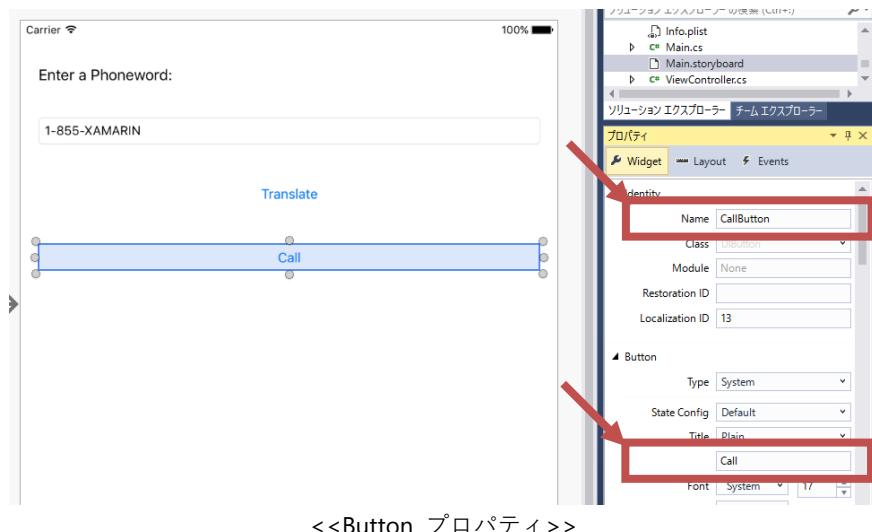
- 3.7 デザイン画面で選択した [Text Field] で、右下の [プロパティ] ウィンドウを使用して [Name] プロパティを「PhoneNumberText」に変更し、[Text Field] の文字列を「1-855-XAMARIN」に変更します。



- 3.8 次に、[ツールボックス] からデザイン画面に [Button] をドラッグし、[Text Field] の下に配置します。[Button] の幅も [Text Field] と [Label] と同じ幅になるように調整します。
- 3.9 デザイン画面で [Button] を選択し、[プロパティ] ウィンドウの [Identity] セクションの [Name] プロパティを「TranslateButton」に変更します。[Title] プロパティを「Translate」に変更します。



- 3.10 上記 2つのステップを繰り返し、デザイン画面の [ツールボックス] から [Button] をドラッグし、最初の [Button] の下に配置します。その [Button] の幅を最初の [Button] と同じ幅になるように調整します。
- 3.11 デザイン画面で選択した 2 番目の[Button]で、[プロパティ]パッドの [Identity]セクションにある[Name]プロパティを「CallButton」に変更します。[Title]プロパティを「Call」に変更します。

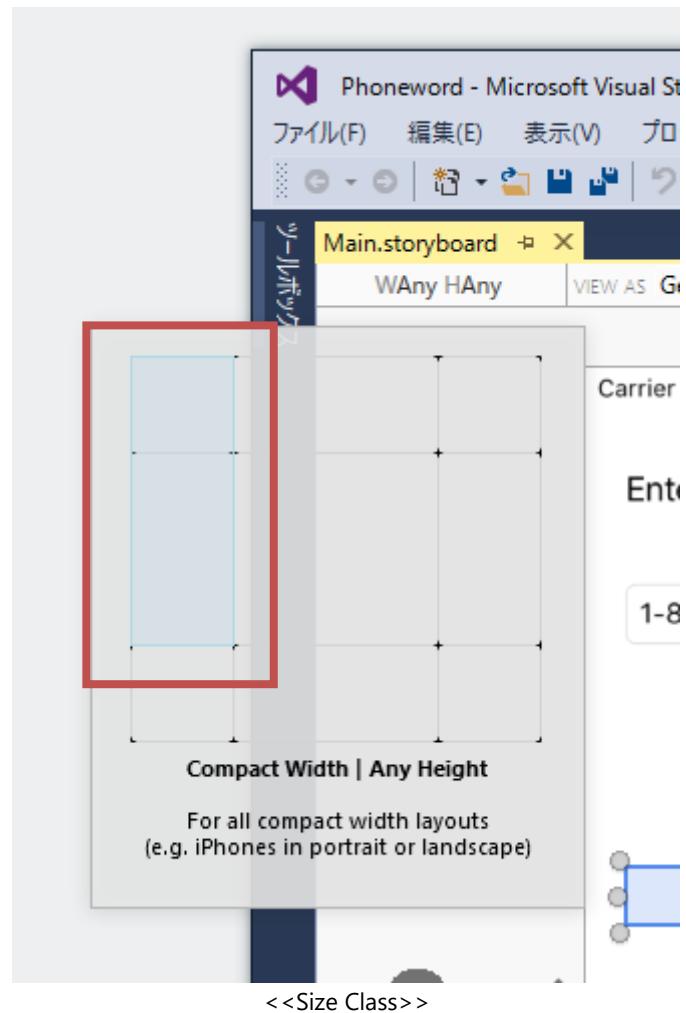


- 3.12 [ファイル>すべて保存]または[Ctrl + S]ボタンを押して作業内容を保存します。

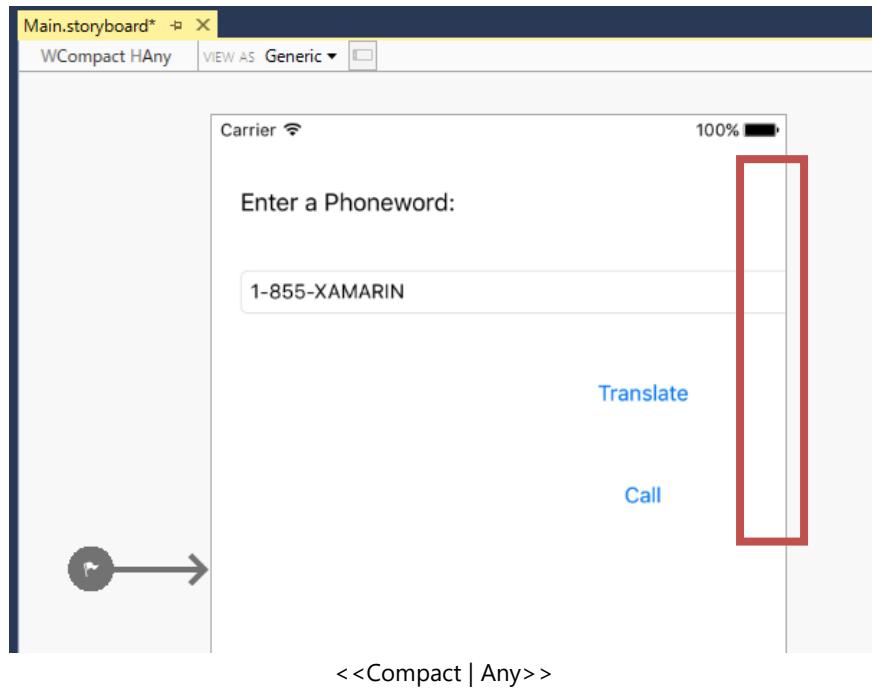
4 Constraints の設定

iOS は様々なデバイスの画面サイズに対応するために、Storyboard の各コントロールに Constraints (制約) を指定します。この章では Constraints の追加方法を学びます。

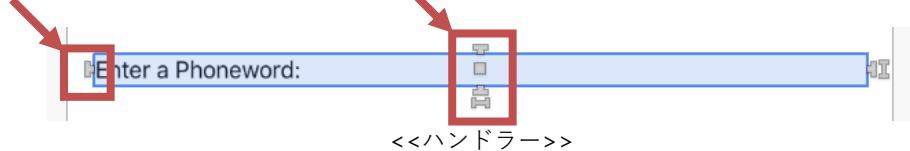
- 4.1 デザイン画面の左上の [Size Class] ボタンから [Compact Width | Any Height] をタップします。



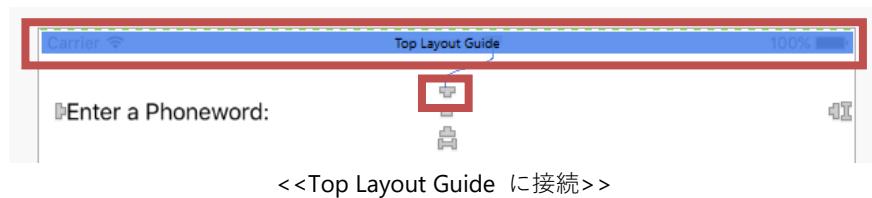
- 4.2 [Compact Width | Any Height] は iPhone 5 / 6 / 6 Plus などで使用されるレイアウトですが、作成したオブジェクトが隠れてしまっているのが分かります。



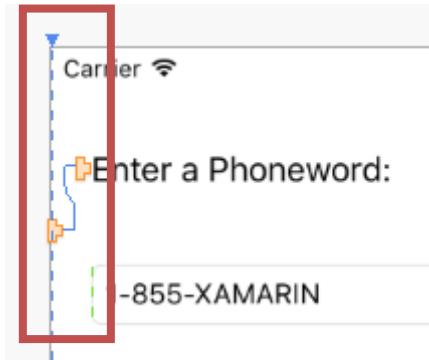
- 4.3 [Any | Any] のレイアウトに戻し、[Constraints]（制約）を追加することですべてのレイアウトに対応できるようにします。[Label] を選択して再度クリックすると、ハンドラーが [○] から [□] や [凸] などに変わります。これらのハンドラーを操作して、制約を作成します。



- 4.4 [Label] の上部のハンドラーをドラッグして、[View] の上部でドロップします。

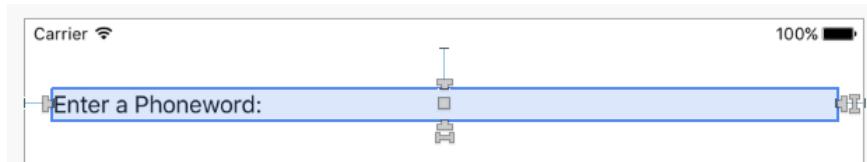


- 4.5 左側のハンドラーをドラッグして、[View] の左端でドロップします。[制約] を指定できる個所になると、青い点線が表示されるので活用してください。どうように右側のハンドラーも [制約] を追加します。

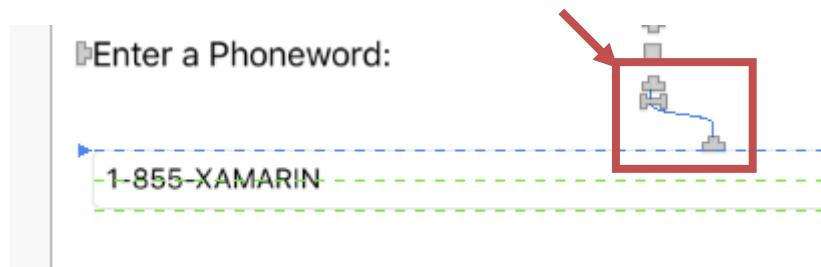


<<左の制約>>

- 4.6 オブジェクトの制約がすべて設定できると、選択時の色が青になります。

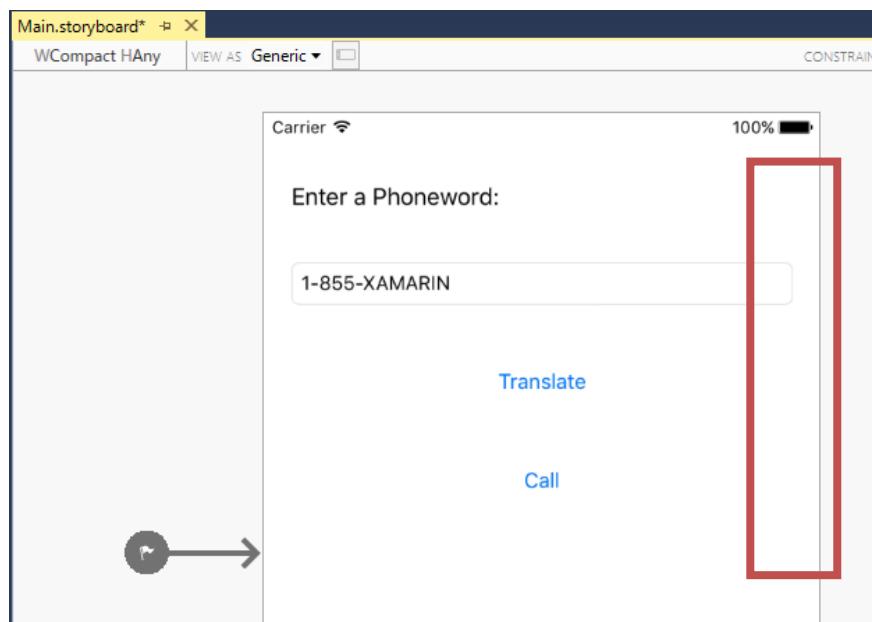


- 4.7 [Label] と [Text Field] など、オブジェクト同士の距離も [制約] を指定します。



<<オブジェクト同士の制約>>

- 4.8 同様に、すべてのオブジェクトに [制約] を指定してください。すべて指定すると、[Compact Width | Any Height] を表示しても各オブジェクトが画面内に収まるのが分かります。



5 コードの追加

Storyboard の操作方法が分かったところで、コードを記述していきましょう。コードの共通化についても学びます。

- 5.1 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] ウィンドウの [Phoneword (移植可能)] プロジェクトの [MyClass.cs] を右クリックして、[名前の変更] をクリックします。
- 5.2 「PhoneTranslator」に名前を変更して [Enter] キーを押すと、確認ダイアログが表示されますので、[はい] をクリックします。
- 5.3 ここで、新しい C# クラスを作成します。テンプレートのすべてのコードを削除し、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;
            else
                return null;
        }
    }
}
```

```

        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;
}
}
}

```

[PhoneTranslator.cs] ファイルを保存して閉じます。

- 5.4 [Phoneword (移植可能)] プロジェクトを右クリックして [ビルド] しておきましょう。
- 5.5 次に [ViewController] クラスにコードを追加して、ユーザーインターフェースを操作します。[ソリューション エクスプローラー] から [Phoneword.iOS] プロジェクトの [ViewController.cs] をダブルクリックして開きます。
- 5.6 Storyboard の編集で最初に削除したボタン用のコードが残っているので以下のコードを削除します。

```

Button.AccessibilityIdentifier = "myButton";
Button.TouchUpInside += delegate {
    var title = string.Format ("{0} clicks!", count++);
    ButtonSetTitle (title, UIControlState.Normal);
};

```

- 5.7 クラス変数 `int count = 1;` も削除します。
- 5.8 次に [TranslateButton] を操作します。[ViewController] クラスで、[ViewDidLoad] メソッドを見つけ、ボタンのコードを追加します。以下が、[ViewDidLoad()] の呼び出しです。

```

public override void ViewDidLoad ()
{
    base.ViewDidLoad ();
    // code goes here
}

```

5.9 [TranslateButton] と名前を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。以下のコードを [ViewDidLoad] の後に追加します。

```
string translatedNumber = "";
TranslateButton.TouchUpInside += (object sender, EventArgs e) =>
{
    // PhoneTranslator.cs を使用してテキストから電話番号に変換します
    translatedNumber = Core.PhonewordTranslator.ToNumber(PhoneNumberText.Text);
    // TextField がタップされたらキーボードを Dismiss します
    PhoneNumberText.ResignFirstResponder();

    if (translatedNumber == "")
    {
        CallButtonSetTitle("Call", UIControlState.Normal);
        CallButton.Enabled = false;
    }
    else
    {
        CallButtonSetTitle("Call " + translatedNumber, UIControlState.Normal);
        CallButton.Enabled = true;
    }
};
```

5.10 次に、[CallButton] と名前を付けた二番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。[TranslateButton] のコードの下に以下のコードを追加します

```
CallButton.TouchUpInside += (object sender, EventArgs e) =>
{
    // 標準の電話アプリを呼び出すために tel: のプリフィックスで URL ハンドラーを使用します
    var url = new NSUrl("tel:" + translatedNumber);
    // できない場合は UIAlertView を呼び出します。
    if (!UIApplication.SharedApplication.OpenUrl(url))
    {
        var av = new UIAlertView("Not supported",
            "Scheme 'tel:' is not supported on this device",
            null,
            "OK",
            null);
        av.Show();
    }
};
```

[NSURL] クラスが存在しないというエラーが表示されるので、[NSURL] 上で [.] キーを押し、[using Foundation] を選択して [using] を追加します。また `UIAlertView` は旧型式ですが、後で修正します。

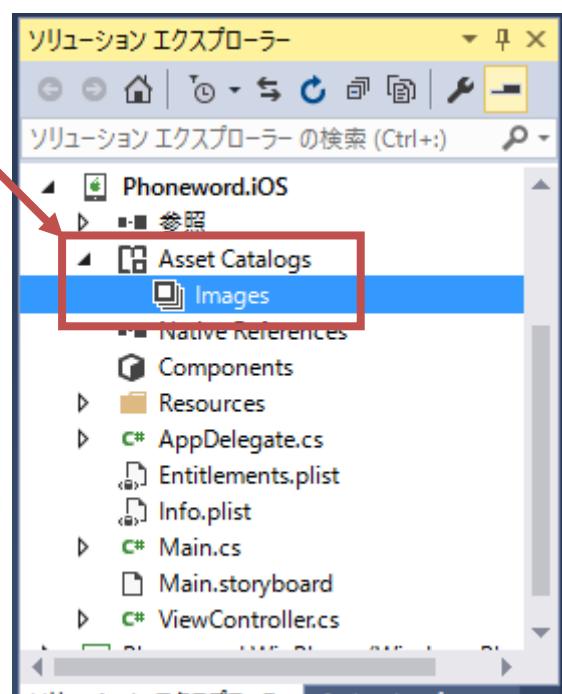


5.11 作業を保存し、[ビルド>Phoneword.iOS のビルド] を選択または [Shift + F6] キーを押して、アプリケーションをビルドします。アプリケーションをビルドすると、Visual Studio の左下に [ビルド正常終了] とメッセージが表示されます。

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。

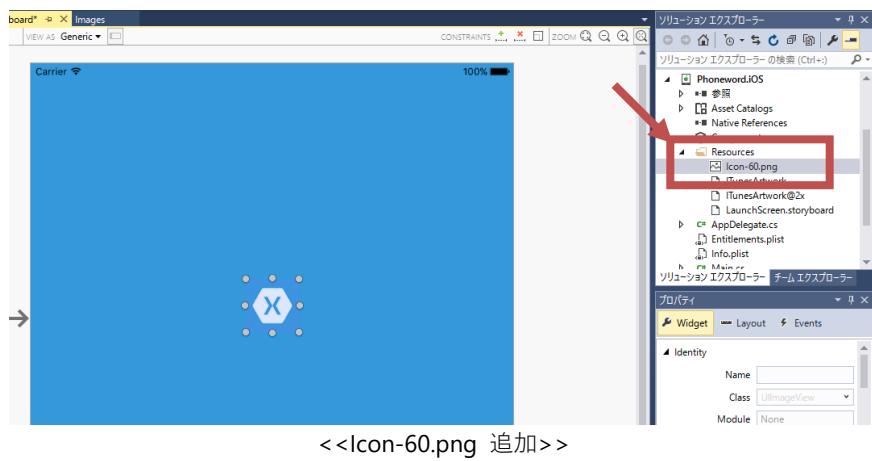
6 その他の設定と動作確認

- 6.1 これで、アプリケーションが動作します。[Info.plist] ファイルを開き、アプリケーションの情報を確認します。[Application name] が iOS 上で表示されるアプリ名となり、[バンドル識別子] が App Store に申請する際の識別子となります。[Application Name]を「Phoneword」に変更します。
- 6.2 アプリケーションのアイコンと起動イメージを確認しましょう。[ソリューション エクスプローラー] で [Phoneword.iOS] プロジェクトの [Asset Catalogs] 内の [Images] をダブルクリックして開きます。

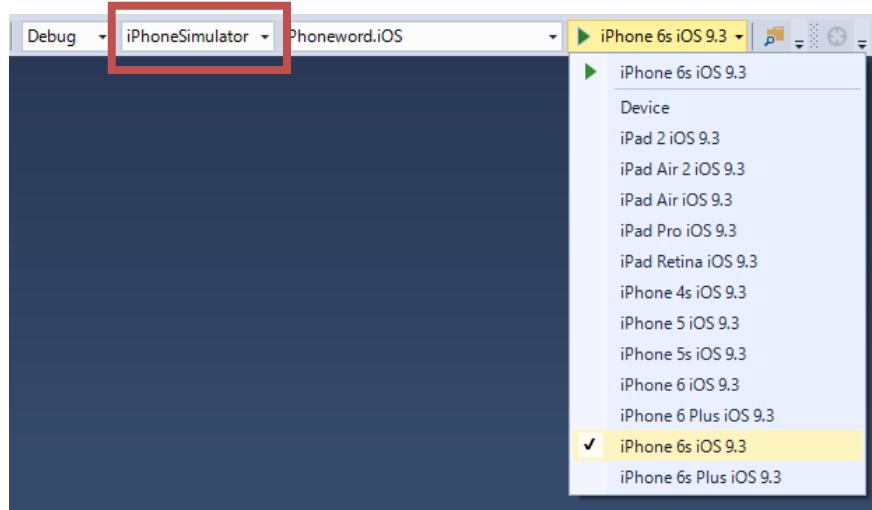


- 6.3 新規にアプリを作成する際はそれぞれのサイズに合わせたアイコンを用意して [AppIcons] に登録していきます。

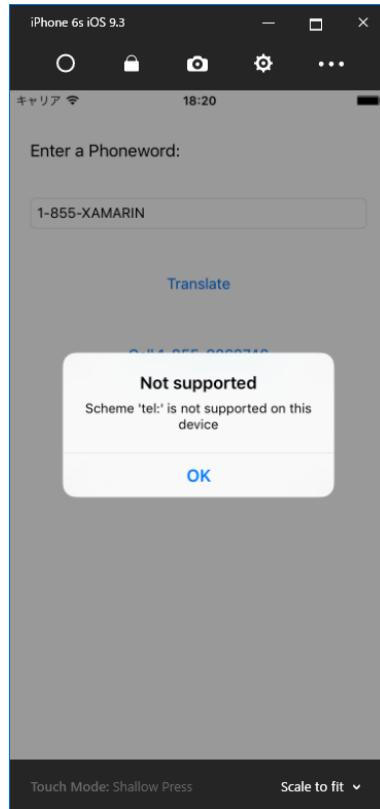
- 6.4 次に [LaunchScreen] の設定を行います。[Resources] フォルダ内の [LaunchScreen.storyboard] ファイルをダブルクリックして開きます。
- 6.5 中央の [UIImageView] に指定されている画像「Icon-60.png」を [Phoneword.iOS] プロジェクトの [Resources] フォルダにドラッグ & ドロップします。



- 6.6 最後に、iOS Simulator または実機でアプリケーションをテストします。Visual Studio の標準ルールバーで、[ソリューションプラットフォーム] ドロップダウンで [iPhoneSimulator] を選択し、次の [スタートアッププロジェクト] のドロップダウンで [Phoneword.iOS] を選択し、適切な対象デバイスを選択し、Start ボタン[▶]を押します。



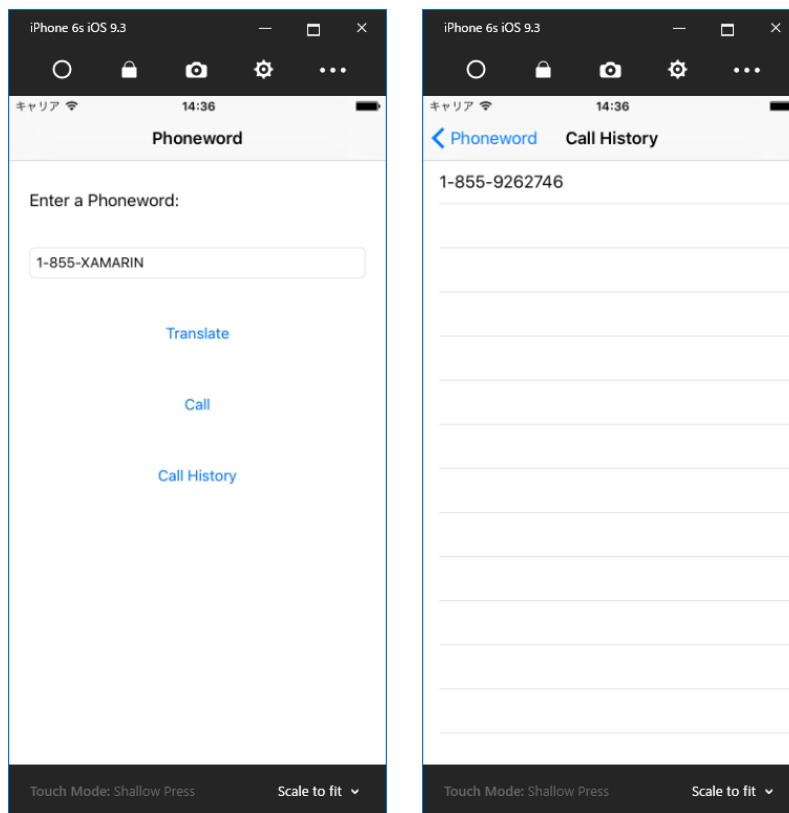
- 6.7 iOS Simulator または実機でアプリケーションが起動します。
- 6.8 iOS Simulator では電話の発信をサポートしていません。そのため「Call」ボタンをクリックした際にアラートダイアログを表示します。



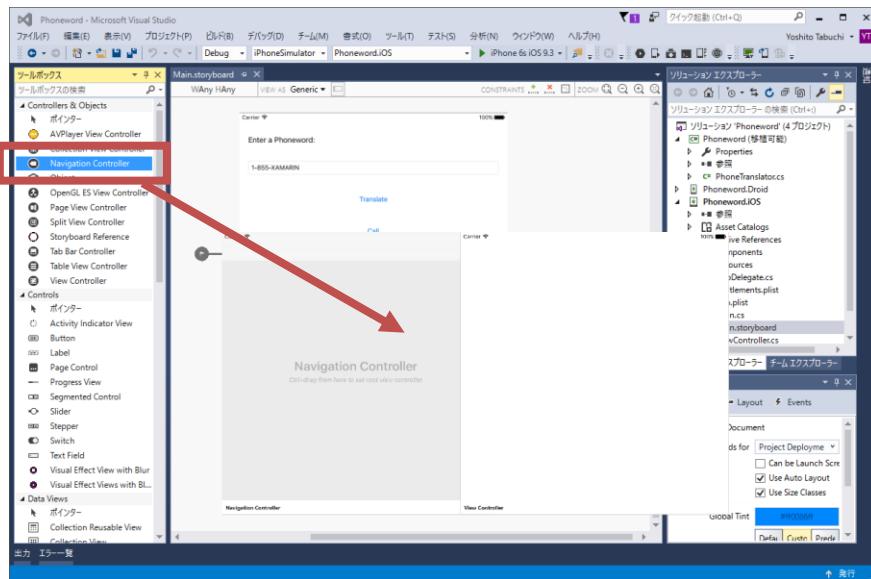
初めての Xamarin.iOS アプリケーションの完成です！次のステップ Hello, iOS Multiscreen Quickstart で、このガイドで習得したツールとスキルをさらに試しましょう。

7 Hello, iOS Multiscreen Quickstart

この章では、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。本章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。

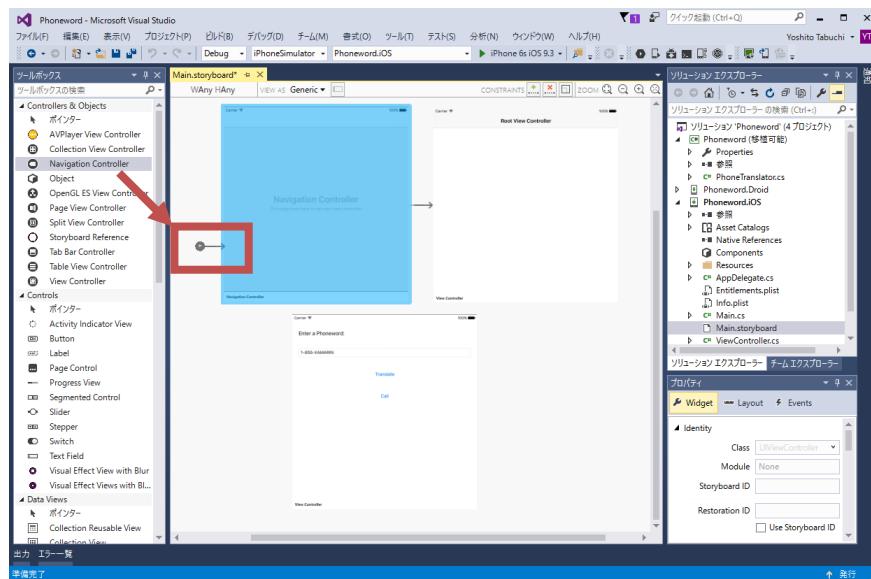


- 7.1 Visual Studio で Phoneword プロジェクトを開きます。
- 7.2 まずはユーザーインターフェースの編集から始めます。[ソリューション エクスプローラー] から [Main.storyboard] ファイルを開きます。
- 7.3 ツールボックスから、デザイン画面に [Navigation Controller] をドラッグ & ドロップします



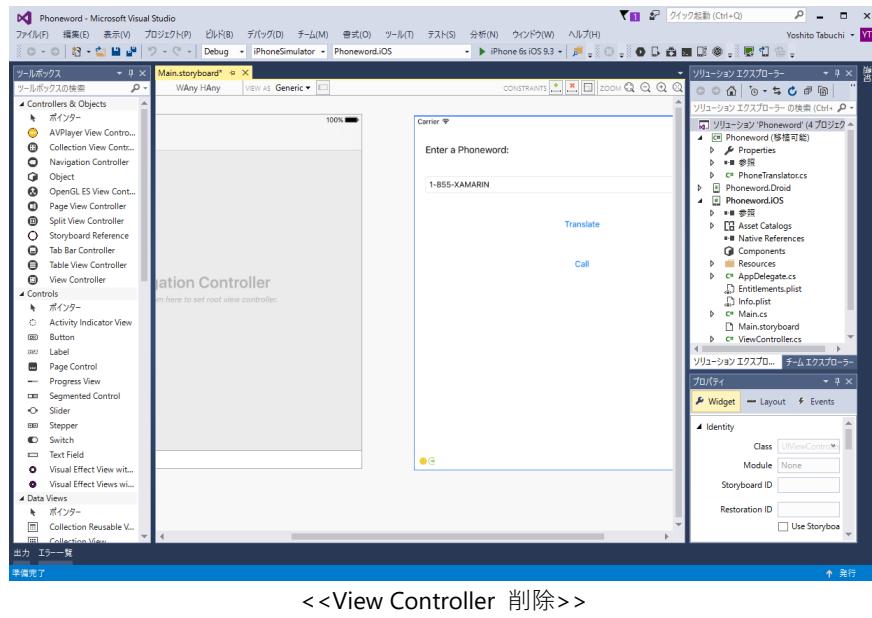
<< Navigation Controller 追加>>

- 7.4 [Sourceless Segue] (Phoneword 画面の左にある灰色の矢印) を [Phoneword] 画面から [Navigation Controller] へドラッグし、アプリケーションのスタートポイントを変更します。



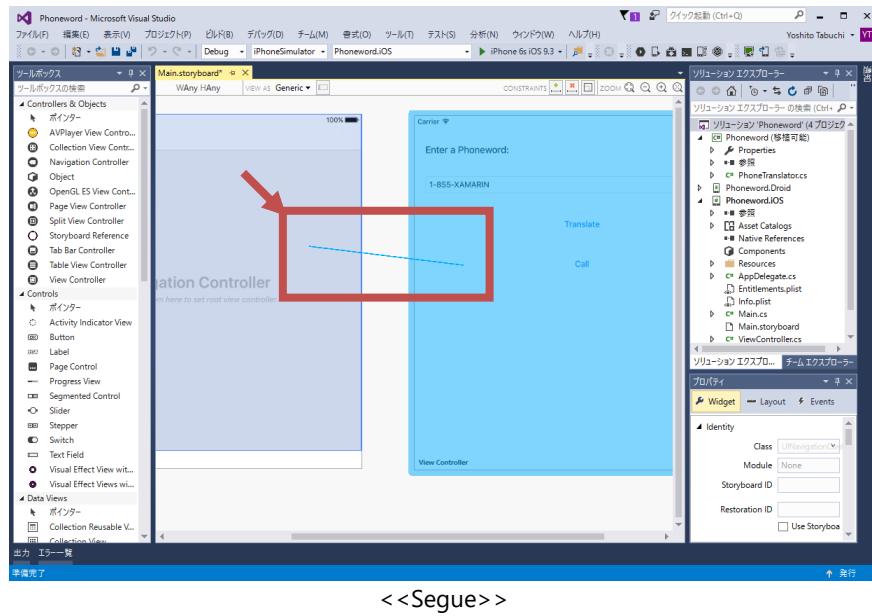
<< スタートポイント変更 >>

7.5 [Root View Controller] 下部の白いバーをクリックして選択し、[Delete] キーを押してデザイン画面から削除します。その後、[Navigation Controller] を [Phoneword] 画面の横に移動させます



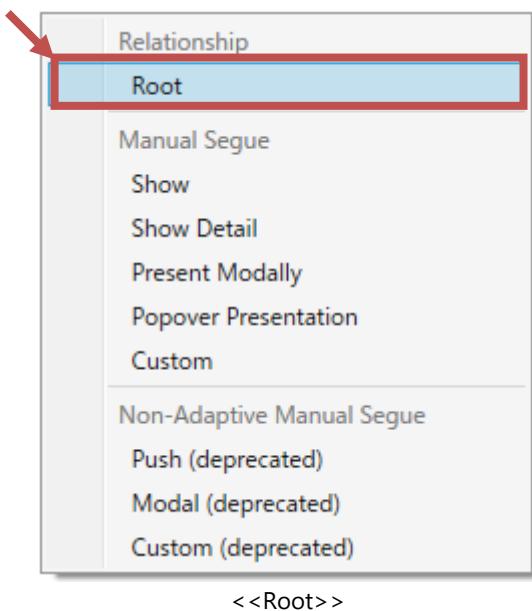
<<View Controller 削除>>

7.6 [Navigation Controller] の [Root View Controller] として [ViewController] をセットします。[Ctrl] キーを押し、[Navigation Controller] 内部をクリックします。青い線が表示されるので、[Ctrl] キーを押したまま [Navigation Controller] から [Phoneword] 画面までドラッグします。



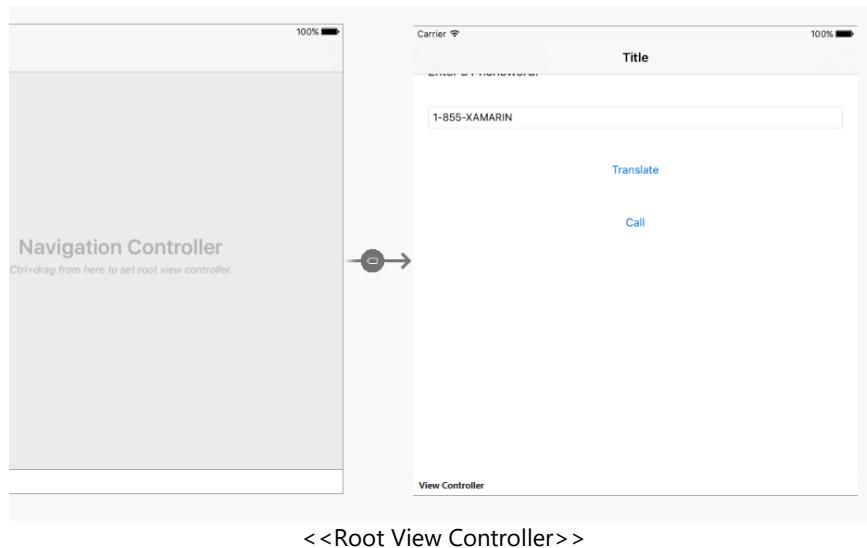
<<Segue>>

7.7 ポップオーバーから、[Relationship]の[Root]をクリックします。



<<Root>>

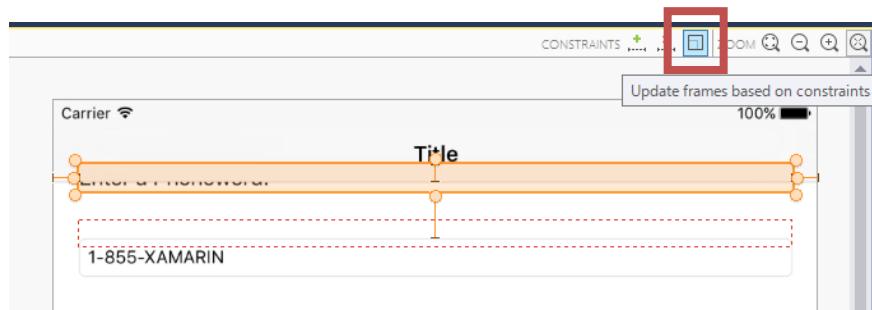
7.8 これで [ViewController] が [Navigation Controller] の [Root View Controller] になりました。



<<Root View Controller>>

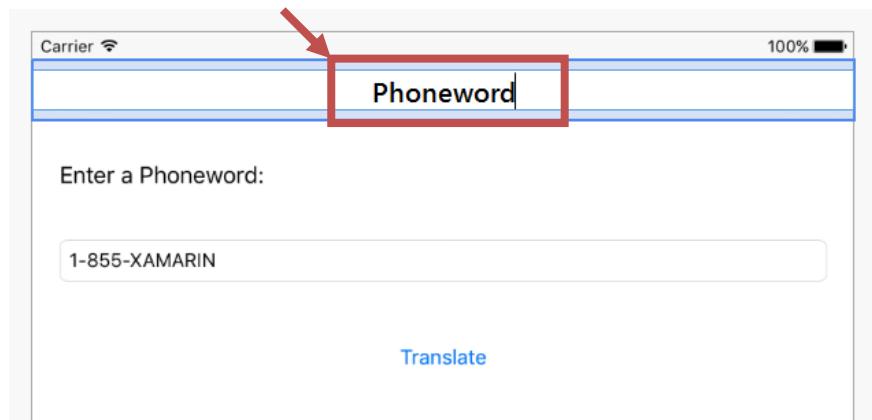
上記のようにオブジェクトが隠れてしまう場合は、[制約]が[Navigation Controller]に合わせて下がってきているため、[Update frames based on constraints] ボタンでオブジェクトの場所を自動調整します。





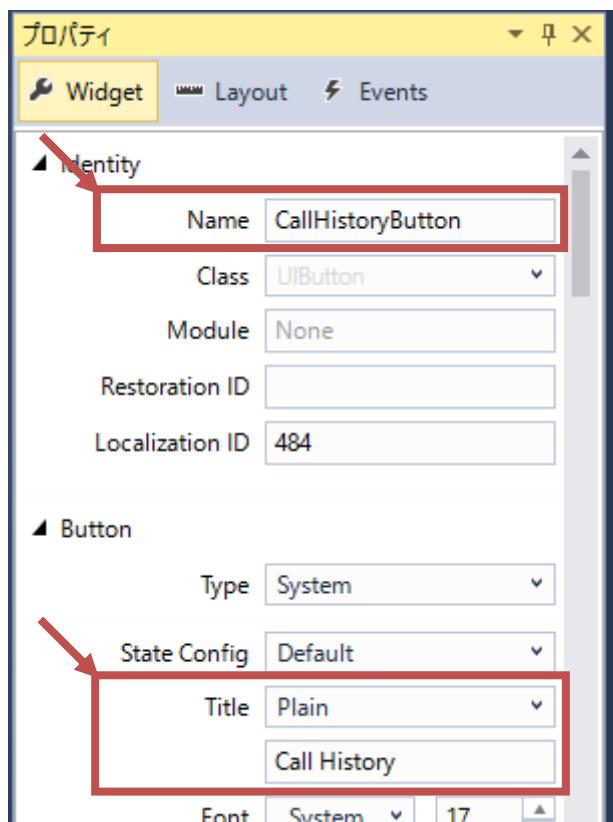
<<Update frame based on constraints>>

- 7.9 [Phoneword] 画面のタイトルバーをダブルクリックし、「Title」の文字列を「Phoneword」に変更します。

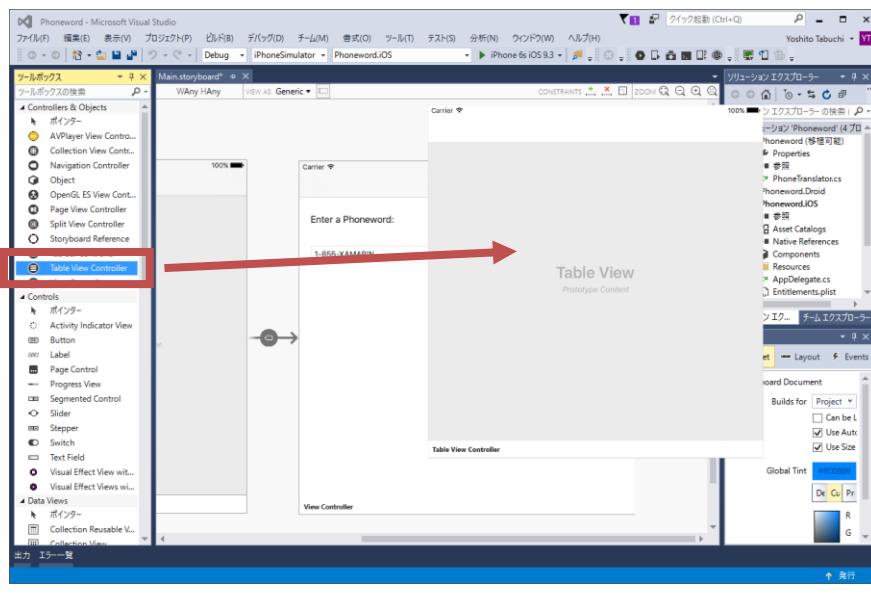


<<タイトル変更>>

- 7.10 ツールボックスから [Button] をドラッグし、[Call] ボタンの下に配置します。ハンドルをドラッグして、新しいボタンを [Call] ボタンと同じ幅に合わせます。
- 7.11 [プロパティ] ウィンドウで、ボタンの名前を [CallHistoryButton] に変更し、タイトルを「Call History」に変更します。

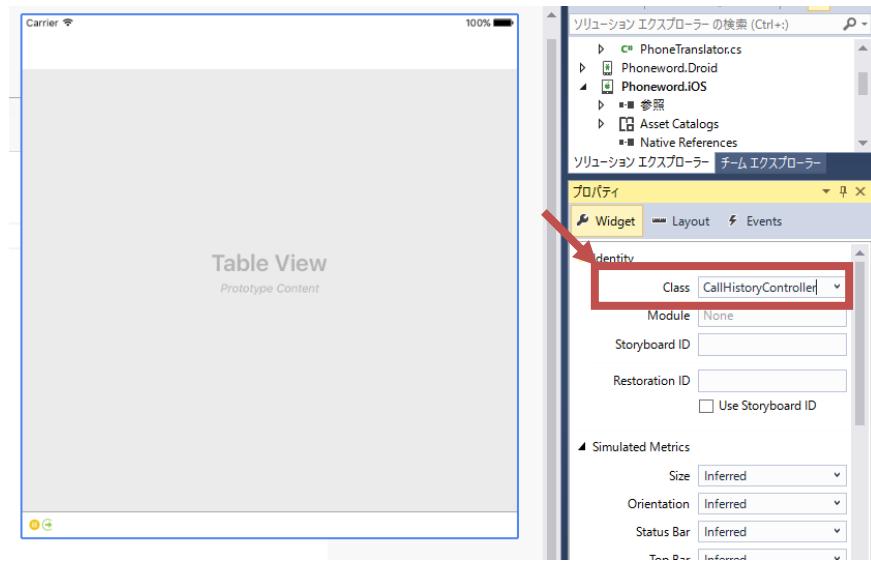


7.12 [Call History] 画面を作成します。[ツールボックス] からデザイン画面に [Table View Controller] をドラッグします。



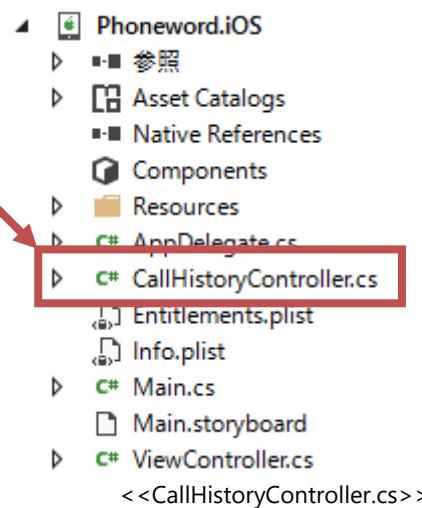
<<Table View Controller 追加>>

7.13 [Table View Controller] の下の白いバーをクリックします。[プロパティ] ウィンドウで [Table View Controller] のクラスを「CallHistoryController」に変更し、[Enter]キーを押します。



<<クラス追加>>

iOS Designer は、[CallHistoryController] と呼ばれるカスタムクラスを生成し、Content View のオブジェクトを管理します。[CallHistoryController.cs] ファイルが [ソリューションエクスプローラー] に表示されます。



7.14 [CallHistoryController.cs] ファイルをダブルクリックして開き、コンテンツを下記のコードに置き換えます。

```

using System;
using System.Collections.Generic;
using Foundation;
using UIKit;

namespace Phoneword.iOS
{
    public partial class CallHistoryController : UITableViewController
    {
        public List<string> PhoneNumbers { get; set; }
        static NSString callHistoryCellId = new NSString("CallHistoryCell");
        public CallHistoryController(IntPtr handle) : base(handle)
        {
            TableView.RegisterClassForCellReuse(typeof(UITableViewCell), callHistoryCellId);
            TableView.Source = new CallHistoryDataSource(this);
            PhoneNumbers = new List<string>();
        }
        class CallHistoryDataSource : UITableViewSource
        {
            CallHistoryController controller;
            public CallHistoryDataSource(CallHistoryController controller)
            {
                this.controller = controller;
            }
            // テーブルの各セクションの行数を返します
            public override nint RowsInSection(UITableView tableView, nint section)
            {
                return controller.PhoneNumbers.Count;
            }
            // NSIndexPath の Row プロパティで指定された行のテーブルセルを返します
            // このメソッドは、表の各行を挿入するために複数回呼び出されます
            // このメソッドは自動的に画面外にスクロールした Cell を使用または必要に応じて新しいものを作成
            // します
            public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)

```

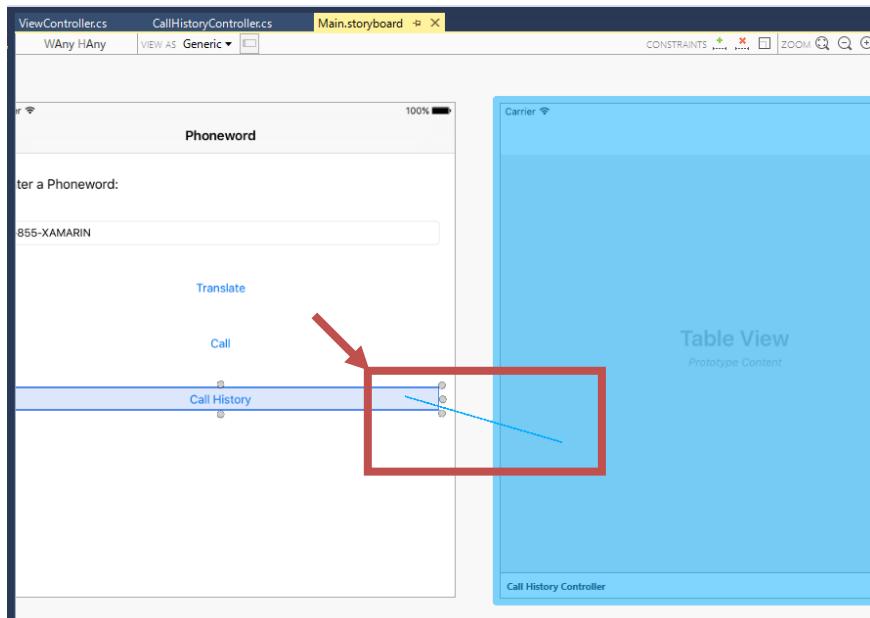
```

    {
        var cell = tableView.DequeueReusableCell(CallHistoryController.callHistoryCellId);
        int row = indexPath.Row;
        cell.TextLabel.Text = controller.PhoneNumbers[row];
        return cell;
    }
}
}
}

```

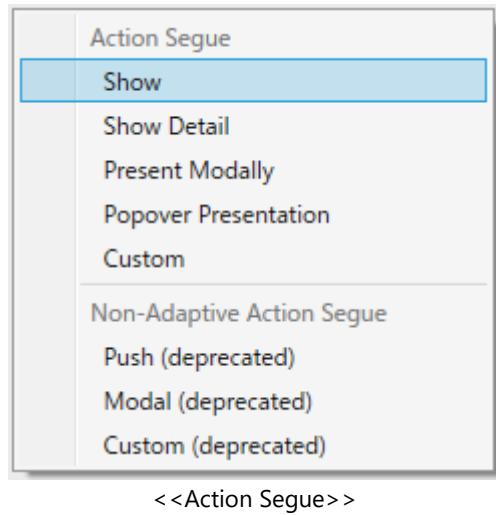
アプリケーションを保存し、エラーがないかビルドを実行して確認します。

7.15 [Phoneword] 画面と [Call History] 画面の切り替え (Segue) を作成します。[Phoneword] 画面で [Call History] ボタンを選択し、ボタンから [Call History] 画面に [Ctrl] を押しながらドラッグします。

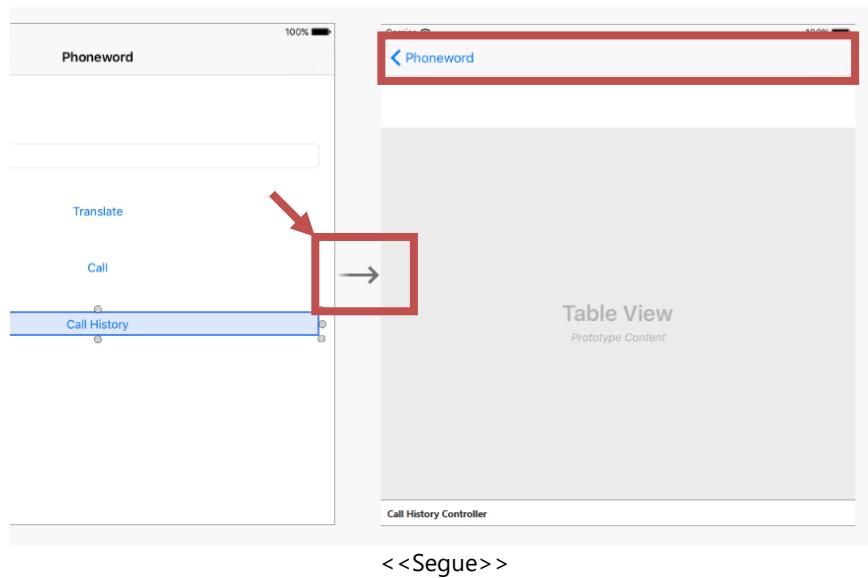


<<Show Segue 追加>>

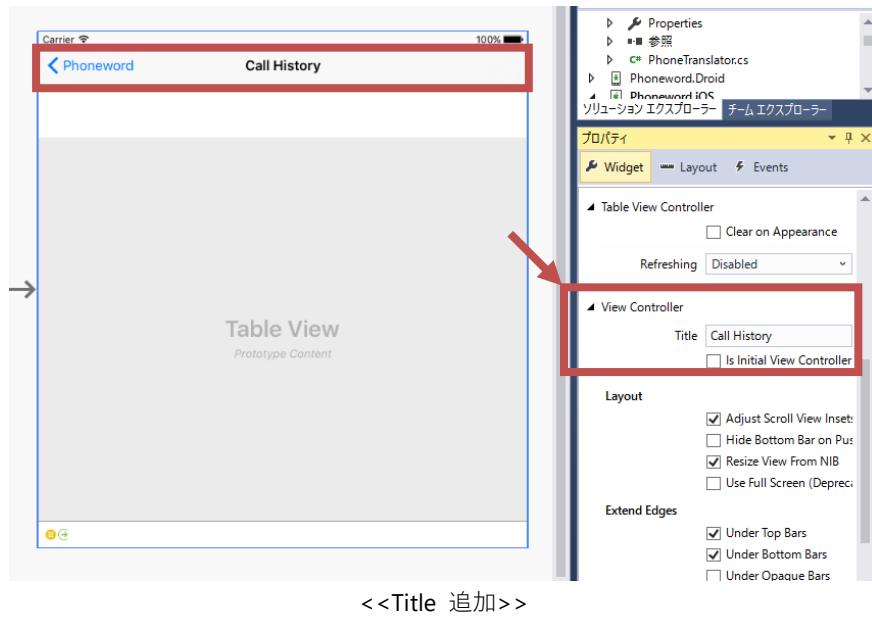
7.16 [Action Segue] のポップアップから [Show] を選択します。



7.17 iOS Designer が 2 つの画面間に [Segue] を追加します。[Navigation Controller] を経由しているので、[Call History Controller] の画面上部に [Navigation Bar] も追加されます。



7.18 画面下の白いバーを選択して [Table View Controller] にタイトルを追加します。[プロパティ] ウィンドウの [ViewController] の [Title] を「Call History」に変更します。



<<Title 追加>>

7.19 もしアプリケーションを今実行した場合、[Call History] ボタンで [Call History] 画面を表示できますが、電話番号の履歴を保持するコードが含まれていないため [Table View] には何も表示されません。これから [ViewController.cs] にその機能を追加していきます。リストをサポートするには、[System.Collections.Generic] を [ViewController] の上位で using 宣言に追加します。

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

7.20 下記のコードを使って [ViewController] クラスを修正します。

[translatedNumber] も [ViewDidLoad] メソッド内からクラス変数へ移動します。太字が修正分です。

```
namespace Phoneword.iOS
{
    public partial class ViewController : UIViewController
    {
        // translatedNumber を ViewDidLoad()から移動します
        string translatedNumber = "";
        public List<string> PhoneNumbers { get; set; }

        public ViewController (IntPtr handle) : base (handle)
```

```

    {
        // Call History 画面用に電話番号の List を初期化します
        PhoneNumbers = new List<string> ();
    }
    // ViewDidLoad, etc...
}

```

7.21 [CollButton] のコードを編集して、[PhoneNumbers.Add
(translatedNumber)] を呼ぶことで電話を掛けた番号を電話番号のリストに追加します。コード全体は下記のようになります。

```

CallButton.TouchUpInside += (object sender, EventArgs e) => {
    // 変換した電話番号を PhoneNumbers に追加します
    PhoneNumbers.Add (translatedNumber);
    // 標準の電話アプリを呼び出すために tel: のプリフィックスで URL ハンドラーを使用します
    var url = new NSUrl ("tel:" + translatedNumber);
    // できない場合は UIAlertView を呼び出します
    if (!UIApplication.SharedApplication.OpenUrl (url)) {
        var alert = UIAlertController.Create ("Not supported", "Scheme 'tel:' is not
supported on this device", UIAlertControllerStyle.Alert);
        alert.AddAction (UIAlertAction.Create ("Ok", UIAlertActionStyle.Default,
null));
        PresentViewController (alert, true, null);
    }
};

```

7.22 最後に、下記のメソッドを [ViewController] クラスに追加します。下記のコードを [ViewDidLoad] メソッドより下方に配置してください。

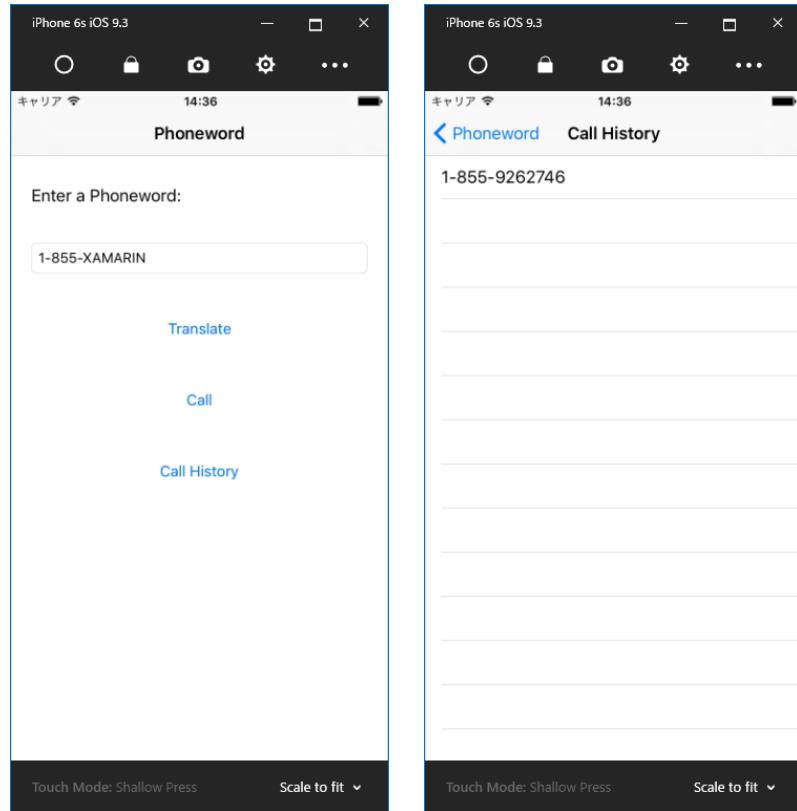
```

public override void PrepareForSegue(UIStoryboardSegue segue, NSObject sender)
{
    base.PrepareForSegue(segue, sender);
    // set the View Controller that's powering the screen we're
    // transitioning to
    var callHistoryController = segue.DestinationViewController as CallHistoryController;
    //set the Table View Controller's list of phone numbers to the
    // list of dialed phone numbers
    if (callHistoryController != null)
    {
        callHistoryController.PhoneNumbers = PhoneNumbers;
    }
}

```

プロジェクトを保存し、アプリケーションをビルドしてエラーがないか確認します。

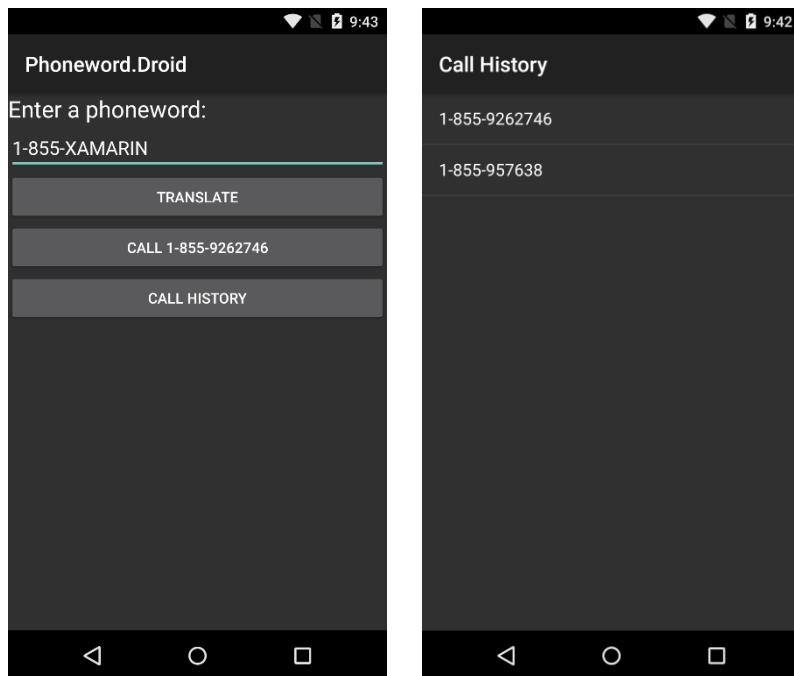
7.23 Start ボタンを押して iOS Simulator 上でアプリケーションを実行します。



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

Xamarin.Android で Android アプリ開発

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学びます。Xamarin.Android アプリケーションのビルト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。本章では、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 推奨システム

- Windows 10
- Visual Studio 2017
- Java SDK 1.8
- Android SDK

基本的にはすべての環境を最新にすることをお勧めします。Windows 10 は最新のバージョンを使用します。Visual Studio は 2017 を使用します。Java SDK は Android API 24 以降で Java 8 が必須となるため、Xamarin の開発環境でも 1.8 を使用します。Android SDK は Visual Studio のインストーラーでインストールしても構いませんし、独自にインストールしてもかまいませんが、Xamarin が対応している最新バージョンまでアップデートしてください。

1.2 実機またはエミュレーターの準備

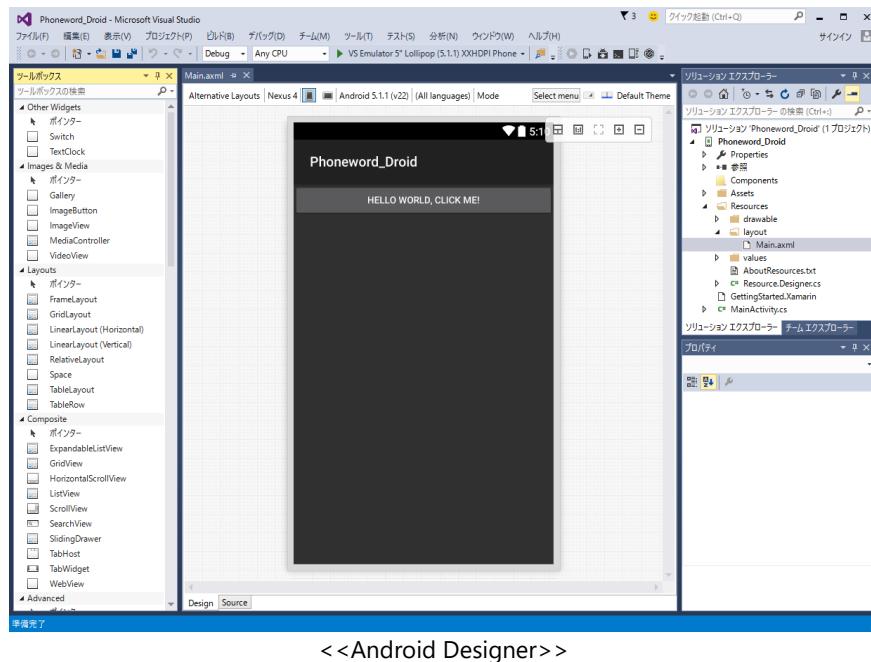
Android アプリを開発する際には実機を強くお勧めします。実機が無い場合のみ、エミュレーターを使用してください。Android のエミュレーターは Android SDK に付属するハードウェア アクセラレーション (Intel HAXM) を使用した x86 ベースの Google エミュレーターをお勧めします。ハードウェアアクセラレーションの設定方法に関しては、Accelerating Android Emulators ガイド（http://developer.xamarin.com/guides/android/getting_started/installation/accelerating_android_emulators）をご参照ください。

2 Hello, Android

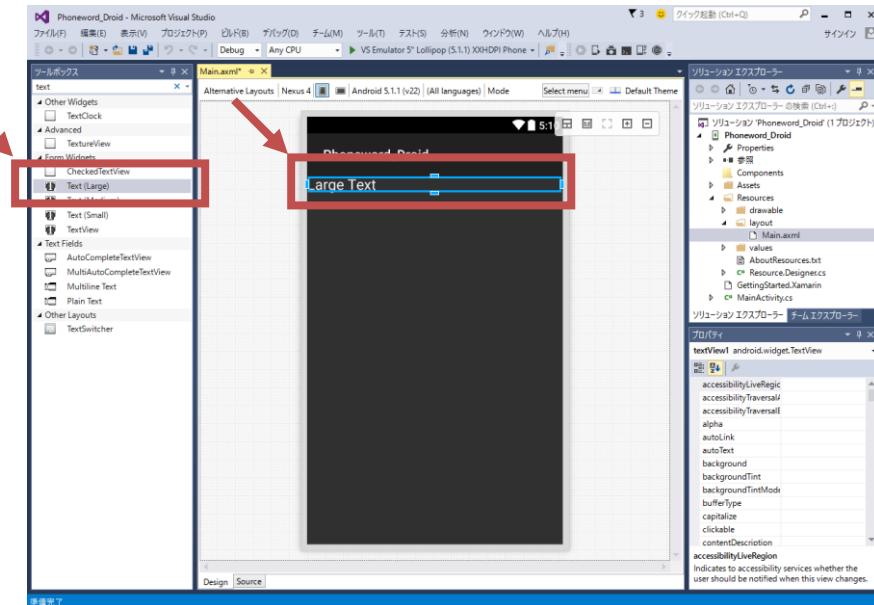
- 2.1 先ほど作成した [Phoneword] プロジェクトを使用します。
- 2.2 Visual Studio を起動し、[スタートページ>プロジェクトを開く] をクリックして、[Phoneword.sln] を開きます。

3 Android Designer で layout にコントロールを追加

- 3.1 次に、ソリューションエクスプローラーから [Phoneword.Android] 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。

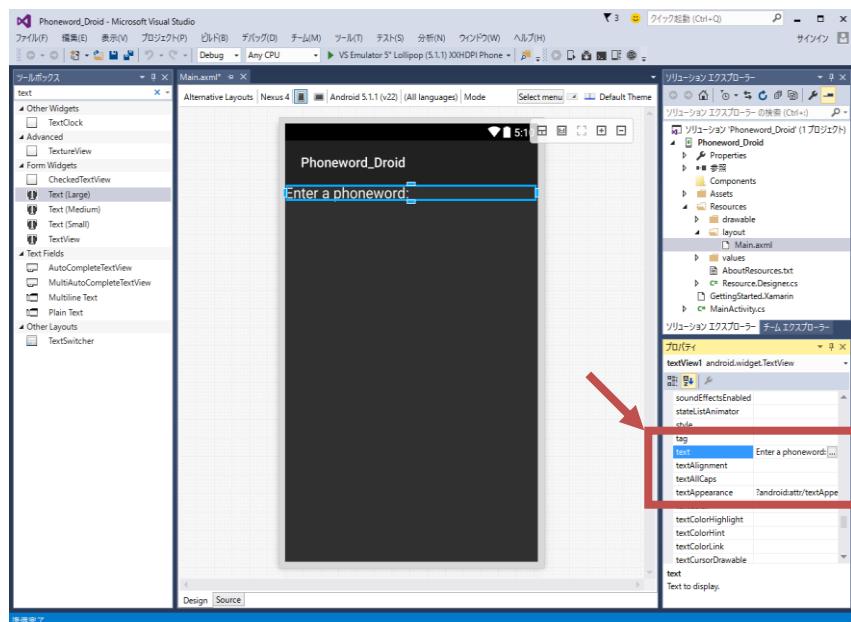


3.2 デザイン画面に標準で配置されている [HELLO WORLD, CLICK ME!] と書いてある [Button] を削除します。画面左側の [ツールボックス] の上部にある検索欄に「text」と入力し、[Text (Large)] を選択し、デザイン画面にドラッグ & ドロップして配置します。



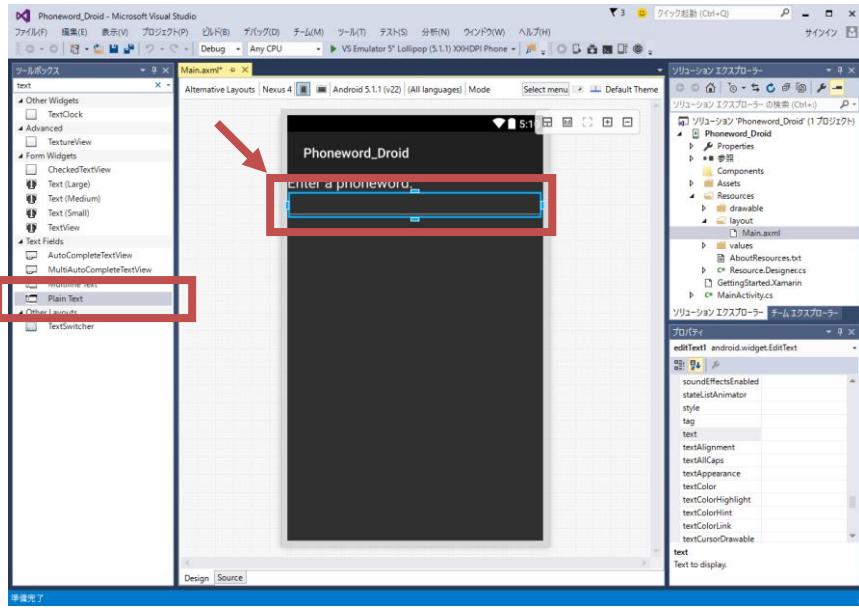
<<Text (Large) をドラッグ>>

3.3 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。



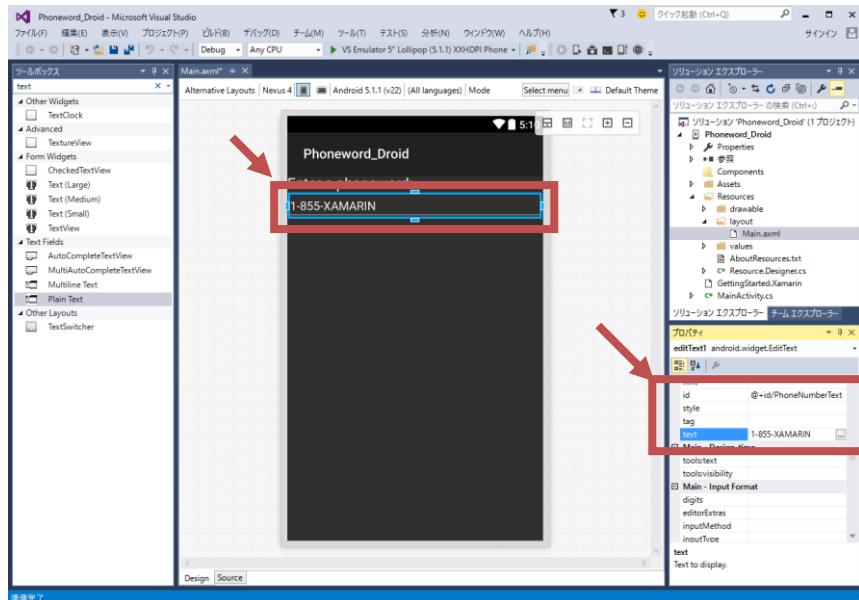
<<プロパティ変更>>

- 3.4 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。



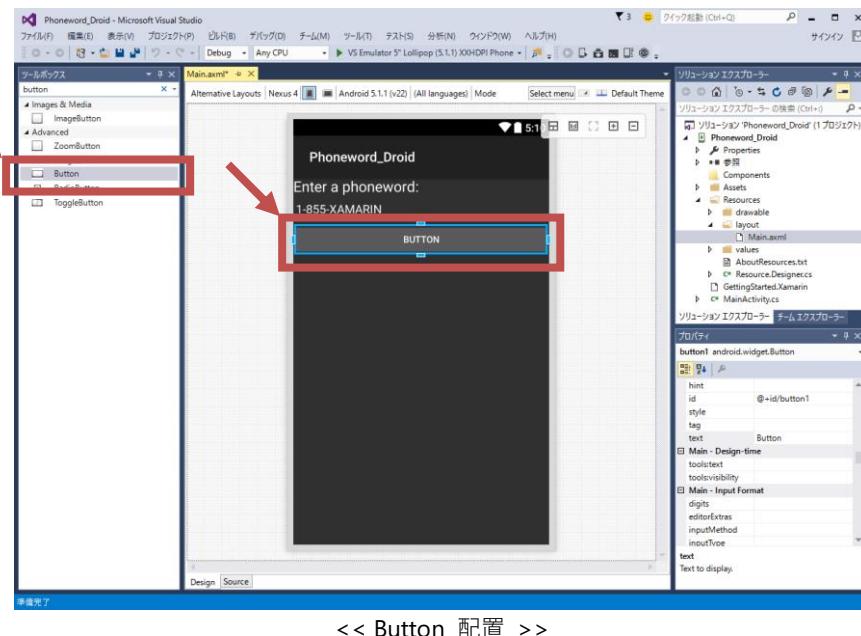
<< Plain Text 配置 >>

- 3.5 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。

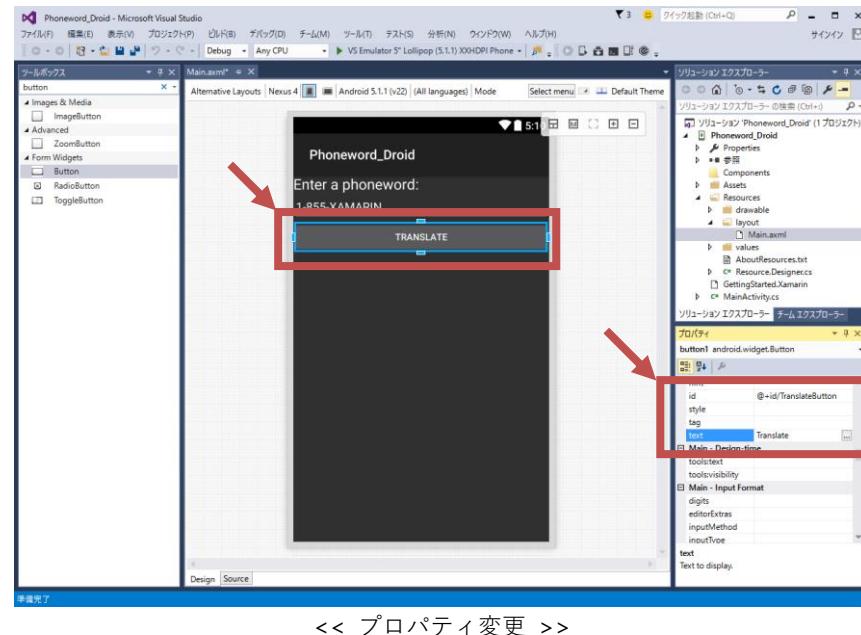


<< プロパティ変更 >>

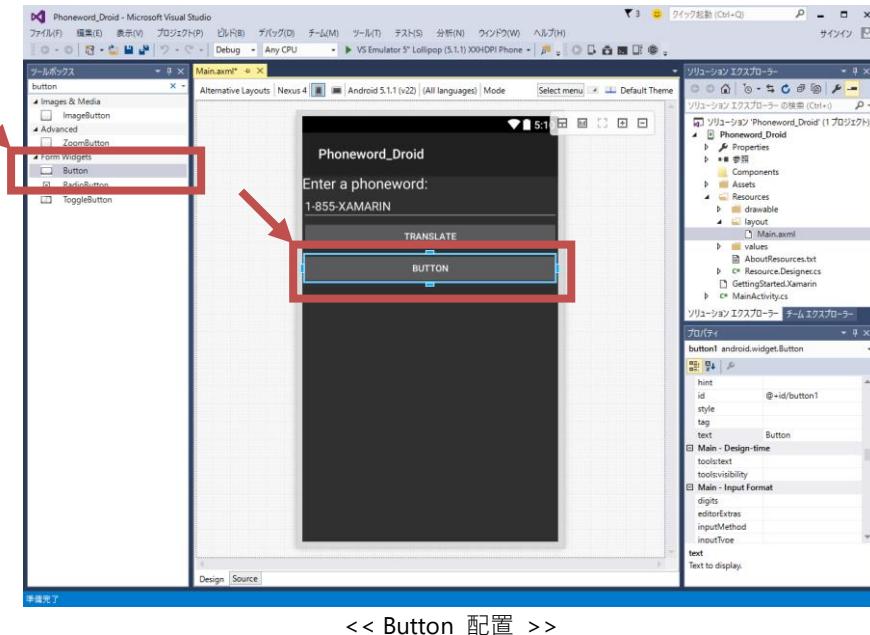
3.6 ツールボックスで [button] で再度検索します。[Button] コントロールをデザイナイン画面にドラッグして、[Text Field] コントロールの下に配置します。



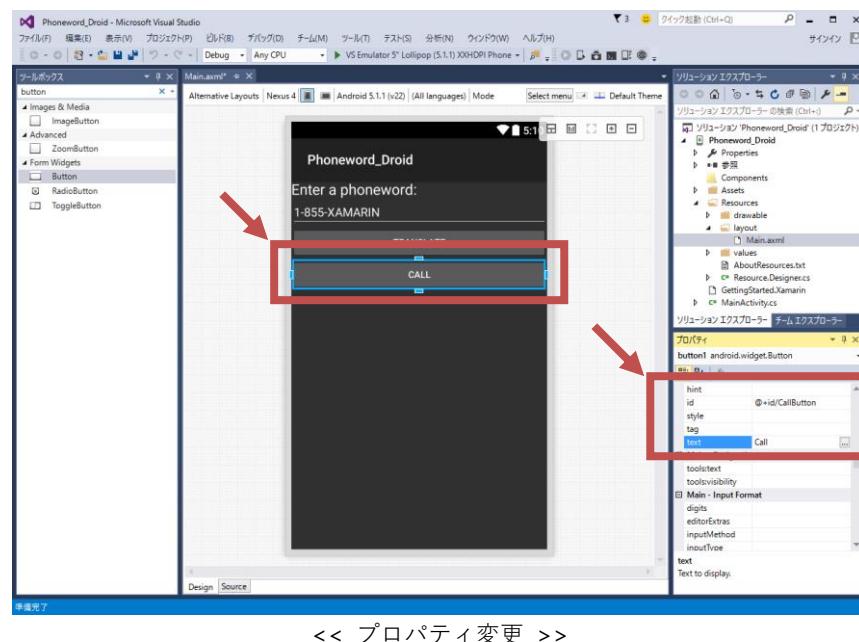
3.7 デザイン画面で[Button]を選択し、[プロパティ] ウィンドウを使用して [Button] の [Id] プロパティを「@+id/TranslateButton」に、[Text] プロパティを「Translate」にそれぞれ変更します。



3.8 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。



3.9 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。



4 コードの追加

- 4.1 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。
- 4.2 [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 [Shift + F6] キーを押して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.Android", 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);
        }
    }
}
```

- 4.3 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。
var phoneNumberText = FindViewById<EditText>(Resource.Id.PhoneNumberText);
var translateButton = FindViewById<Button>(Resource.Id.TranslateButton);
var callButton = FindViewById<Button>(Resource.Id.CallButton);
```

4.4 [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。
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.IsNullOrEmpty(translatedNumber))
    {
        callButton.Text = "Call";
        callButton.Enabled = false;
    }
    else
    {
        callButton.Text = "Call " + translatedNumber;
        callButton.Enabled = true;
    }
};
```

上記で `Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);` メソッドを呼び出していますが、これは [Phoneword (移植可能)] プロジェクトのコードです。Xamarin ではこのようにして PCL (Portable Class Library) のプロジェクトに .NET の共通化コードをまとめます。

4.5 `Dialog.Show()` メソッドで呼び出すダイアログは画面回転時に破棄されるため、メモリリークを発生させます。回避するために `OnCreateDialog` メソッドを `override` してその中でダイアログを `id` で呼び出します。初めに `class` のメンバー変数に `id` を追加します。

```
private const int DIALOG_ID_CALL = 1;
private string translatedNumber = string.Empty;
```

4.6 次に [CallButton] 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) =>
{
    // ShowDialog メソッドの引数に設定した定数 DIALOG_ID_CALL を指定します。
    ShowDialog(DIALOG_ID_CALL);
};
```

4.7 次に `showDialog` メソッドで呼び出される `Dialog` の `OnCreateDialog` を `override` して処理を記述します。以下のメソッドを `OnCreate` メソッドの下に追加します。

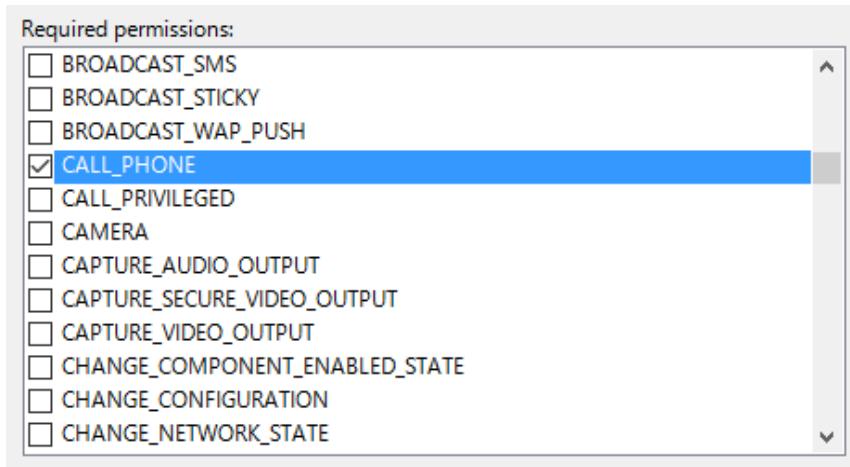
```
protected override Dialog OnCreateDialog(int id, Bundle args)
{
    switch (id)
    {
        case DIALOG_ID_CALL:
        {
            // "Call" ボタンがクリックされたら電話番号へのダイヤルを試みます。
            var callDialog = new AlertDialog.Builder(this);
            callDialog.SetMessage("Call?");
            callDialog.SetNeutralButton("Call", delegate
            {
                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。
            return callDialog.Create();
        }
        default:
            break;
    }

    return base.OnCreateDialog(id);
}

private void CallIntent()
{
    // 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);
}
```

4.8 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。

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



<< 権限設定 >>

- 4.10 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。

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。

5 その他の設定と動作確認

5.1 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。

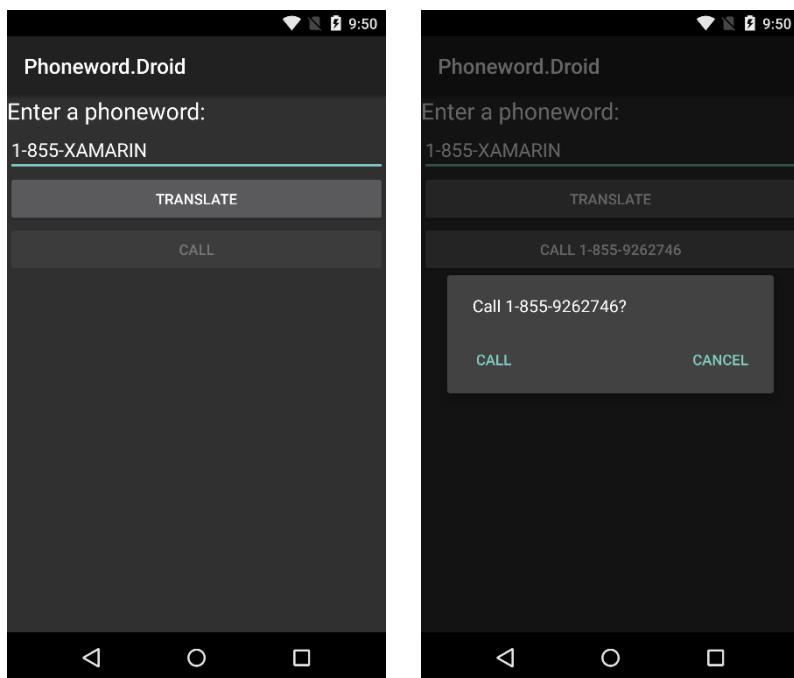
[MainActivity] クラスの上位にある [Label] を [Phoneword] に変更します。

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

5.2 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 マニフェスト > アプリケーション名] の [Phoneword.Android] を「Phoneword」に、[パッケージ名] を「com.example.Phoneword.Android」に、[アプリケーションアイコン] を「@drawable/icon」に変更し、[バージョン番号]、[バージョン名] を追加します。



- 5.3 最後に Android のエミュレーターまたは実機を使用して、アプリケーションをテストします。
- 5.4 ▶ 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] ダイアログが図のように表示されるのを確認してください。



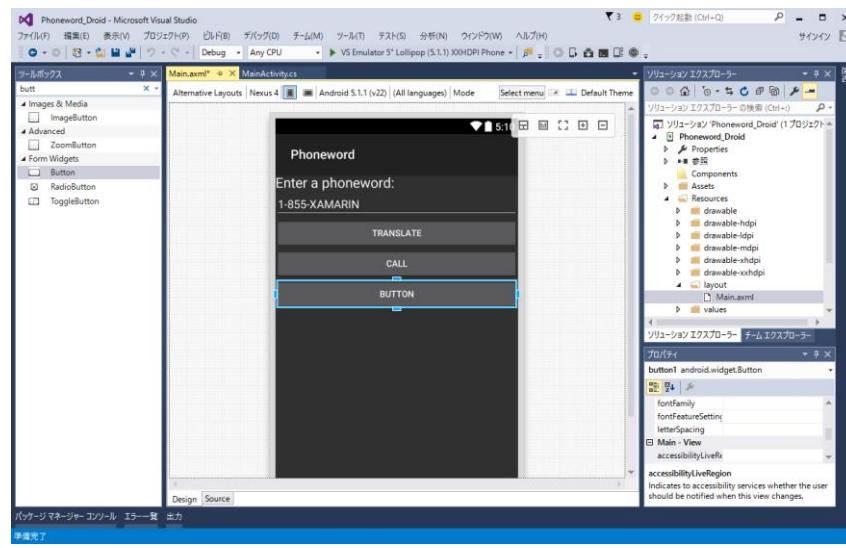
アプリケーションリストで Phoneword アプリケーションと設定したアイコンが表示されます。

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。

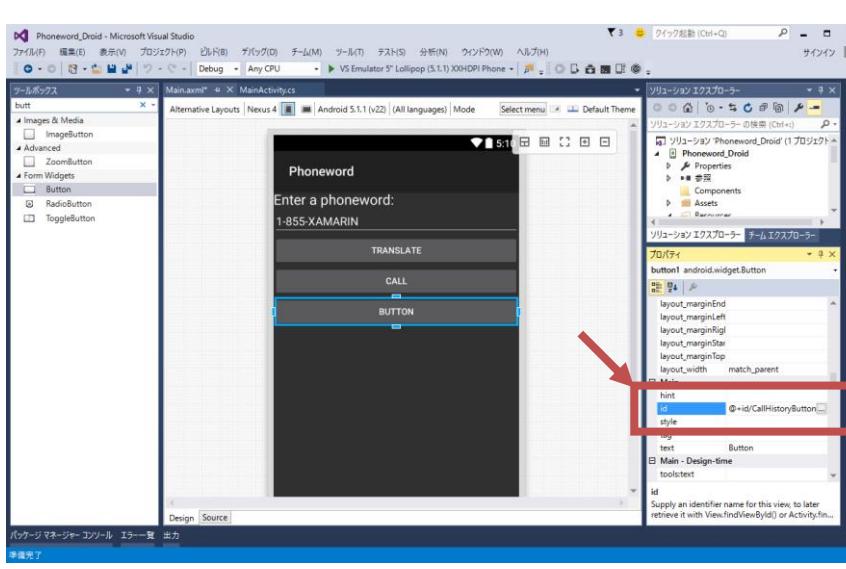
6 Hello, Android Multiscreen Quickstart

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。本章で完成したアプリケーションでは、2 番目の画面に通話履歴を表示します。

- 6.1 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。
- 6.2 [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。

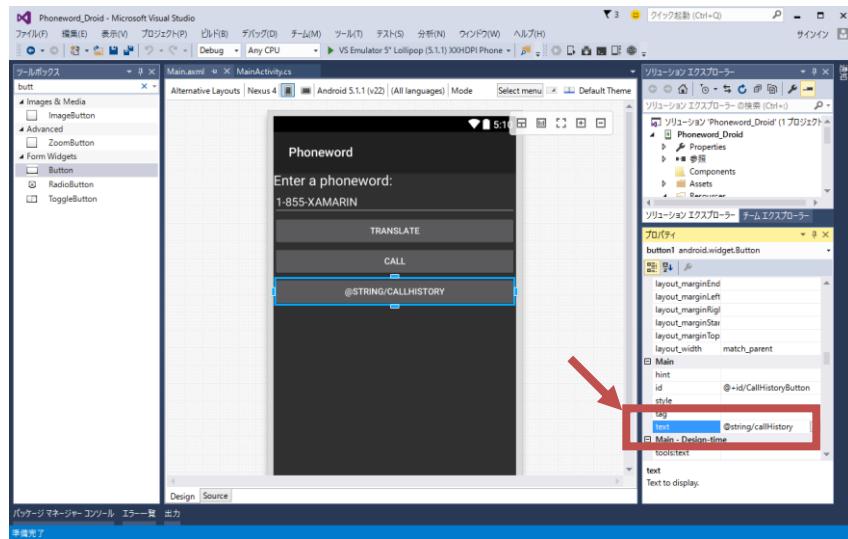


- 6.3 [プロパティ] でボタン [Id] を「@+id/CallHistoryButton」に変更します。



6.4 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 >>

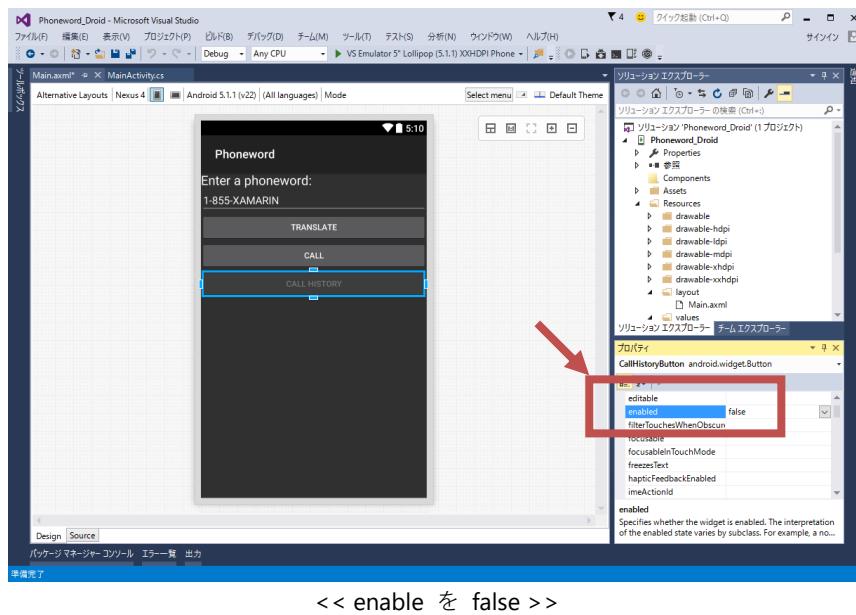
6.5 ソリューションエクスプローラーから [Resources] フォルダ以下の [values] フォルダを展開します。文字列のリソースファイル [Strings.xml] をダブルクリックして開きます。

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

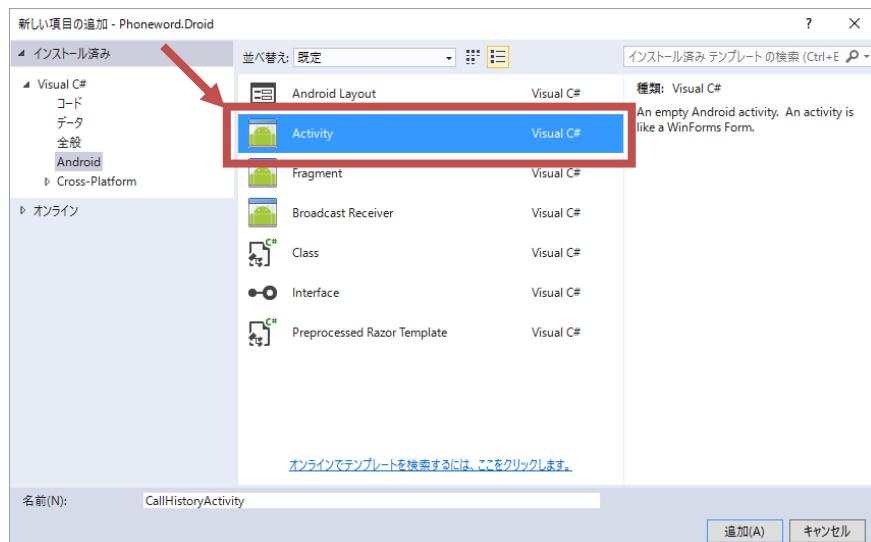
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="callHistory">Call History</string>
</resources>
```

[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）。

- 6.7 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。



- 6.8 2つ目の画面を作成します。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。
- 6.9 [新しい項目の追加] ダイアログから [Visual C#>Android>アクティビティ] を選択し、Activity に 「CallHistoryActivity.cs」と名前を付けます。



6.10 [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];
            thisListAdapter = 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。

6.11 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、ボタンを宣言します。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>();
    Button callHistoryButton;
    ...
    // OnCreate, etc.
}
```

6.12 [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。

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

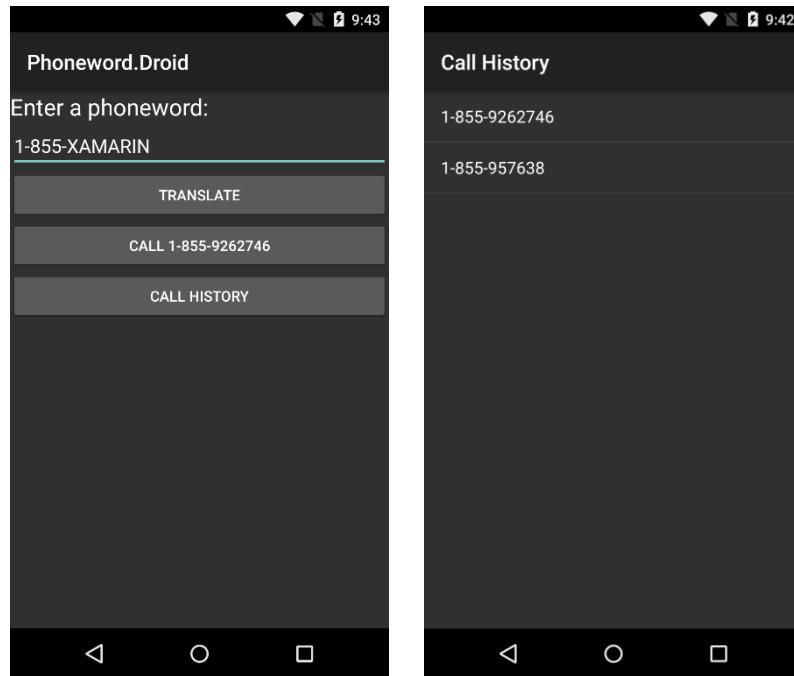
```
callHistoryButton.Click += (sender, e) =>
{
    var intent = new Intent(this, typeof(CallHistoryActivity));
    intent.PutStringArrayListExtra("phone_numbers", phoneNumbers);
    StartActivity(intent);
};
```

6.13 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;
    // 電話の Intent を呼び出します。
    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。

6.14 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。

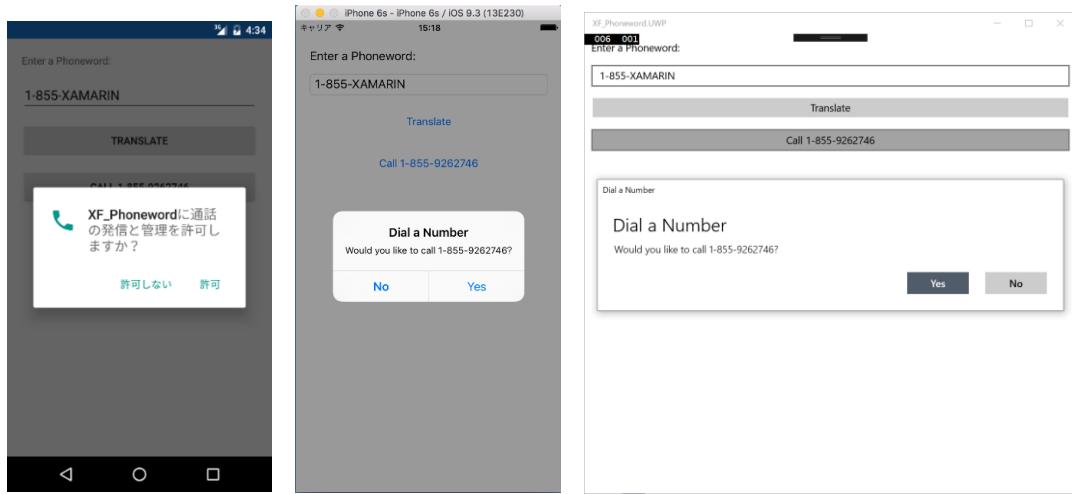


おめでとうございます。複数画面を操作する最初の Xamarin.Android アプリケーションが完成しました！

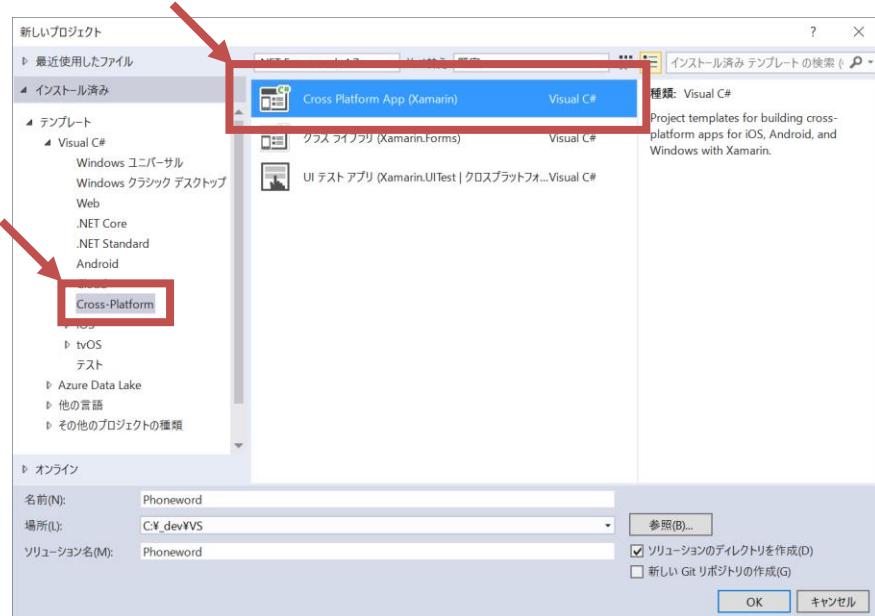
Xamarin.Forms で iOS／Android／UWP アプリ開発

1 Xamarin.Forms Quickstart

このセクションでは、これまでで作成した iOS／Android の 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。



- 1.1 Visual Studio を起動し、メニューから [ファイル>新規作成>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。
- 1.2 [新しいプロジェクト] 画面で、[Cross-Platform>Cross Platform App (Xamarin)] を選択し、名前を「XF_Phoneword」と付けます。

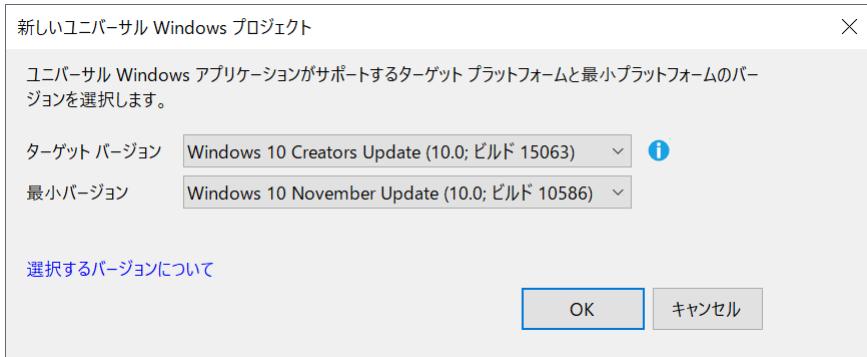


<<新しいプロジェクト>>

- 1.3 [空のアプリ]、[ネイティブ]、[ポータブルクラスライブラリ(PCL)] を選択し、[OK] をクリックします。



UWP バージョン確認のダイアログが出る場合はそのまま [OK] をクリックします。

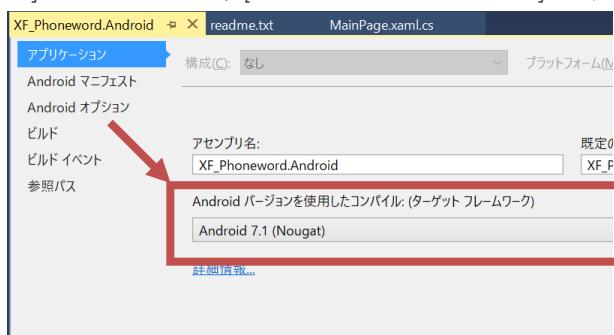


- 1.4 プロジェクトにコードを追加する前に、「Plugin.Permission」を追加します。Android 6.0 以降で、ランタイムパーミッションの設定をする必要があります。

2017 年 7 月時点の注意点です。

「Plugin.Permission」の最新版 2.0.1 は `Xamarin.Android.Support.Compat` の 25.3.1 以上を使用するため、Android 用の `Xamarin.Forms` に標準でインストールされるサポートライブラリの 23.3.0 と競合してしまいます。そのため、Android プロジェクトの `Xamarin.Forms` を一度削除し、25.3.1 を使用するように再追加する必要があります。

<https://www.nuget.org/packages/Xamarin.Forms/> に依存関係が記載されていますが、「`MonoAndroid 1.0`」がサポートライブラリの 23.3.0 を使用して、「`MonoAndroid 7.0`」がサポートライブラリの 23.3.0 以上を使用します。`MonoAndroid` のバージョンは、Android プロジェクトのターゲットフレームワークにより決まります。ターゲットフレームワークを変更するには、ソリューションエクスプローラーで `[XF_Phoneword.Android]` 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、[アプリケーション] をクリックして、[ターゲットフレームワーク] を最新にします。



[Android マニフェスト] をクリックして、[Target Android version] に最新を指定します。



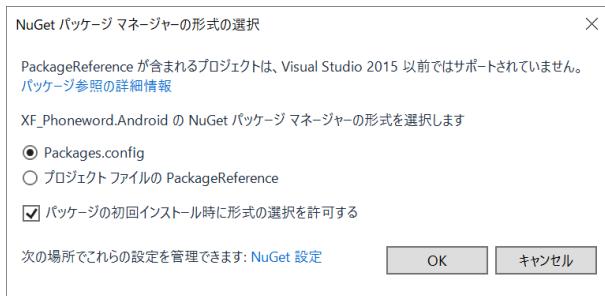
「XF_Phonework.Android」プロジェクトを右クリックして、[NuGet パッケージの管理] をクリックします。[インストール済み] タブから 「Xamarin.Forms」 を選択し、[オプション] の [依存関係の削除] と [依存関係が存在する場合でも強制的にアンインストールする] のチェックを付けて、アンインストールします。



[参照] タブで 「Xamarin.Forms」 を検索し、インストールする際に [オプション]>依存関係の動作>最高] を選択してインストールします。



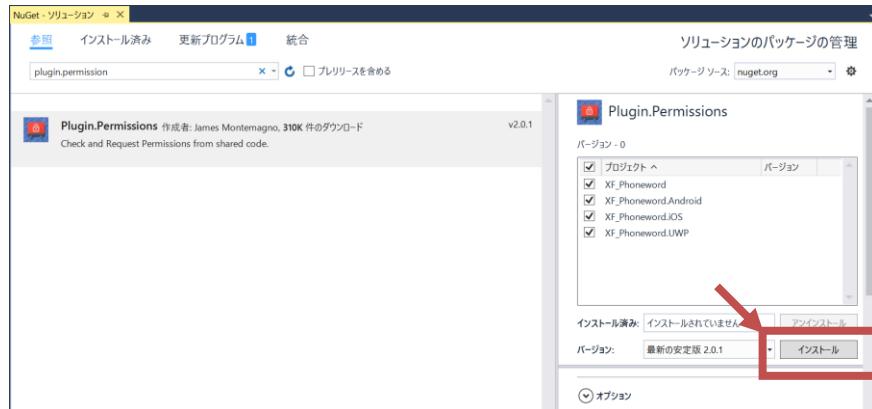
次のダイアログが表示された場合は、「Packages.config」を選択して [OK] をクリックしてください。



各種サポートライブラリの 25.3.1 がインストールされていることを確認してください。



- 1.5 [ソリューションエクスプローラー] で、ソリューションを右クリックして、[ソリューションの NuGet パッケージの管理] をクリックします。
- 1.6 [参照] タブを開き、「Plugin.Permissions」で検索し、Permission Plugin をインストールします。



コードを追加する準備が整いましたので、追加していきましょう。

2 UI、コードの追加

- 2.1 最初に iOS／Android の演習でも使用した電話番号を変換するコードを用意します。ソリューションエクスプローラーで、[XF_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。
- 2.2 [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。
- 2.3 [PhoneTranslator.cs] 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。

- 2.4 続いて View を編集します。 MainPage.xaml をダブルクリックして開きます。
- 2.5 [MainPage.xaml] 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 x:Class="XF_Phoneword.MainPage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">
<ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness"
        Android="10"
        WinPhone="10"
        iOS="10, 30, 10, 10" />
</ContentPage.Padding>
<ContentPage.Content>
    <StackLayout Spacing="15" VerticalOptions="FillAndExpand">
        <Label Text="Enter a Phoneword:" />
        <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
        <Button x:Name="translateButton"
            Clicked="OnTranslate"
            Text="Translate" />
        <Button x:Name="callButton"
            Clicked="OnCall"
            IsEnabled="false"
            Text="Call" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>

```

[Ctrl + S] を押して変更を保存します。

- 2.6 [MainPage.xaml] を展開し、[MainPage.xaml.cs] を開きます。
- 2.7 [MainPage.xaml.cs] 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実行されます。

```

using Plugin.Permissions;
using Plugin.Permissions.Abstractions;
using System;
using Xamarin.Forms;

namespace XF_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)
        {
            // Alert の選択を bool で取得します。
            var call = await DisplayAlert(
                "Dial a Number",
                "Would you like to call " + translatedNumber + "?",
                "Yes",
                "No");
            if (call)
            {
                try
                {
                    // PermissionPlugin を使用して Phone の Permission の Status を取得します。
                    var status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Phone);

                    // Permission 未設定の場合は確認ダイアログを表示し Permission の取得を確認します。
                
```

```
        if (status != PermissionStatus.Granted)
        {
            if (await
CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Phone))
            {
                await DisplayAlert("Need Permission", "It will get permission of phonecall", "OK");
            }

            var results = await CrossPermissions.Current.RequestPermissionsAsync(new[]
{ Permission.Phone });
            status = results[Permission.Phone];
        }

        // Permission が許可されていれば電話を掛けます。
        if (status == PermissionStatus.Granted)
        {
            var dialer = DependencyService.Get<IDialer>();
            if (dialer != null)
                dialer.Dial(translatedNumber);
        }
        else if (status != PermissionStatus.Unknown)
        {
            await DisplayAlert("Permission Denied", "Can not continue, try again.", "OK");
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.InnerException);
    }
}
}
```

[Ctrl]+[S] を押して変更を保存します。

IDialer の型が見つからないエラーが出ますが、この後の作業で作成します。

- 2.8 ソリューションエクスプローラーで、[XF_Phoneword (移植可能)] 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。
- 2.9 [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。
- 2.10 [IDialer.cs] を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 XF_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。[XF_Phoneword (移植可能)] プロジェクトを右クリックから、[ビルド] をクリックしてプロジェクトをビルドしておきます。

3 ネイティブ API を利用する Dependency Service の実装

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。

iOS

- 3.1 ソリューションエクスプローラーで、iOS プロジェクト [XF_Phoneword.iOS] 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。
- 3.2 [新しい項目の追加] 画面から、[Apple>Code>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。
- 3.3 [PhoneDialer.cs] 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 iOS 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。

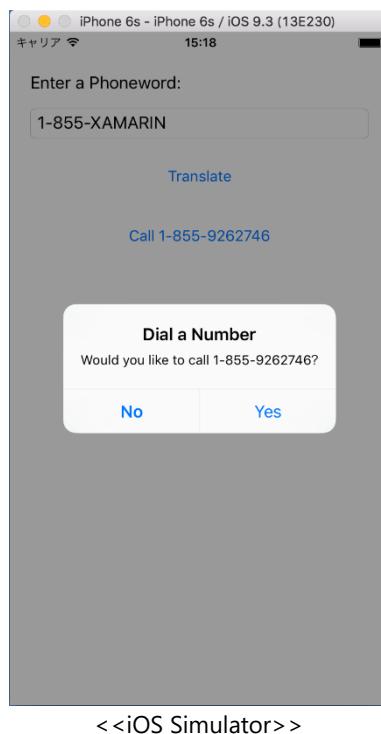
```
using Foundation;
using XF_Phoneword.iOS;
using UIKit;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]

namespace XF_Phoneword.iOS
{
    public class PhoneDialer : IDialer
    {
        public bool Dial(string number)
        {
            return UIApplication.SharedApplication.OpenUrl (
                new NSUrl ("tel:" + number));
        }
    }
}
```

[Ctrl + S] を押して変更を保存します。

- 3.4 [Phoneword.iOS] 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デバッグします。iOS Simulator では Permission の許可ダイアログは表示されません。



<<iOS Simulator>>

- 3.5 iOS の実機をお持ちで Apple Developer Program に加入している方は実機でも試してみてください。

Android

- 3.6 「Permission.Plugin」を使用するコードが必要になりますので、
MainActivity.cs を開き、`OnCreate` のオーバーライドの後に
`OnRequestPermissionsResult` のオーバーライドを追加します。

```
public override void OnRequestPermissionsResult(int requestCode, string[] permissions, Permission[] grantResults)
{
    Plugin.Permissions.PermissionsImplementation.Current.OnRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

- 3.7 ソリューションエクスプローラーで、[XF_Phoneword.Android] 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。
- 3.8 [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。
- 3.9 [PhoneDialer.cs] 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 XF_Phoneword.Android;
using System.Linq;
using Xamarin.Forms;

using Uri = Android.Net.Uri;

[assembly: Dependency(typeof(PhoneDialer))]

namespace XF_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;
        }
    }
}

```

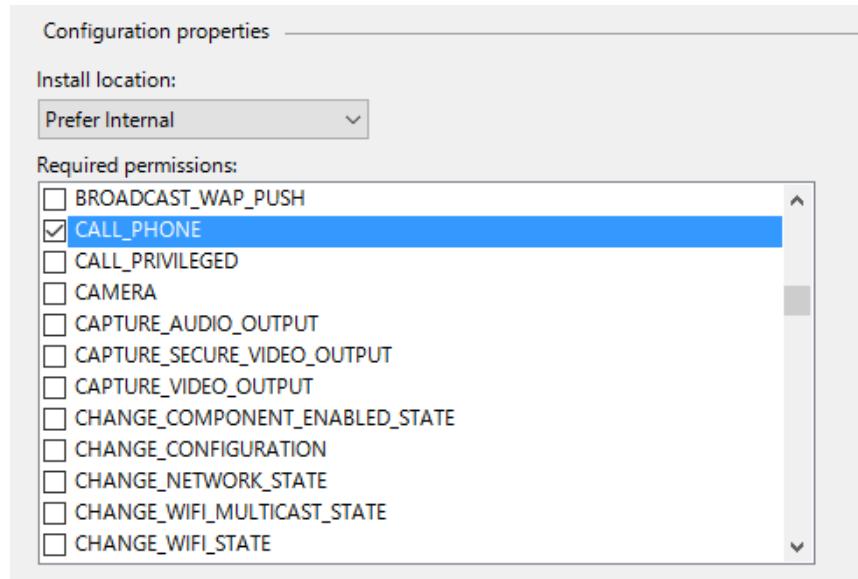
```
        }  
    }  
}
```

[Ctrl + S]を押して変更を保存します。

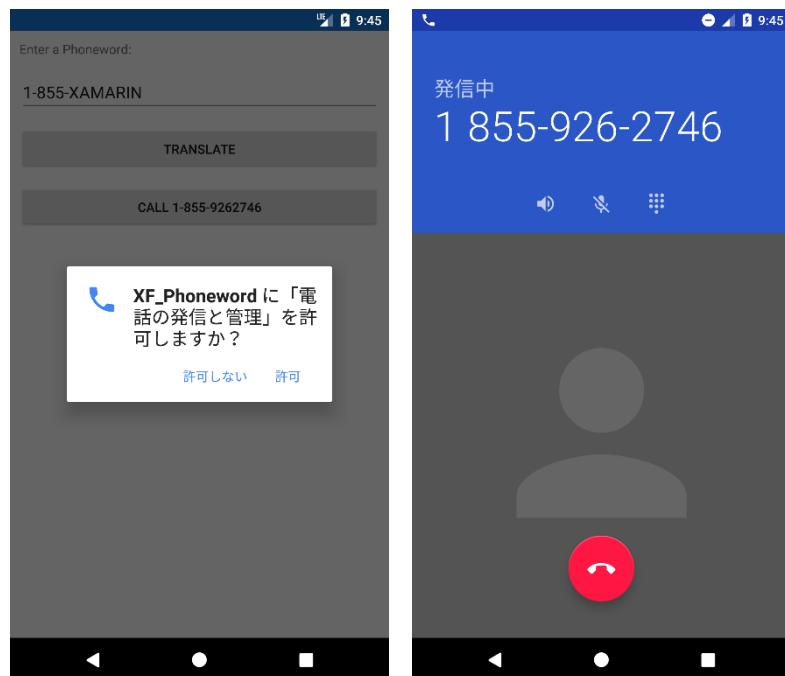
3.10 [Android マニフェスト] をクリックして、[アプリケーション名] に [XF_Phoneword]、[パッケージ名] に [com.example.XF_Phoneword.Android]などを指定します。



- 3.11 [必要なアクセス許可] 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。



- 3.12 [XF_Phoneword.Android] 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て、アプリケーションをデバッグしてみましょう。通話許可のダイアログが表示されることを確認してください。



UWP

- 3.13 ソリューションエクスプローラーで、[XF_Phoneword.UWP] 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。
- 3.14 [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。
- 3.15 [PhoneDialer.cs] 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 UWP で変換された電話番号で電話アプリを起動するメソッドを含んでいます。

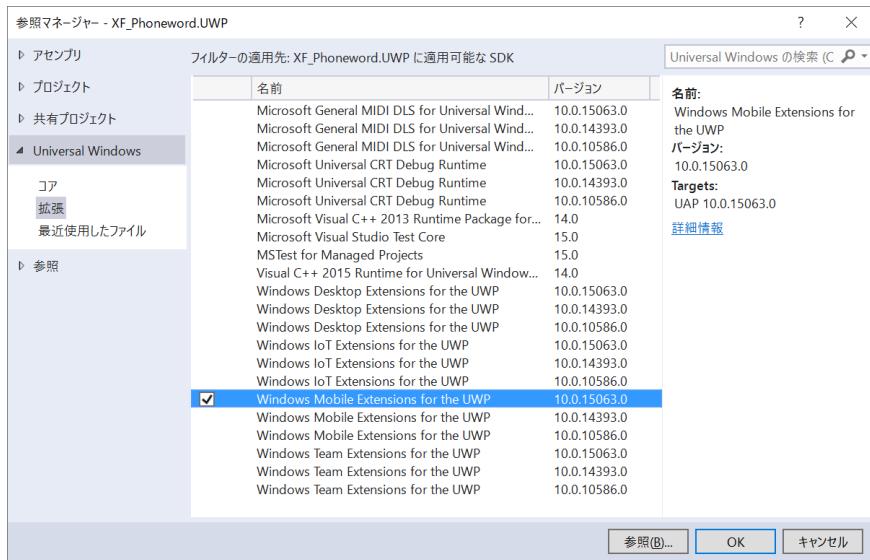
```
using Windows.ApplicationModel.Calls;
using XF_Phoneword.UWP;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]

namespace XF_Phoneword.UWP
{
    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。

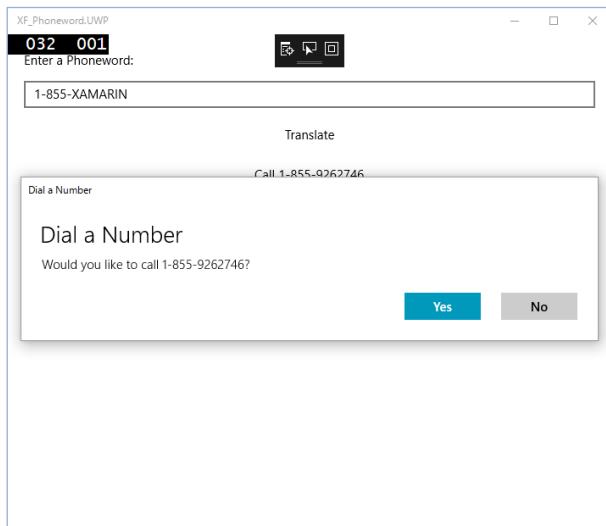
3.16 UWP で電話を掛ける (=Windows 10 Mobile の機能を使用する) には、UWP の拡張ライブラリを追加する必要があります。UWP プロジェクトの参照を右クリックして[参照の追加]を選択し、[Universal Windows > 拡張]から [Windows Mobile Extensions for the UWP] の最新版を追加します。



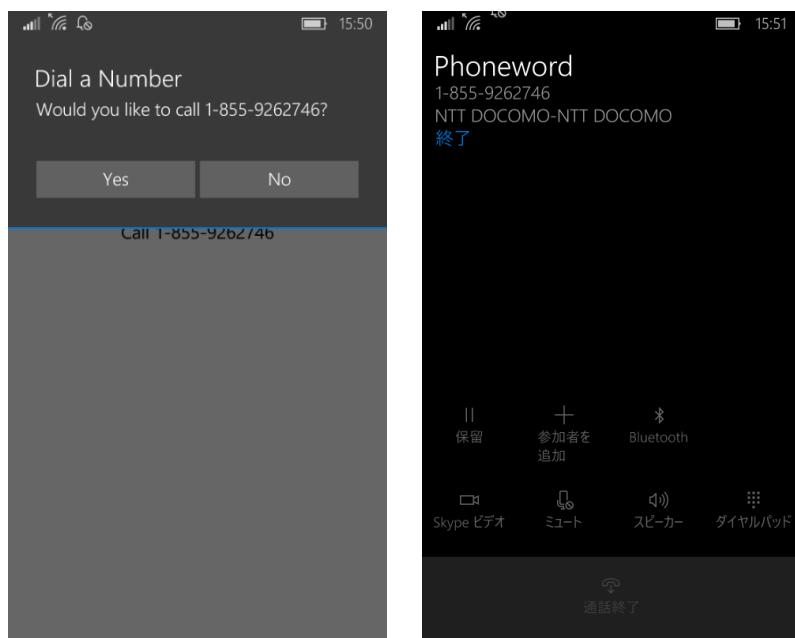
[IDialer] が見つからないというエラーが表示されることがあります。Xamarin.Forms のプロジェクト [XF_Phoneword] をビルドすると解決します。



3.17 [Phoneword.UWP] をスタートアッププロジェクトに設定し、[ソリューションプラットフォーム] から [x86] を選択し、[ローカルコンピューター] の状態で [▶] ボタンをクリックし、アプリケーションをデバッグします。

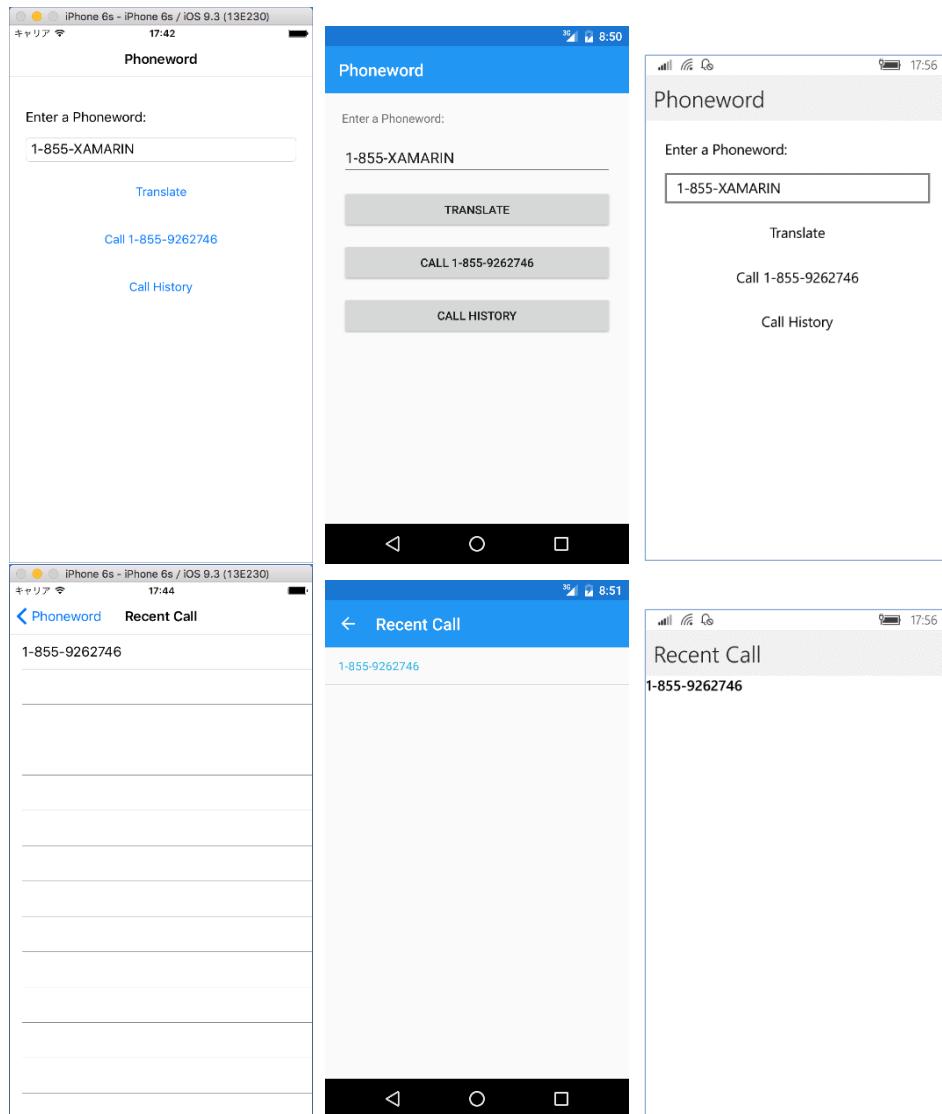


3.18 Windows 10 Mobile の実機をお持ちの方は [ソリューションプラットフォーム] で [ARM] を選択し、[Device] にデプロイ、デバッグしてみましょう。



4 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になります。



- 4.1 先ほどまで作業していた XF_Phoneword プロジェクトを開きます。
- 4.2 [App.xaml.cs] をダブルクリックして開きます。
- 4.3 `PhoneNumbers` プロパティを宣言し、App のコンストラク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。太字が変更部分です。

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;

using Xamarin.Forms;

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

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

```

[Ctrl+S] を押して変更を保存します。

- 4.4 [XF_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。
- 4.5 [新しい項目の追加] 画面から、[Visual C#>Xamarin.Forms>ContentPage (C#)] 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。
- 4.6 [CallHistoryPage.cs] 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 XF_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="XF_Phoneword.CallHistoryPage"
    Title="Recent Call">
<ContentPage.Content>
    <StackLayout VerticalOptions="FillAndExpand"
        Orientation="Vertical">
        <ListView ItemsSource="{x:Static local:App.PhoneNumbers}" />
    </StackLayout>
</ContentPage.Content>
</ContentPage>
```

4.7 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。ページタイトルを追加して、最初のページの [NavigationBar] にページタイトルが表示されるようにします。

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="XF_Phoneword.MainPage"
    Title="Phoneword">
```

4.8 [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 に遷移するために使用されます。コードの一部は次のようにになります。太字が変更部分です。

```
<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。

4.9 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。

4.10 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。太字が変更部分です。

```

using System;
using Xamarin.Forms;

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

        async void OnCall(object sender, EventArgs e)
        {
            ...

            if (status == PermissionStatus.Granted)
            {
                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());
        }
    }
}

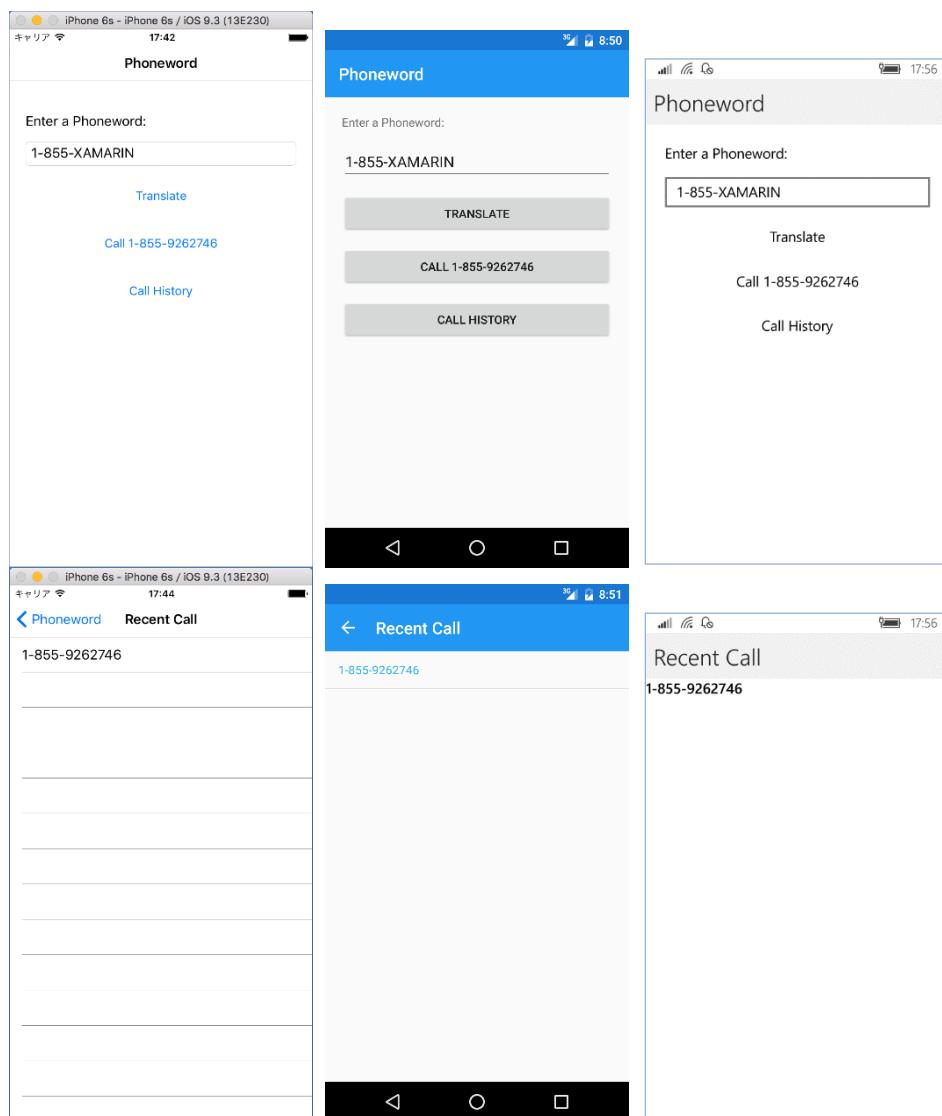
```

[Ctrl + S]を押して変更を保存します。

4.11 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。

4.12 Android、iOS、UWP をそれぞれ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デバッギングしましょう。



お疲れさまでした！Xamarin.Forms を使用した 2 画面のアプリが完成しました。

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 iOS／Android アプリの作成方法、Xamarin.Forms で iOS／Android／UWP アプリの作成方法を学びました。この体験を生かして次のアプリ開発にご活用いただければ幸いです。